MacAnova Reference Manual

Version 5.05

Technical Report #618
School of Statistics

University of Minnesota

Christopher Bingham Gary W. Oehlert

September 1997, Revised August 28, 2005

Contents

1 Introduction **19**

2 MacAnova Help File **21**

2.1 abs() . . . 21
2.2 acos() . . . 21
2.3 addchars() . . . 22
2.4 addhelpfile() . . . 24
2.5 addlines() . . . 25
2.6 adddatapath() . . . 26
2.7 addmacrofile() . . . 27
2.8 addpoints() . . . 28
2.9 addstrings() . . . 30
2.10 alltrue() . . . 31
2.11 anova() . . . 32
2.12 anovapred() . . . 34
2.13 anymissing() . . . 35
2.14 anytrue() . . . 35
2.15 appendnotes() . . . 36
2.16 arginfo_fun . . . 36
2.17 argvalue() . . . 37
2.18 arimahelp() . . . 39
2.19 arithmetic . . . 39
2.20 array() . . . 43
2.21 arrays . . . 44
2.22 asciisave() . . . 45
2.23 asin() . . . 46
2.24 asLong() . . . 47
2.25 assignment . . . 48
2.26 atan() . . . 52
2.27 atanh() . . . 52
2.28 attachnotes() . . . 53
2.29 autoreg() . . . 54
2.30 batch() . . . 56
2.31 bcprd() . . . 57

2.32 bin() . . . 58
2.33 bit_ops . . . 59
2.34 boxcox() . . . 61
2.35 boxplot() . . . 61
2.36 break . . . 63
2.37 breakall . . . 64
2.38 breakif() . . . 65
2.39 callback_fun . . . 66
2.40 carapace . . . 66
2.41 cat() . . . 68
2.42 cconj() . . . 69
2.43 cdivc() . . . 69
2.44 cdivcj() . . . 70
2.45 ceiling() . . . 71
2.46 cellstats() . . . 71
2.47 cft() . . . 72
2.48 changestr() . . . 73
2.49 cholesky() . . . 75
2.50 chplot() . . . 76
2.51 cimag() . . . 79
2.52 CLIPBOARD . . . 79
2.53 clipreaddata . . . 82
2.54 clipwritedat() . . . 83
2.55 cluster() . . . 84
2.56 cmplx() . . . 86
2.57 coefs() . . . 87
2.58 colplot() . . . 88
2.59 comments . . . 89
2.60 complex . . . 90
2.61 compnames() . . . 91
2.62 console() . . . 91
2.63 contrast() . . . 92
2.64 convolve() . . . 95
2.65 copyright . . . 95
2.66 cor() . . . 99
2.67 cos() . . . 99
2.68 cosh() . . . 100
2.69 cpolar() . . . 101
2.70 cprdc() . . . 101
2.71 cprdcj() . . . 102
2.72 creal() . . . 103
2.73 crect() . . . 104
2.74 ctoh() . . . 104
2.75 cumbeta() . . . 105

2.76 cumbin() . . . 106
2.77 cumchi() . . . 107
2.78 cumdunnett() . . . 108
2.79 cumF() . . . 110
2.80 cumgamma() . . . 111
2.81 cumnor() . . . 112
2.82 cumpoi() . . . 113
2.83 cumstu() . . . 113
2.84 cumstudrng() . . . 115
2.85 customize . . . 116
2.86 data_files . . . 118
2.87 DATAPATHS . . . 119
2.88 delete() . . . 120
2.89 describe() . . . 122
2.90 descriptive() . . . 125
2.91 design . . . 125
2.92 designhelp() . . . 127
2.93 det() . . . 128
2.94 diag() . . . 128
2.95 digamma() . . . 129
2.96 dim() . . . 130
2.97 dmat() . . . 131
2.98 dos_windows . . . 131
2.99 edit() . . . 133
2.100 eigen() . . . 135
2.101 eigenvals() . . . 136
2.102 else . . . 136
2.103 elseif . . . 137
2.104 enter() . . . 137
2.105 enterchars() . . . 138
2.106 equal() . . . 138
2.107 error() . . . 141
2.108 evaluate() . . . 141
2.109 exp() . . . 142
2.110 factor() . . . 143
2.111 fastanova() . . . 144
2.112 file_names . . . 146
2.113 files . . . 147
2.114 findfile() . . . 149
2.115 floor() . . . 149
2.116 for . . . 150
2.117 fprint() . . . 151
2.118 formatpval() . . . 151
2.119 fromclip() . . . 152

2.120 fwrite() . . . 153
2.121 getascii() . . . 153
2.122 getdata() . . . 154
2.123 getfilename() . . . 155
2.124 gethelp() . . . 157
2.125 gethistory() . . . 162
2.126 getkeywords() . . . 163
2.127 getlabels() . . . 164
2.128 getmacros() . . . 165
2.129 getnotes() . . . 166
2.130 getoptions() . . . 166
2.131 getseeds() . . . 168
2.132 gettime() . . . 168
2.133 getusage() . . . 169
2.134 glm . . . 170
2.135 glm_keys . . . 173
2.136 glmfit() . . . 175
2.137 glmpred() . . . 178
2.138 glmtable() . . . 179
2.139 goodfactors() . . . 182
2.140 grade() . . . 183
2.141 graphicshelp() . . . 184
2.142 graphs . . . 185
2.143 GRAPHWINDOWS . . . 189
2.144 graph_assign . . . 191
2.145 graph_border . . . 193
2.146 graph_files . . . 194
2.147 graph_keys . . . 196
2.148 graph_ticks . . . 201
2.149 halfnorm() . . . 203
2.150 haslabels() . . . 204
2.151 hasnotes() . . . 204
2.152 hconcat() . . . 205
2.153 hconj() . . . 205
2.154 hdivh() . . . 206
2.155 hdivhj() . . . 207
2.156 help() . . . 208
2.157 hft() . . . 210
2.158 himag() . . . 211
2.159 hist() . . . 212
2.160 hpolar() . . . 213
2.161 hprdh() . . . 214
2.162 hprdhj() . . . 215
2.163 hreal() . . . 215

2.164 hrect() . . . 216
2.165 htoc() . . . 216
2.166 hypot() . . . 217
2.167 if . . . 218
2.168 inforead() . . . 219
2.169 interrupt . . . 221
2.170 invbeta() . . . 221
2.171 invchi() . . . 222
2.172 invdunnett() . . . 223
2.173 invF() . . . 226
2.174 invgamma() . . . 226
2.175 invnor() . . . 227
2.176 invstu() . . . 228
2.177 invstudrng() . . . 229
2.178 ipf() . . . 230
2.179 isarray() . . . 232
2.180 ischar() . . . 233
2.181 isdefined() . . . 233
2.182 isfactor() . . . 234
2.183 isfunction() . . . 234
2.184 isgraph() . . . 235
2.185 islocked() . . . 236
2.186 islogic() . . . 236
2.187 ismacro() . . . 237
2.188 ismatrix() . . . 237
2.189 ismissing() . . . 238
2.190 isname() . . . 239
2.191 isnull() . . . 240
2.192 isnumber() . . . 241
2.193 isreal() . . . 242
2.194 isscalar() . . . 242
2.195 isstruc() . . . 243
2.196 isvector() . . . 244
2.197 keyvalue() . . . 245
2.198 keywords . . . 248
2.199 kmeans() . . . 249
2.200 labels . . . 250
2.201 launching . . . 255
2.202 length() . . . 259
2.203 lgamma() . . . 260
2.204 lineplot() . . . 260
2.205 list() . . . 262
2.206 listbrief() . . . 264
2.207 loadUser() . . . 265

2.208 locks . . . 265
2.209 lockvars() . . . 267
2.210 log() . . . 268
2.211 log10() . . . 268
2.212 log2() . . . 269
2.213 logic . . . 269
2.214 logistic() . . . 272
2.215 lowess() . . . 274
2.216 macintosh . . . 277
2.217 mac_classic . . . 278
2.218 macro() . . . 283
2.219 macro_files . . . 285
2.220 macro_syntax . . . 288
2.221 macroread() . . . 294
2.222 macros . . . 297
2.223 macrousage() . . . 299
2.224 macrowrite() . . . 299
2.225 makecols() . . . 301
2.226 makefactor() . . . 303
2.227 makestr() . . . 304
2.228 makesymbols() . . . 305
2.229 manova() . . . 306
2.230 match() . . . 308
2.231 mathhelp() . . . 310
2.232 matprint() . . . 311
2.233 matread() . . . 314
2.234 matread_file . . . 317
2.235 matrices . . . 324
2.236 matrix() . . . 327
2.237 matwrite() . . . 329
2.238 max() . . . 330
2.239 memory . . . 332
2.240 memoryinfo() . . . 333
2.241 min() . . . 334
2.242 modelinfo() . . . 336
2.243 models . . . 340
2.244 modelvars() . . . 344
2.245 more() . . . 347
2.246 Mouse() . . . 347
2.247 movavg() . . . 351
2.248 mulvarhelp() . . . 353
2.249 nameof() . . . 354
2.250 nbits() . . . 354
2.251 ncols() . . . 355

2.252 ncomps() . . . 356
2.253 ndims() . . . 356
2.254 next . . . 357
2.255 notes . . . 358
2.256 nrows() . . . 360
2.257 NULL . . . 360
2.258 number . . . 361
2.259 options . . . 364
2.260 outer() . . . 374
2.261 padto() . . . 375
2.262 partacf() . . . 376
2.263 paste() . . . 376
2.264 plot() . . . 379
2.265 poisson() . . . 381
2.266 polygamma() . . . 383
2.267 polyroot() . . . 384
2.268 popmodel() . . . 385
2.269 power() . . . 386
2.270 power2() . . . 387
2.271 precedence . . . 388
2.272 predtable() . . . 391
2.273 primefactors() . . . 394
2.274 print() . . . 395
2.275 printoptions() . . . 399
2.276 probit() . . . 400
2.277 prod() . . . 402
2.278 propinterval() . . . 404
2.279 proptest() . . . 405
2.280 pushmodel() . . . 405
2.281 putascii() . . . 407
2.282 qr() . . . 408
2.283 quitting . . . 409
2.284 rank() . . . 409
2.285 rankits() . . . 411
2.286 rational() . . . 412
2.287 rbin() . . . 413
2.288 read() . . . 414
2.289 readcols() . . . 415
2.290 readdata() . . . 416
2.291 redo() . . . 418
2.292 regcoefs() . . . 419
2.293 regpred() . . . 420
2.294 regress() . . . 421
2.295 regresshelp() . . . 423

2.296 releigen() . . . 424
2.297 releigenvals() . . . 425
2.298 rename() . . . 425
2.299 rep() . . . 426
2.300 replacestr() . . . 427
2.301 restore() . . . 428
2.302 restorenames() . . . 430
2.303 return . . . 431
2.304 reverse() . . . 432
2.305 rft() . . . 433
2.306 rnorm() . . . 433
2.307 robust() . . . 434
2.308 rotate() . . . 436
2.309 rotation() . . . 437
2.310 round() . . . 438
2.311 rowplot() . . . 439
2.312 rpoi() . . . 439
2.313 rsample() . . . 440
2.314 rsolve() . . . 441
2.315 run() . . . 441
2.316 runi() . . . 442
2.317 samplesize() . . . 443
2.318 save() . . . 444
2.319 scalars . . . 447
2.320 screen() . . . 448
2.321 secoefs() . . . 451
2.322 select() . . . 453
2.323 sethistory() . . . 454
2.324 setlabels() . . . 455
2.325 setodometer() . . . 455
2.326 setoptions() . . . 458
2.327 setseeds() . . . 460
2.328 shapeof() . . . 460
2.329 shell() . . . 461
2.330 showplot() . . . 463
2.331 sin() . . . 464
2.332 sinh() . . . 465
2.333 solve() . . . 465
2.334 sort() . . . 466
2.335 split() . . . 467
2.336 spool() . . . 469
2.337 sqrt() . . . 470
2.338 stemleaf() . . . 470
2.339 strconcat() . . . 471

2.340 stringplot() . . . 472
2.341 structure() . . . 474
2.342 structures . . . 476
2.343 subscripts . . . 479
2.344 sum() . . . 482
2.345 svd() . . . 484
2.346 swp() . . . 486
2.347 syntax . . . 487
2.348 t() . . . 495
2.349 t2int() . . . 496
2.350 t2val() . . . 497
2.351 tabs() . . . 498
2.352 tan() . . . 500
2.353 tanh() . . . 501
2.354 tek() . . . 501
2.355 tekx() . . . 502
2.356 time_series . . . 502
2.357 tint() . . . 505
2.358 toclip() . . . 505
2.359 toeplitz() . . . 506
2.360 trace() . . . 507
2.361 transformations . . . 507
2.362 transpose() . . . 509
2.363 trideigen() . . . 510
2.364 trilower() . . . 511
2.365 triunpack() . . . 512
2.366 triupper() . . . 513
2.367 tserhelp() . . . 514
2.368 tinterval() . . . 515
2.369 ttest() . . . 515
2.370 tval() . . . 515
2.371 twotailt() . . . 516
2.372 typeof() . . . 516
2.373 unique() . . . 517
2.374 unix . . . 519
2.375 unlockvars() . . . 521
2.376 unwind() . . . 522
2.377 usage() . . . 522
2.378 userfunhelp() . . . 523
2.379 user_fun . . . 524
2.380 variables . . . 525
2.381 varnames() . . . 527
2.382 vboxplot() . . . 528
2.383 vconcat() . . . 529

2.384 vecread() . . . 530
2.385 vecread_file . . . 536
2.386 vecread_keys . . . 538
2.387 vector() . . . 542
2.388 vectors . . . 544
2.389 vt() . . . 545
2.390 vtx() . . . 545
2.391 while . . . 545
2.392 workspace . . . 547
2.393 write() . . . 548
2.394 writedata() . . . 548
2.395 wtanova() . . . 550
2.396 wtmanova() . . . 550
2.397 wtregress() . . . 550
2.398 xrows() . . . 551
2.399 xvariables() . . . 552
2.400 yates() . . . 553
2.401 yulewalker() . . . 554
2.402 zinterval() . . . 555
2.403 ztest() . . . 555

3 Arima Macros Help File 557
3.1 acfarma() . . . 557
3.2 arima() . . . 558
3.3 arimahelp() . . . 563
3.4 arimares() . . . 564
3.5 ARSIGN . . . 565
3.6 detarma() . . . 565
3.7 hannriss() . . . 566
3.8 innovations() . . . 567
3.9 innovest() . . . 569
3.10 MASIGN . . . 570
3.11 moveoutroots() . . . 572
3.12 neg2logLarma() . . . 573
3.13 rhatcovar() . . . 575
3.14 rhatvar() . . . 575
3.15 specarma() . . . 576

4 Design Macros Help File 579
4.1 aberration2() . . . 579
4.2 aliases2() . . . 579
4.3 aliases3() . . . 580
4.4 all3anova() . . . 582
4.5 all4anova() . . . 583

4.6 allaliases2() . . . 584
4.7 boxcoxvec() . . . 584
4.8 buildfactor() . . . 586
4.9 choosedef2() . . . 587
4.10 choosegen2() . . . 587
4.11 confound2() . . . 589
4.12 confound3() . . . 590
4.13 doconfound2() . . . 591
4.14 doff2() . . . 592
4.15 ems() . . . 592
4.16 ffdesign2() . . . 597
4.17 findncp() . . . 597
4.18 findpower() . . . 598
4.19 findsampsize() . . . 599
4.20 interactplot() . . . 600
4.21 interblock() . . . 602
4.22 mixed() . . . 603
4.23 pairwise() . . . 605
4.24 quadmax() . . . 607
4.25 randsign() . . . 608
4.26 randt2() . . . 609
4.27 randt() . . . 610
4.28 reml() . . . 611
4.29 rscanon() . . . 613
4.30 sidebyside() . . . 614
4.31 stdordlabels() . . . 615
4.32 typeIIIss() . . . 615
4.33 varcomp() . . . 615
4.34 yatesplot() . . . 617

5 Graphics Macros Help File 619
5.1 bargraph() . . . 619
5.2 boxplot5num() . . . 620
5.3 colplot() . . . 621
5.4 contour() . . . 621
5.5 contourplot() . . . 623
5.6 ellipse() . . . 624
5.7 findcontour() . . . 625
5.8 graphicshelp() . . . 627
5.9 hist() . . . 627
5.10 news . . . 629
5.11 panelhist() . . . 629
5.12 panelplot() . . . 631
5.13 panel_graphs . . . 632

5.14 piechart() . . . 633
5.15 plotmatrix() . . . 635
5.16 plotpanes() . . . 637
5.17 plotresids() . . . 639
5.18 rowplot() . . . 639
5.19 sampcdf() . . . 640
5.20 vboxplot() . . . 641

6 Mathematical Macros Help File **643**
6.1 bfs() . . . 643
6.2 binom() . . . 644
6.3 blockdmat() . . . 644
6.4 broyden() . . . 645
6.5 cdiag() . . . 645
6.6 ceigen() . . . 646
6.7 chebcoefs() . . . 647
6.8 cjtranspose() . . . 647
6.9 cmatmultc() . . . 648
6.10 continfrac() . . . 648
6.11 csolve() . . . 649
6.12 csubscr() . . . 650
6.13 ctrace() . . . 651
6.14 ctranspose() . . . 651
6.15 dfp() . . . 652
6.16 economize() . . . 652
6.17 factorial() . . . 653
6.18 factors() . . . 653
6.19 i0() . . . 654
6.20 i1() . . . 655
6.21 invchebcoefs() . . . 655
6.22 invertseries() . . . 655
6.23 kronecker() . . . 656
6.24 mathhelp() . . . 657
6.25 matsqrt() . . . 657
6.26 minimizer() . . . 658
6.27 moorepenrose() . . . 660
6.28 levmar() . . . 661
6.29 neldermead() . . . 666
6.30 orthopoly() . . . 668
6.31 partitions() . . . 669
6.32 printfactors() . . . 670
6.33 qrdcomp() . . . 671

7 Multivariate Macros Help File **673**
7.1 backstep() . . . 673
7.2 chiqqplot() . . . 674
7.3 compf() . . . 675
7.4 covar() . . . 676
7.5 daentervar() . . . 677
7.6 daremovevar() . . . 678
7.7 dasteplook() . . . 679
7.8 dastepsetup() . . . 680
7.9 _DASTEPSTATE . . . 681
7.10 dastepstatus() . . . 683
7.11 discrim() . . . 684
7.12 discrimquad() . . . 685
7.13 distcomp() . . . 686
7.14 facanal() . . . 687
7.15 forstep() . . . 690
7.16 glscrit() . . . 692
7.17 glsfactor() . . . 692
7.18 glsresids() . . . 694
7.19 goodfit() . . . 695
7.20 groupcovar() . . . 695
7.21 hotellval() . . . 696
7.22 hotell2val() . . . 697
7.23 mlcrit() . . . 697
7.24 jackknife() . . . 698
7.25 mulvarhelp() . . . 699
7.26 mvngen() . . . 700
7.27 probsquad() . . . 700
7.28 rmvnorm() . . . 701
7.29 standardize() . . . 701
7.30 stepgls() . . . 702
7.31 stepml() . . . 703
7.32 stepuls() . . . 705
7.33 ulscrit() . . . 706
7.34 ulsfactor() . . . 707
7.35 ulsresids() . . . 709

8 Regression Macros Help File **711**
8.1 anovapred() . . . 711
8.2 betalimits() . . . 712
8.3 entervar() . . . 713
8.4 estimlimits() . . . 714
8.5 nlreg() . . . 716
8.6 predlimits() . . . 721

8.7 regcoefs() . . . 722
8.8 regresshelp() . . . 723
8.9 regs() . . . 723
8.10 removevar() . . . 724
8.11 resid() . . . 725
8.12 resvsindex() . . . 726
8.13 resvsrankits() . . . 728
8.14 resvsyhat() . . . 729
8.15 steplook() . . . 731
8.16 stepsetup() . . . 731
8.17 stepstatus() . . . 732
8.18 testbeta() . . . 733
8.19 testestim() . . . 734
8.20 yhat() . . . 734

9 Time Series Macros Help File **737**
9.1 arspectrum() . . . 737
9.2 autocor() . . . 738
9.3 autocov() . . . 739
9.4 bandwidth . . . 741
9.5 burg() . . . 743
9.6 compfa() . . . 744
9.7 complex_data . . . 746
9.8 complex_fun . . . 747
9.9 compza() . . . 751
9.10 costaper() . . . 753
9.11 crosscor() . . . 754
9.12 crosscov() . . . 755
9.13 crsspectrum() . . . 757
9.14 detrend() . . . 758
9.15 dpss() . . . 759
9.16 evalpoly() . . . 760
9.17 ffplot() . . . 761
9.18 fourier . . . 762
9.19 gettsmacros() . . . 764
9.20 hermitian . . . 765
9.21 multitaper() . . . 766
9.22 spectrum() . . . 767
9.23 testnfreq() . . . 768
9.24 tsplot() . . . 769

10 Graphical User Interface Help File **773**
10.1 alert() . . . 773
10.2 doguihelp() . . . 773

10.3 getdirname() . . . 773
10.4 getmenubar() . . . 774
10.5 guiabout() . . . 774
10.6 guianova() . . . 774
10.7 guiboxplot() . . . 774
10.8 guifilepath() . . . 776
10.9 guihelp() . . . 776
10.10 guihist() . . . 777
10.11 guilistxml() . . . 778
10.12 guilistctrl() . . . 778
10.13 guilistdlg() . . . 778
10.14 guintrctplt() . . . 780
10.15 guipatterned() . . . 781
10.16 guiplotresid() . . . 782
10.17 guirandom() . . . 783
10.18 guireadfile() . . . 783
10.19 guirsample() . . . 784
10.20 guitypein() . . . 784
10.21 setmenubar() . . . 784

11 User Function Help File **789**
11.1 arginfo_fun . . . 789
11.2 c_macros . . . 793
11.3 callback_fun . . . 800
11.4 compile_dos . . . 805
11.5 compile_mac . . . 806
11.6 compile_unix . . . 810
11.7 compile_win . . . 810
11.8 loadUser . . . 811
11.9 type_codes . . . 812
11.10 User . . . 813
11.11 userfunhelp() . . . 817
11.12 user_fun . . . 817

12 Search Key Tables **823**

Chapter 1

Introduction

This document is provided as a detailed reference to the commands and functions distributed with MacAnova. The sections here are slightly reformatted versions of the material that is available on-line in MacAnova using the `help()` command. Thus, for example, `help(anova)` in MacAnova prints the same material that is given here.

The MacAnova distribution includes five help files (MacAnova.hlp.txt, Design.hlp.txt, Tser.hlp.txt, Userfun.hlp.txt, and Gui.hlp.txt) and the help sections of five macro files (Arima.mac.txt, Graphics.mac.txt, Math.mac.txt, Mulvar.mac.txt, and Regress.mac.txt). Help topics from these files are arranged in ten sections below. Some topics in MacAnova.hlp.txt have not been included here as they are primarily concerned with news about changes in MacAnova; these sections are: `macanova`, `macanova3`, `news`, `old`, and `updates`. The index topics in other files have also been deleted here.

All MacAnova help files can contain search keys for finding commands. For example, "plotting" is a search key that can be used to find help topics related to plotting via the command `help(key:"plotting")`. Chapter 12 contains tables for each help file that list the help topics related to each search key.

This document is not an introduction to or explanation of how MacAnova itself works. For that, consult the *MacAnova User's Guide* by Oehlert and Bingham.

Chapter 2

MacAnova Help File

This Chapter contains MacAnova help topics that are in the standard MacAnova help file. The material here is a reformatting of that help file.

2.1 abs()

Usage:

```
abs(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                  Usage
abs(x) returns the absolute values of the elements of x, when x is a
REAL scalar, vector, matrix or array.  The result has the same shape as
x.

When any element of x is MISSING, so is the corresponding element of
abs(x) and a warning message is printed.

                            Structure argument
When x is a structure, all of whose non-structure components are REAL,
abs(x) is a structure of the same shape and with the same component
names as x, with each non-structure component transformed by abs().

                             Cross reference
See topic 'transformations' for more information on abs().
```

2.2 acos()

Usage:

```
acos(x [, degrees:T or radians:T or cycles:T]), x REAL or a structure
  with REAL components value in radians (default), cycles, or degrees as
  specified by option "angles" or the optional keyword
```

Keywords: transformations

```
                                  Usage
acos(x) computes the inverse cosines of the elements of x, where x is a
REAL scalar, vector, matrix or array.  The result has the same shape as
x.  cos(acos(x)) should be the same as x except for rounding error.

The units of the result are radians, degrees or cycles as determined by
the value of option 'angles'.  The default is radians.  See subtopic
'options:"angles"'.

acos(x, radians:T), acos(x, degrees:T), acos(x, cycles:T) return results
in the indicated units, regardless of the value of option 'angles'.

When any element of x is MISSING or is above 1 or below -1, the
corresponding element of the result is MISSING and a warning message is
printed.

                            Structure argument
When x is a structure, all of whose non-structure components are REAL,
acos(x [,UNITS:T]), where UNITS is one of 'radians', 'degrees' or
'cycles', is a structure of the same shape and with the same component
names as x with each non-structure component transformed by acos().

                                  Example
  Cmd> vector(acos(.5),acos(.5,degrees:T),acos(.5,cycles:T))
  (1)      1.0472          60     0.16667

                              Cross reference
See topic 'transformations' for more information on acos(), including
its use with a CHARACTER argument.
```

2.3 addchars()

Usage:

```
addchars([Graph,] x,y [,symbols:c] [,lines:T,impulse:T]\
  [, graphics keyword phrases])
addchars([Graph] [,x,y [,symbols:c]], keys:str), str a structure whose
  components names match graphics keywords.
```

Keywords: plotting

```
                                  Usage
addchars(x,y,symbols:c) is equivalent to chplot(x,y,symbols:c,add:T).
It adds character labeled points to the plot in LASTPLOT, displays the
plot, and updates LASTPLOT with the new information.  For compatibility
with past versions, the use of keyword 'symbols' is optional.

Arguments x, y, and c are as for chplot() and the points are labeled the
same way as is done by chplot().  When 'symbols:c' is omitted, the same
default is used as for chplot().  It is not an error when x or y is
NULL; a warning message is printed and no plotting occurs.
```

It is an error if LASTPLOT does not exist.

Whenever LASTPLOT is updated, the appropriate component of GRAPHWINDOWS is made identical to LASTPLOT. See topic 'GRAPHWINDOWS'.

Graph Variable Argument

addchars(Graph,x,y,symbols:c) or chplot(Graph,x,y,symbols.c,add:T) displays GRAPH variable Graph with the addition of character labeled points, saving the modified plot in LASTPLOT. Graph is not changed (unless it is LASTPLOT).

Keywords 'keep' and 'show'

addchars(x,y,symbols:c,keep:F) suppresses any change to LASTPLOT. The appropriate element of GRAPHWINDOWS is set to NULL.

addchars(x,y,symbols:c,show:F) suppresses immediate display of the modified graph or change to GRAPHWINDOWS but updates LASTPLOT. This is useful when you are building a complex graph in stages using addchars(), addlines(), addstrings(), or addpoints(). When you are done, simply type showplot(). You can't use both show:F and keep:F.

Keywords

Keywords 'dumb', 'lines', 'linetype', 'thickness', 'impulse', 'xmin', 'xmax', 'ymin', 'ymax', 'logx', 'logy', 'xlab', 'ylab', 'title', 'xaxis', 'yaxis', 'borders', 'ticks', 'xticks', 'yticks', 'xticklen', 'yticklen', 'xticklabs', 'yticklabs', 'height', 'width', 'pause', 'silent' and 'notes' may be used as for other plotting commands. See topics 'graph_keys', 'graph_border' and 'graph_keys'

Keyword 'keys'

addchars([Graph,] keys:structure(x:x,y:y,symbols:c [other keyword phrases)) is equivalent to addchars([Graph,] x,y,symbols:c [other keyword phrases]). See topic 'graph_keys' for details.

When option 'dumbplot' has been set False (see subtopic 'options:"dumbplot"'), the plot will be a low resolution plot unless 'dumb:F' is an argument.

Missing value for 'xmin', 'xmax', 'ymin' and 'ymax'

A value of MISSING for any of xmin, xmax, ymin or ymax (for example, xmin:?) forces determination of an extreme value from the current data and data already in the graph.

Keyword Use

New labels and title may be set only by keywords 'xlab', 'ylab' and 'title'. That is addchars(newx:x, newy:y,symbols:c) has no effect on the axis labels of the graph being modified.

addchars(x,y,symbols:c,add:F, ...) is equivalent to chplot(x,y, symbols:c, ...) except that LASTPLOT must be defined.

Cross references

See topic 'graph_assign' for information on another way to add data and

```
other information to a plot.

See topic 'graphs' for general information on plots and on variable
LASTPLOT.  See topic 'graph_keys' for information on keywords.  See
topic 'graph_files' for information on writing a graph to a file.
```

2.4 addhelpfile()

Usage:

```
addhelpfile(names [,T]), names a CHARACTER scalar or vector of length 2
```

Keywords: general, files

```
                                  Usage
addhelpfile(fileName) adds fileName at the beginning of CHARACTER vector
HELPFILES which contains the names of files to be searched for help.
fileName must be a quoted string or CHARACTER scalar.  Because fileName
is added at the beginning of HELPFILES, the file will usually be the
first one searched.  In addition, the name "index" is added at the start
of CHARACTER vector HELPINDICES.

addhelpfile(fileName,T) does the same except the file name is added at
the end of HELPFILES so the file will be searched last.  "index" is
added at the end of HELPINDICES.

addhelpfile(vector(fileName,indexName) [,T]) does the same, except
that CHARACTER scalar indexName is added at the start or end of
HELPINDICES instead of "index".

When HELPFILES does not already exist, addhelpfile() creates it.

                          Directories searched
When fileName is a simple file name containing no directory or folder
information (for example, "survival.hlp" but not "./survival.hlp" or
":survival.hlp"), the file is first assumed to be in the default
directory.  If not found there, MacAnova looks for it in the folders or
directories listed in variable DATAPATHS. See topics 'DATAPATHS',
adddatapath(), 'file_names', 'files'.

                            Windowed versions
On Windowed versions of MacAnova, you may combine addhelpfile() with
getfilename() to choose the file interactively;  for example,
addhelpfile(getfilename() [,T])    This will include the complete
path name (file name with directory information) of the selected
file in HELPFILES.

                                 Examples
Example:
  Cmd> addhelpfile(vector("survival.hlp","survival_index"))
  Cmd> addhelpfile("C:/mvmacros/survival.hlp",T) # for DOS/Windows
```

```
  Cmd> addhelpfile("MyDisk:MVMacros:Survival.hlp",T) # on a Mac
  Cmd> addhelpfile("") # file added to be selected in dialog box

                            Cross references
See also topics help(), gethelp().
```

2.5 addlines()

Usage:

```
   addlines([Graph,] x,y [,linetype:PosInt,thickness:PosReal] [,impulse:T]\
      [,other graphics keyword phrases])
   addlines([Graph] [,x,y], keys:str), str a structure whose
     components names match graphics keywords.
```

Keywords: plotting

```
                                  Usage
addlines(x,y) is equivalent to lineplot(x,y,add:T).  It displays the
plot in LASTPLOT, adding lines such as are produced by lineplot(), and
updates LASTPLOT with the new information.

Arguments x, and y are as for lineplot().  They can be replaced by a
structure with at least two REAL components.  Any components beyond the
first two are ignored.  It is not an error when x or y is NULL; no
plotting occurs.

It is an error if LASTPLOT does not exist.

When 'dumbplot' has been set False (see subtopic 'options:"dumbplot"'),
the plot will be a low resolution plot unless 'dumb:F' is an argument.

                         Graph variable as argument
addlines(Graph,x,y) or lineplot(Graph,x,y,add:T) displays GRAPH variable
Graph with the addition of lines connecting points specified by x and
y, saving the modified plot in LASTPLOT.  Graph is not changed (unless
it is LASTPLOT).

                           Keep and show keywords
addlines(x,y,keep:F) (or lineplot(x,y,keep:F,add:T)) suppresses any
change to LASTPLOT.

addlines(x,y,show:F) (or lineplot(x,y,show:F,add:T)) suppresses
immediate display of the modified graph but updates LASTPLOT.  This is
useful when you are building a complex graph in stages using addlines(),
addchars(), addstrings(), or addpoints().  When you are done, simply
type showplot().  You can't use both show:F and keep:F.

                                  Keywords
You can use keywords 'linetype' and 'thickness' to control the type of
lines used (solid, dashed, etc.).  See topic 'graph_keys'.
```

```
Keywords 'dumb', 'lines', 'impulse', 'xmin', 'xmax', 'ymin', 'ymax',
'logx', 'logy', 'xlab', 'ylab', 'title', 'xaxis', 'yaxis', 'borders',
'ticks', 'xticks', 'yticks', 'xticklen', 'yticklen', 'xticklabs',
'yticklabs', 'height', 'width', 'pause', 'silent' and 'notes' may be
used as for other plotting commands.  See topics 'graph_keys',
'graph_border' and 'graph_ticks'

                              Keyword 'keys'
addlines([Graph,] keys:structure(x:x,y:y [other keyword phrases)) is
equivalent to addlines([Graph,] x:x,y:y [other keyword phrases]).  See
topic 'graph_keys' for details.

A value of MISSING for any of xmin, xmax, ymin or ymax (for example,
xmin:?) forces determination of a value from the current data and data
already in the graph.

New labels and title may be set only by keywords 'xlab', 'ylab' and
'title'.

addlines(x,y,add:F, ...) is equivalent to lineplot(x,y, ...).

                             Cross references
See topic 'graph_assign' for information on another way to add data and
other information to a plot.

See topic 'graphs' for general information on plots and on variable
LASTPLOT.  See topic 'graph_keys' for information on keywords.  See
topic 'graph_files' for information on writing a graph to a file.
```

2.6 adddatapath()

Usage:

```
adddatapath(dirName [,T]), dirName a quoted string or CHARACTER vector
  specifying one or more additional directory or folder names to search
  when attempting to read a file
```

Keywords: input, files

```
                                    Usage
adddatapath(DirName) adds DirName at the beginning of CHARACTER vector
DATAPATHS.  DirName must be a quoted string or CHARACTER scalar
specifying the name of a directory or folder.  See topic 'DATAPATHS'.

When commands such as read(), matread(), macroread() and vecread() don't
find a wanted file in the default directory (see 'files'), they search
for it in the folders or directories whose names are in DATAPATHS.
Because DirName is added at the start of DATAPATHS, the folder or
directory will be searched before any other locations in DATAPATHS.

adddatapath(DirName,T) does the same except DirName is added at the end
```

```
of DATAPATHS so the directory or folder will be searched last.

For both usages, DirName can also be a CHARACTER vector, each element of
which specifies a directory or folder to be searched.

When DATAPATHS does not already exist, adddatapath(DirName) creates it.

                                Example
Example:
  Cmd> adddatapath("MyDisk:Survey Folder:") # on Mac OS 9

  Cmd> adddatapath("D:/SURVEY.DIR/") # on DOS/Windows

  Cmd> adddatapath("~/survey.dir") # on Linux/Unix

  Cmd> adddatapath(getfilename(pathonly:T)) # Windowed versions only

The last example adds to DATAPATHS the folder or directory containing
the file selected in a file navigation dialog box.

                            Cross references
See also topics matread(), read(), vecread(), macroread(), inforead(),
getfilename(), 'files'.
```

2.7 addmacrofile()

Usage:

```
addmacrofile(fileName [,T]), fileName a quoted string or CHARACTER
  vector specifying one or more additional files to be searched by
  getmacros().
```

Keywords: macros, files

```
                                Usage
addmacrofile(fileName) adds fileName at the start of CHARACTER vector
MACROFILES which contains the names of files to be searched for macros.
fileName must be a quoted string or CHARACTER scalar.  Because fileName
is added at the startof MACROFILES, the file will be the first one
searched for a macro.

addmacrofile(fileName,T) does the same except the file name is added at
the end of MACROFILES so the file will be searched last.

When fileName is a simple file name containing no directory or folder
information (for example, "mydata.dat" but not "./mydata.dat" or
":mydata.dat"), the file is first assumed to be in the default
directory.  If not found there, MacAnova looks for it in the folders or
directories listed in variable DATAPATHS. See topics 'DATAPATHS',
adddatapath(), 'file_names', 'files', 'macro_files'.

For both usages, fileName can also be a CHARACTER vector, each element
```

```
of which specifies a file to be searched.

When MACROFILES does not already exist, addmacrofile(fileName) creates
it.

                              Windowed versions
addmacrofile("" [,T] does the same, except you pick the file
interactively using a file navigation dialog box.  The complete path
name (file name with directory information) of the selected file will be
added to MACROFILES.  This works only in windowed versions.  An
equivalent usage is addmacrofile(getfilename(type:"text") [,T]).
See getfilename().

If the first argument is a CHARACTER vector, on windowed versions, any
of its elements can be "".

                                  Examples
Example:
  Cmd> addmacrofile("survival.mac")

  Cmd> addmacrofile("C:/mvmacros/survival.mac",T) # for DOS/Windows

  Cmd> addmacrofile("MyDisk:MVMacros:Survival.mac",T) # on a Mac

  Cmd> addmacrofile("~/mvmacros/survival.mac",T) # on Linux/Unix

  Cmd> addmacrofile("") # file added to be selected in dialog box

                              Cross references
See also topics getfilename(), getmacros(), 'macros'.
```

2.8 addpoints()

Usage:

```
addpoints([Graph,] x,y [,lines:T,impulse:T][, graphics keyword phrases])
addpoints([Graph] [,x,y], keys:str), str a structure whose components
  names match graphics keywords.
```

Keywords: plotting

```
                                    Usage
addpoints(x,y) is equivalent to plot(x,y,add:T).  It displays the plot
in LASTPLOT, adding points such as are produced by plot(), and updates
LASTPLOT with the new information.  It is an error if LASTPLOT does not
exist.

Arguments x and y are as in plot().  They can be replaced by a structure
with at least two components.  Any components beyond the first two are
ignored.  It is not an error when x or y is NULL; a warning message is
printed and no plotting occurs.
```

Graph variable as argument

addpoints(Graph,x,y) displays GRAPH variable Graph with the addition of points such as are produced by plot(), and saves the plot in LASTPLOT. Graph is not changed (unless it is LASTPLOT).

Keywords 'keep' and 'show'

addpoints(x,y,keep:F) suppresses any change to LASTPLOT.

addpoints(x,y,show:F) suppresses immediate display of the modified graph but updates LASTPLOT. This is useful when you are building a complex graph in stages using addlines(), addchars(), addstrings(), or addpoints(). When you are done, simply type showplot(). You can't use both show:F and keep:F.

Keywords

Keywords 'dumb', 'lines', 'linetype', 'thickness', 'impulse', 'xmin', 'xmax', 'ymin', 'ymax', 'logx', 'logy', 'xlab', 'ylab', 'title', 'xaxis', 'yaxis', 'borders', 'ticks', 'xticks', 'yticks', 'xticklen', 'yticklen', 'xticklabs', 'yticklabs', 'height', 'width', 'pause', 'silent' and 'notes' may be used as for other plotting commands. See topics 'graph_keys', 'graph_border' and 'graph_ticks'

Keyword 'keys'

addpoints([Graph,] keys:structure(x:x,y:y [other keyword phrases)) is equivalent to addpoints([Graph,] x:x,y:y [other keyword phrases]). See topic 'graph_keys' for details.

Determination of extremes

A value of MISSING for any of xmin, xmax, ymin or ymax (for example, xmin:?) forces determination of a value from the current data and data already in the graph.

Keyword use

New labels and title may be set only by keywords 'xlab', 'ylab' and 'title'.

addpoints(x,y,add:F, ...) is equivalent to plot(x,y, ...).

Cross references

See topic 'graph_assign' for information on another way to add data and other information to a plot.

See topic 'graphs' for general information on plots and on variable LASTPLOT. See topic 'graph_keys' for information on keywords. See topic 'graph_files' for information on writing a graph to a file.

2.9 addstrings()

Usage:

```
addstrings([Graph,] x,y, strings:charVec,[, graphics keyword phrases])
addstrings([Graph,] [x,y, strings:charVec],keys:str), str a structure
  whose component names are graphics keywords
```

Keywords: plotting

```
                                Usage
addstrings(x,y,strings:charVec) displays the plot in LASTPLOT and then
writes the i-th element of CHARACTER vector charVec at position (x[i],
y[i]), updating LASTPLOT to include the new information.  It is
completely equivalent to stringplot(x,y,strings:charVec,add:T).

It is not an error when x or y is NULL; a warning message is printed and
no plotting occurs.

For backward compatibility with earlier versions, keyword 'strings' can
be omitted (addstrings(x,y,charVec)).

When option 'dumbplot' has been set False (see subtopic
'options:"dumbplot"'), the plot will be a low resolution plot unless
'dumb:F' is an argument.

                      Graph variable as argument
addstrings(Graph,x,y,strings:charVec), displays GRAPH variable Graph,
adding the string or strings in charVec, and saves the modified plot in
LASTPLOT.  Graph is not changed (unless it is LASTPLOT).

                        Limitations on x and y
In contrast with other plotting commands, non-NULL x and y must both be
vectors of the same length.  The most usual use is when both x and y are
REAL scalars and charVec is a quoted string or CHARACTER scalar to be
written at coordinates (x,y).  A typical usage would be
  Cmd> addstrings(110,20,strings:"Frequency 1 cycle/week")).

                            Keyword 'justify'
By default, each string is written centered at (x[i], y[i]).  However,
if 'justify:"l"' or 'justify:"r"' is an argument following charVec, each
string will be left or right justified.

                      Keywords 'keep' and 'show'
addstrings(x,y,strings:charVec,keep:F) suppresses any change to
LASTPLOT.

addstrings(x,y,strings:charVec,show:F) suppresses immediate display of
the modified graph but updates LASTPLOT.  This is useful when you are
building a complex graph in stages using addlines(), addchars(),
addpoints(), or addstrings().  When you are done, simply type
showplot().  You can't use both show:F and keep:F.

                                Keywords
Keywords 'dumb', 'xmin', 'xmax', 'ymin', 'ymax', 'logx', 'logy', 'xlab',
```

```
'ylab', 'title', 'xaxis', 'yaxis', 'borders', 'ticks', 'xticks',
'yticks', 'xticklen', 'yticklen', 'xticklabs', 'yticklabs', 'height',
'width', 'pause', 'silent' and 'notes' may be used as for other plotting
commands.  See topics 'graph_keys', 'graph_border' and 'graph_keys'.
Keywords 'impulse' and 'lines' are ignored.

                              Keyword 'keys'
addstrings([Graph,] keys:structure(x:x,y:y,strings:charVec [other
keyword phrases)) is equivalent to addstrings([Graph,] x:x,y:y,
strings:charVec [other keyword phrases]).  See topic 'graph_keys' for
details.

                           Determining extremes
A value of MISSING for any of xmin, xmax, ymin or ymax (for example,
xmin:?) forces determination of a value from the current data and data
already in the graph.

                                Keyword use
New labels and title may be set only by keywords 'xlab', 'ylab' and
'title'.

addstrings(x,y,strings:s,add:F, ...) is equivalent to stringplot(x,y,
strings:s, ...).

                              Cross references
See topic 'graphs' for general information on plots and on variable
LASTPLOT.  See topic 'graph_keys' for information on keywords.  See
topic 'graph_files' for information on writing a graph to a file.
```

2.10 alltrue()

Usage:

```
alltrue(arg1,arg2,...,argm), all arguments LOGICAL scalars
```

Keywords: logical variables, syntax

```
                                  Usage
alltrue(a1,a2,...,aM) is equivalent to a1 && a2 && ... && aM, except
that no arguments are evaluated unnecessarily, that is, it evaluates no
arguments after the first false one.  All arguments must be LOGICAL
scalars.

                                 Example
Example:
   if(!alltrue(isscalar(x,real:T),x > 0, x == floor(x))){
      error("x is not positive integer")
   }

The apparently more natural way to do the same thing
   if(!(isscalar(x,real:T) && x > 0 && x == floor(x))){
```

```
      error("x is not positive integer")
   }
would not do what you want for a non-REAL x since an attempt would be
made to evaluate floor(x), which is illegal for non-REALs.

                            Cross references
See also topics 'logic', anytrue().
```

2.11 anova()

Usage:

```
anova([Model] [,print:F or silent:T,fstats:T,pvals:T,coefs:F,\
   unbalanced:T, marginal:T])
```

Keywords: glm, anova

```
                                  Usage
anova(Model) computes and prints an ANOVA table for the linear model in
the CHARACTER variable Model.

                                Examples
Examples (y a REAL vector, a and b factors, x a variate):
   anova("y = a")                   One-way ANOVA of y
   anova("y = a+b")                 Two-way ANOVA of y with no
                                    interaction
   anova("y = x+a+b+a.b")           Two-way analysis of covariance of
                                    y with interaction and covariate x
   anova("{log10(y)} = {sqrt(x)}+a") One-way analysis of covariance of
                                    log10(y) with covariate sqrt(x)

All variables referred to in Model must be REAL vectors or factors and
have the same lengths.

See topic 'models' for more information on how to specify Model.

                                 Weights
anova(Model,weights:Wts) does an analysis using weighted least squares.
Wts must be a REAL vector with no negative elements, with the same
length as the response vector.  You can abbreviate 'weights:Wts' to
'wts:Wts'.

                              Omitting model
When you omit Model (anova() or anova(,...)), the model used by the most
recent GLM command such as anova(), regress() or poisson() is used.

When the previous GLM command was regress(), no new computations are
done.  The ANOVA table is based on what was previously computed.

When there haven't been previous GLM commands, but CHARACTER variable
STRMODEL exists, anova() uses STRMODEL as Model.
```

Side effect variables created

Side effect variables created are RESIDUALS, HII, DF, SS, DEPVNAME, TERMNAMES, and STRMODEL.

When weights are specified, RESIDUALS = Response - Fitted and WTDRESIDUALS = sqrt(Wts)*RESIDUALS is an additional side effect vector. You should use WTDRESIDUALS rather than RESIDUALS in residual plots or other diagnostic procedures.

Keywords

Other keyword phrases that can be used with anova() are 'unbalanced:T', 'print:T', 'silent:T', 'fstats:T', 'pvals:T', 'coefs:F' and marginal:T'. See topic 'glm_keys' for details. See 'options' for information on changing the default values of 'fstats' and 'pvals'.

Balanced designs

No design with MISSING values, weights or non-factor variables is ever considered to be balanced. This is true, even when all the weights are 1 and the non-MISSING values make up a balanced design.

Otherwise MacAnova recognizes balance in only two cases:

(1) The design is completely balanced, that is, all cells have the same number of cases.

(2) The design is a balanced main effect design such as a Latin square.

In these cases, computations are done by a fast method which uses marginal totals, quite analogous to the usual hand computations for a balanced analysis of variance. Otherwise, the analysis is done by explicitly constructing the design matrix and doing modified Gram-Schmidt orthogonalization.

You can force an unbalanced computation for balanced data by 'unbalanced:T'.

Nonbalanced designs

For non-balanced designs or with 'unbalanced:T', unless 'marginal:T' is an argument, sums of squares are computed sequentially and an advisory message to that effect is printed.

This means that, in an unbalanced ANOVA, to get all the sums of squares useful for testing hypotheses, you may need to run anova() several times, with the terms in the model in different orders. For example, the A main effect sum of squares in a two way unbalanced ANOVA is the sum of squares for 'a' from anova("y=b+a") and the B main effect sum of squares is the sum of squares for 'b' from anova("y=a+b").

In many cases, use of 'marginal:T' can simplify things. For example the A and B sums of squares produced by anova("y=a+b",marginal:T) are the A sums of squares from anova("y=b+a") and the B sums of squares from anova("y=a+b").

After regress()

```
When the previous GLM command was regress() the behavior of anova() with
Model missing is slightly modified -- it uses the results from the
previous computation instead of computing things afresh, even if the
variables in the previous model have been changed or deleted.
Specifically, any factors in the model are treated as variates (with 1
degree of freedom) and, if the previous GLM command was regress() with
weights specified by keyword 'weights' or 'wts', the entries in the
ANOVA table pertain to the weighted regression.

For example, even when a, b, and c are factors, the commands

  Cmd> regress("y=a+b+c",weights:w); anova()

print a summary of the weighted multiple regression, followed by an
weighted regression ANOVA table with 1 degree of freedom for each of a,
b and c.  This is different from anova("y=a+b+c") which computes an
unweighted factorial ANOVA with no interactions.

                            Cross references
See also coefs(), cellstats(), contrast(), factor(), fastanova(),
modelinfo(), predtable(), regress(), secoefs(), xvariables().
```

2.12 anovapred()

Usage:

```
anovapred(a,b,...), a, b, ... all the factors in STRMODEL
```

Keywords: glm, anova

```
                                  Usage
anovapred(a,b,...), where a, b, ... are all the factors in the most
recent GLM model, computes the fitted (predicted) value, the standard
error of estimation, and the standard error of prediction for each
cell. The result is a structure with components 'estimate', 'SEest' and
'SEpred', each of which is a vector, matrix, or array with dimensions
derived from the number of levels of a, b, ....  It uses side effect
variables DEPVNAME, RESIDUALS, and HII.

When the most recent GLM model was manova() with a p-dimensional
dependent variable, each component will have an extra dimension of size
p.

When the most recent GLM model included variates (non-factors), or when
you do not include all factors in the argument list, the results will
probably be wrong, although no warning message will be printed.

anovapred() is implemented as a pre-defined macro.

                            Cross references
See also predtable(), regpred(), 'glm'.
```

2.13 anymissing()

Usage:

```
anymissing(x), x REAL, LOGICAL, or CHARACTER, returns True or False
```

Keywords: missing values, null variables

```
                                Usage
anymissing(x) returns the value True if x contains any missing values
and the value False otherwise.  x must be a vector, matrix, or array.
When x is CHAR, anymissing(x) is True if and only if any string in x is
empty ("").

When x is a NULL variable, anymissing(x) returns the value False.

anymissing(Str), where Str is a structure, returns a structure whose
non-structure components parallel those of Str, but are LOGICAL scalars,
indicating whether the corresponding component of Str contains any
missing values.  To test whether any component of a structure Str
contains any missing values, use if(sum(vector(anymissing(Str))) !=
0){...}.

                               Examples
Examples:
  Cmd> vector(anymissing(vector(1,5,?)),anymissing(vector("A","B","")))
  (1) T       T

  Cmd> vector(anymissing(vector(1,5,7)),anymissing(vector("A","B","C")))
  (1) F       F

  Cmd> anymissing(structure(a1:vector(1,5,?),a2:vector("A","B","C")))
  component: a1
  (1) T
  component: a2
  (1) F

                           Cross references
See also topics ismissing(), 'NULL'.
```

2.14 anytrue()

Usage:

```
anytrue(arg1,arg2,...,argM), all arguments LOGICAL scalars
```

Keywords: logical variables, syntax

```
                                Usage
anytrue(a1,a2,...,aM) is equivalent to a1 || a2 || ... || aM, except
that no arguments are evaluated unnecessarily, that is, it evaluates no
arguments after the first true one.  All arguments must be LOGICAL
```

```
scalars.

                                Example
Example:
   if(anytrue(!isscalar(x,real:T),x <= 0, x != floor(x))){
      error("x is not positive integer")
   }

The apparently more natural way to do the same thing
   if(!isscalar(x,real:T) || x <= 0 || x != floor(x)){
      error("x is not positive integer")
   }
would not do what you want for a non-REAL x since an attempt would be
made to evaluate floor(x), which is illegal for non-REALs.

                            Cross references
See also topics 'logic', alltrue()
```

2.15 appendnotes()

Usage:

```
   appendnotes(x,Notes), Notes a CHARACTER scalar or vector
```

Keywords: general, macros, variables

```
                                Usage
appendnotes(x, Notes) appends Notes to the notes "attached" to variable
x.  When x has notes, appendnotes(x, Notes) is equivalent to
attachnotes(x, vector(getnotes(x),Notes)).  When x does not have notes,
appendnotes(x, Notes) is equivalent to attachnotes(x,Notes).

x must be an existing variable of any type, including a structure, a
macro or a GRAPH variable.

Notes must be a CHARACTER scalar or vector, usually containing
descriptive or usage information about variable x.  When Notes is NULL,
any notes attached to x are not changed.

You can retrieve notes using getnotes().  You can test whether a
variable has notes attached using hasnotes().

                            Cross references
See also topics 'notes', attachnotes(), getnotes(), hasnotes().
```

2.16 arginfo_fun

Keywords: general, control

```
This topic is now in file userfun.hlp.  Type
  userfunhelp(arginfo_fun)

It provides a brief introduction to the form of an arginfo function,
that is, an externally compiled function to be called by MacAnova to
obtain information about the arguments expected by a user function.

Some other useful entries in userfun.hlp are arginfo_fun and
callback_fun.  Type
  userfunhelp()
for a complete list of entries.
```

2.17 argvalue()

Usage:

```
   argvalue(var, argName, [, Properties]), var any variable, argName
     CHARACTER scalar, Properties CHARACTER scalar or vector whose elements
     are one or more of "array", "character", "count", "graph", "integer",
     "logic", "macro", "matrix", "nonmissing", "nonnegative", "notnull",
     "number", "positive", "real", "scalar", "square", "string",
     "structure", "TF" and "vector"
```

Keywords: syntax, macros

```
                                    Usage
argvalue(Var, argName, Properties), where argName is a quoted string or
CHARACTER scalar and Properties is a CHARACTER scalar or vector, checks
Var to see if satisfies the restrictions specified by Properties.  When
Var does satisfy the restrictions, the value of Var is returned and
argName is ignored.  Otherwise, it is an error and the resulting error
message incorporates argName.

Var can be any defined variable but cannot be a built-in function.

Properties is usually a quoted string or CHARACTER scalar such as
"nonmissing real", made up of one or more words separated by spaces or
tabs.  Alternatively, Properties can be a CHARACTER vector like
vector("nonmissing", "real"), each element of which contains one or more
words.

                                  Properties
Properties recognized are "array", "character", "count", "graph",
"integer", "logic", "macro", "matrix", "nonmissing", "nonnegative",
"notnull", "number", "positive", "real", "scalar", "square", "string",
"structure", "TF" and "vector".

Not all combinations or words are permitted.  See keyvalue() for
details.

The following properties are abbreviations for combinations of other
```

```
properties specifying types of scalars:
  "number" means  "nonmissing real scalar"
  "count"  means  "nonnegative integer scalar"
  "TF"     means  "nonmissing logical scalar"
  "string" means  "character scalar"

Any 3 character or longer initial segment of a property will match it,
except that "nonnegative", "nonmissing", "string" and "structure"
require 4.  For example, "vec", "vect", "vecto", ... all match "vector".

y <- argvalue(var, argName), with no Properties argument, is
essentially equivalent to y <- var except that argName is used in an
error message if var is not defined.

                                Example
argvalue() is designed to be used in writing macros.  It allows easy
checking of non-keyword macro arguments with automatic printing of
informative error messages.  As a typical example of its use, here is
the text of macro gamma() that uses lgamma() to compute the gamma
function of a REAL array of positive elements:

  if ($v != 1 || $k > 0){error("usage is gamma(x)")}
  @x <- argvalue($1,"$1","positive array")
  exp(lgamma(@x))

Instead of "$1" as argument 2 to argvalue(), you might use "argument 1".
Either choice is likely to yield an informative error message.

When gamma() executed, $1 is replaced by argument 1 to gamma(), $v and
$k are replaced by the number of keyword and non-keyword arguments,
respectively, and $S is replaced by the name of the macro, here 'gamma'.
See topics 'macros', 'macro_syntax' and macro() for details.

  Cmd> gamma(run(4))
  (1)           1           1           2           6

  Cmd> gamma(run(0,2))
  ERROR: run(0,2) is not an array of positive REALs

The second line of the macro was expanded to
  @x <- argvalue(run(0,2),"run(0,2)","positive array")

                           Cross references
See also keyvalue(), nameof(), getkeywords(), isscalar(), isvector(),
ismatrix(), isarray(), isreal(), ischar(), islogic(), ismacro(),
isstruc(), isnumber(), isgraph(), isdefined(), 'macros'.
```

2.18 arimahelp()

Usage:

```
arimahelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
arimahelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
  Subtopics
arimahelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
arimahelp(key:Key), CHARACTER scalar Key
arimahelp(index:T [,scrollback:T])
```

Keywords: general, time series

```
                                Usage
arimahelp(Topic1 [, Topic2, ...]) prints help on topics Topic1, Topic2,
... related to macros in file arima.mac.  The help is taken from file
arima.mac.

arimahelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these macros.

arimahelp(index:T) or simply arimahelp() prints an index of the topics
available using arimahelp().  Alternatively, help(index:"arima")
does the same thing.

arimahelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER
scalar or vector, prints subtopics of topic Topic.  With subtopic:"?", a
list of subtopics is printed.

arimahelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where Suptopics1
and Subtopics2 are CHARACTER scalars or vectors, prints the specified
subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to arimahelp().  In windowed
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

arimahelp(key:key) where key is a quoted string or CHARACTER scalar
lists all topics cross referenced under Key.  arimahelp(key:"?") prints
a list of available cross reference keys for topics in the file.

arimahelp() is implemented as a predefined macro.

                          Cross references
See help() for information on direct use of help() to retrieve
information from arima.mac.
```

2.19 arithmetic

Usage:

```
a + b, a - b, a * b, a / b, a %% b, a^b, -a, +a
a <-+ b, a <-- b, a <-* b, a <-/ b, a <-%% b, a <-^ b
```

Keywords: syntax, operations, missing values

```
                                Operators
There are 8 operators for doing arithmetic with MacAnova variables.

     Operators        Precedence  Meaning
       a + b               9      Addition (sum of a and b)
       a - b               9      Subtraction (difference of a and b)
       a * b              10      Multiplication (product of a and b)
       a / b              10      Division (a divided by b)
       a %% b             10      Modular division (see below)
        -a                12      Unary minus ((-1)*a)
        +a                12      Unary plus ((+1)*a))
  a ^ b or a ** b         13      Exponentiation (a to the b-th power)

(Level 11 is the precedence level of matrix multiplication; see
topic 'matrices'.)

                            Same sized operands
The arithmatic operators are all element-wise operations: When a and b
are vectors, matrices, or arrays with identical dimensions, then c <- a
OP b, computes a result of the same size and shape such that c[i,j,...]
is a[i,j,...] OP b[i,j,...], where OP is one of these operators.
Similarly (-a)[i,j,...]  is -(a[i,j,...]).

  Cmd> vector(1,3,2,0) + vector(8,-1,2,9)
  (1)           9           2           4           9

  Cmd> -vector(1,3,2,0)
  (1)          -1          -3          -2           0

See below for how they work when a and b do not have identical
dimensions.

                            Modular division
Modular division x %% y computes the non-integral part of x / y or zero
if y exactly divides x.  It is implemented as x %% y = x - y*floor(x/y).
In particular, 17 %% 4 is 1, -17 %% 4 is 3, and -17 %% -4 is -1.  a %% 0
is always MISSING.

                            Dividing by zero
When a is not zero, a / 0 yields has value MISSING.  However, 0 / 0 has
value 0.  This can be a useful convention when a and b have the same
pattern of zero elements.

                            Non integer power
When p is not an integer, a^p (or a**p) is defined to be
sign(a)*abs(a)^p and a warning message is printed when a < 0.  0^p is
zero when p > 0 or MISSING when p < 0.  a^0 is always 1, even when a =
0.

                            Logical variables
```

LOGICAL variables and constants may be used in arithmetic expressions and comparisons. Values True and False are translated as 1 and 0, respectively. In particular 1*w converts a LOGICAL vector, matrix, or array w to a numerical vector, matrix or array of 0's and 1's..

Missing values

When any elements of a and/or b are MISSING, so is the corresponding element of a OP b and a warning message is printed.

Too large result

When any element of the result is too large a number to be represented in the computer (for example, 1e300/1e-300), the result is set to MISSING and a warning message printed.

Cross references

See subtopic 'options:"warnings"' for information on option 'warnings' which you can set to suppress warning messages.

Effect of precedence level

The precedence level in the list of operators affects the order of evaluation when there is more than one operator in an expression. An operator with higher precedence is evaluated before one with lower precedence. For example, 3 + 2 * 4 is interpreted as 3 + (2*4) = 11 because '*' has higher precedence than '+'. See topic 'precedence' for complete information on the order of evaluation.

Behavior of arithmetic, logical, and bit operations when operands differ in size.

Scalar operand

Scalar operand:

A scalar operand (single number, all dimensions 1) is combined or compared with all the elements of the other operand. For example, x - 2 subtracts 2 from each element of x and x == 2 compares every element with 2. The result has the same dimensions and labels as the other operand.

When both operands are scalars, the result is unlabeled unless both operands have the same number of dimensions and the same labels. If the number of dimensions is different, the result has the larger number of dimensions.

Column vector op matrix

Column vector operand and matrix operand:

When a column vector of length m (m by 1 matrix or vector of length m) is combined or compared with a m by n matrix, it is combined or compared with each column of the matrix to yield a m by n matrix. For example, when a is 3 by 6, run(3) + a adds 1 to row 1, 2 to row 2, and 3 to row 3, and a != vector(1,1,2) compares elements in rows 1 and 2 with 1 and elements in row 3 with 2.

Row vector op matrix

Row vector operand and matrix operand:

When a 1 by n matrix (a row vector) is combined or compared with a m by n matrix, the row vector is combined or compared with each row of the matrix. For example, if x is a matrix

```
Cmd> xbar <- sum(x)/nrows(x); resids <- x - xbar
```

subtracts the average of the rows of x from every row, since sum(x) computes a row vector with the same number of columns as x. This would not work if xbar were computed as xbar <- describe(x,mean:T) because describe() computes it as a vector, not a row vector. However, in this case, resids <- x - xbar' would work.

Row vector op column vector

Row vector operand and column vector operand:

When a column vector of length m is combined or compared with a row vector of length n, the result is the m by n matrix obtained by combining each element of the column vector with each element of the row vector, what might be called an outer product, outer sum, etc.

Examples

```
Cmd> run(2)/run(3)'
(1,1)          1        0.5     0.33333   [1/1  1/2   1/3]
(2,1)          2          1     0.66667   [2/1  2/2   2/3]

Cmd> run(2) <= run(3)'
(1,1) T       T       T                    [1<=1  1<=2   1<=3]
(2,1) F       T       T                    [2<=1  2<=2   2<=3]
```

Structure operands

Structure operand(s)

When one of the operands is a structure, each of its components is combined with the other argument, following the same rules of compatibility just described, producing a structure with the same shape as the structure argument. When both arguments are structures, they must have the same number of components at every level and the corresponding components are combined. NULL components are permitted as long as they appear in both operands in the same places.

Example:

```
Cmd> structure(x:run(2),y:run(4),z:NULL)/structure(run(3)',4,NULL)
component: x
(1,1)          1        0.5     0.33333
(2,1)          2          1     0.66667
component: y
(1)        0.25        0.5        0.75          1
component: z
(NULL)
```

Arithmetic assignment operators

Operators '<-+', '<--', '<-*', '<-/', '<-^', '<-**', and '<-%%' are useful for modifying a variable. They are best illustrated by example.

```
a <-+ 3      is equivalent to a <- a + 3
a <-%% b     is equivalent to a <- a %% b
a <-^ -1     is equivalent to a <- a^(-1)
```

To avoid ambiguity, '<-+' and '<--' must be followed by at least one space so that, for example 'a <-- 3' means 'a <- a - 3' instead of 'a <- -3'.

You can't modify parts of a vector, matrix or array using assignment operators. For example, a[1,2] <-/ 3 is illegal. Use a[1,2] <- a[1,2]/3.

See topic 'precedence' for information on what happens when more than one of these operators is used in an expression.

Cross references

See also topics 'logic', 'structures', 'syntax', 'bit_ops'.

2.20 array()

Usage:

array(x,n1,n2,...[,KeyPhrases]) or array(x,dimVec [,KeyPhrases]), x REAL, LOGICAL or CHARACTER, n1, n2, ... positive integers or dimVec a vector of positive integers

KeyPhrases can be labels:structure(lab1,lab2,...), notes:Notes and silent:T, where lab1, lab2, ... and Notes are CHARACTER scalars or vectors.

Keywords: variables, combining variables, character variables

Usage

array(x,dimVec) makes an array whose dimensions are the elements of dimVec, a vector of positive integers. The data are taken from REAL, LOGICAL or CHARACTER variable x. The dimensions of x are ignored, that is, array(x,dimVec) is equivalent to array(vector(x),dimVec). For example, array(run(24),vector(4,3,2)) creates a 4 by 2 by 2 array.

Except when x is a scalar, there must be exactly N elements in x, where N = product of dimensions. array() duplicates a scalar N times so, for example, array(0,2,3,4) is the same as array(rep(0,24),2,3,4).

The data from x are entered with the leftmost dimensions varying fastest and the rightmost varying slowest. For example, array(run(20), vector(5,4)) is equivalent to matrix(run(20),5).

array(x,n1,n2,...) is equivalent to array(x,vector(n1,n2,...)), when n1, n2, ... are REAL scalars or vectors. Most usually n1, n2, ... are scalars, as in array(run(24),4,3,2) which creates a 4 by 3 by 2 array.

When x is a scalar, vector, matrix or array, array(x), with no dimensions, is equivalent to array(x,dim(x)), that is, it returns a variable identical to x, possibly with the addition of coordinate labels or notes. See below.

```
                    Keywords 'labels' and 'notes'
You can specify coordinate labels for the output using keyword phrase
'labels:Labels'.  See topic 'labels' for details.

You can attach a CHARACTER vector Notes of descriptive notes to the
result using keyword phrase 'notes:Notes'.  See topic 'notes' for
details.

When no dimensions are specified or the new dimensions exactly match the
dimensions of x, any coordinate labels or descriptive notes of x are
transferred to the result unless 'labels' or 'notes' provide new labels
or notes or are NULL.

                            Cross references
See also topics matrix(), 'matrices', 'subscripts', 'variables'.
```

2.21 arrays

Usage:

Create an array by	a <- array(data,dim1,...,dimk)
Extract elements by	b <- a[j1,...,jk]
Determine the number of dimensions	ndims(a)
Determine the dimensions	dim(a)
Determine the number of elements	length(a)
Reorder subscripts by	b <- a' or b <- t(a,J), permutation vector J

Keywords: variables

```
                                Description
Any REAL, LOGICAL or CHARACTER variable is an array with some number of
dimensions, say M.  To extract or change any single element of an array
you need M subscripts, j_1, j_2, ..., j_M.  See topic 'subscripts'.

The "dimensions" of an array are the maximum permitted values for each
of the M subscripts.  If the dimensions are N_1, N_2, ..., N_M, we
sometimes say the array is N_1 by N_2 by ... by N_M.

When extracting or changing elements of an array using subscripts, it is
an error if any subscript j_k > N_k.

You can create an array A by
  Cmd> A <- array(a,N_1,N_2,...,N_M).
where a is a vector or array with exactly N_1*N_2*...*N_M elements.

A one dimensional array is called a vector and a two dimensional array
is called a matrix.  See topic 'vectors' and 'matrices'.

                            Order of elements
Elements of an array are stored with the first subscript changing most
rapidly, the second changing second most rapidly, and so on, with the
```

```
last subscript changing least rapidly.  Thus for a 2 by 2 by 2 array,
the elements are in the order
    a[1,1,1], a[2,1,1], a[1,2,1], a[2,2,1], a[1,1,2], a[2,1,2],
    a[1,2,2],a[2,2,2]

                          Functions of arrays
There are several functions that are helpful in working with arrays.  In
the following, A is an array.
  length(A)      The number of elements in A
  ndims(A)       The number of dimensions M of A
  dim(A)         The sizes vector(N1,...,NM) of all the dimensions of A
  a' or t(a)     The same elements of a, with dimensions reversed, so
                 that a'[j1,j2,...,jk] is a[jk,...,j2,j1]
  t(a,J)         The same elements of a with dimensions permuted by
                 elements of vector J so that t(a,J)[K[1],K[2],...,K[k]]
                 is a[K[J[1]],K[J[2]],...,K[J[k]]]
  vector(a)      The elements of a in a vector retaining the order.

max(a), min(a), sum(a) and prod(a) operate along the first dimension of
a, returning an array with the same number of dimensions with the first
dimension 1.

Transformations of an array a such as cos(a) and sqrt(a) return arrays
with the same dimensions as a.

                          Cross references
See also 'variables', 'scalars', 'vectors', 'matrices', length(),
ndims(), dim(), t(), vector(), matrix(), array().
```

2.22 asciisave()

Usage:

```
asciisave(FileName [,all:T, v335:T, v406:T, nulls:F, options:F,\
   history:T])
asciisave() repeats previous save() or asciisave() with same options
```

Keywords: files, general, output

```
                                  Usage
asciisave(FileName) saves the MacAnova "workspace", that is, all the
current variables and option values, in a file with name given in the
quoted string or CHARACTER variable FileName.  On versions with windows,
FileName can be "", in which case you will be prompted for the file
name.  The file written is an ASCII coded text file which should be
readable by restore() on any computer on which MacAnova runs.

asciisave(FileName,ascii:F) is equivalent to save(FileName), that is,
the file written will be a binary file instead of an ASCII text file.
This option can be used together with others described below.

asciisave(FileName, var1, var2, .... [,ascii:F]) saves only variables or
```

```
macros var1, var2, ... on the file.  When any of the variables saved is
specified in keyword form, the keyword is used for the name.  The items
saved can be restored without deleting everything by
restore(FileName,delete:F).

                              File name omitted
When FileName is omitted and a previous asciisave() or save() was
executed, the same file will be used as before.  Moreover, when the
previous save() specified an obsolete file format (see save() for
details), the same option will be used, unless explicitly changed.  When
there was no previous save() or asciisave(), omitting the file name is
an error.

                                  Keywords
See save() for information on keywords 'all', 'null', 'options',
'graphwind' and 'history' which control whether information on GLM
computations, option values, graph windows and previous commands
should be saved with the workspace.

                           Difference from save()
asciisave() differs from save() in that asciisave() saves the
information in the form of a "text" file that can be transferred between
different types of computers.  Files created by asciisave() are often
bigger than the corresponding file created by save().  On a Macintosh,
the actual type is 'Sasc' rather than 'TEXT'.

The file produced by asciisave() consists of many short lines.  All the
characters written are printable ASCII characters (CR and space through
~), with any other characters in escaped octal format ('\t' for TAB).
The file can be printed, viewed in an editor, or sent by E-mail.  It
cannot be edited safely without specialized knowledge of the actual
format used.

                              Cross references
See also topics restore(), 'files'
```

2.23 asin()

Usage:

```
asin(x [, degrees:T or radians:T or cycles:T]), x REAL or a structure
  with REAL components value in radians (default), cycles, or degrees as
  specified by option "angles" or the optional keyword
```

Keywords: transformations

```
                                    Usage
asin(x) computes the inverse sine of the elements of x, where x is a
REAL scalar, vector, matrix or array.  The result has the same shape as
x.  sin(asin(x)) is the same as x except for rounding error.

The units of the result are radians, degrees or cycles as determined by
```

```
the value of option 'angles'.  The default is radians.  See subtopic
'options:"angles"'.

asin(x, radians:T), asin(x, degrees:T), asin(x, cycles:T) return results
in the indicated units, regardless of the value of option 'angles'.

When any element of x is MISSING or is above 1 or below -1, the
corresponding element of the result is MISSING and a warning message is
printed.

                            Structure argument
When x is a structure, all of whose non-structure components are REAL,
asin(x [,UNITS:T]), where UNITS is one of 'radians', 'degrees' or
'cycles', is a structure of the same shape and with the same component
names as x with each non-structure component transformed by asin().

                                 Examples
  Cmd> vector(asin(.5),asin(.5,degrees:T),asin(.5,cycles:T))
  (1)      0.5236          30    0.083333

                             Cross references
See topic 'transformations' for information on asin().
```

2.24 asLong()

Usage:

```
   asLong(x), x REAL with no MISSING values and with integer values between
     -2147483647 and 2147483647 = 2^31-1.
```

Keywords: transformations, variables

```
                                    Usage
asLong(x), where x is REAL returns a LONG variable the same size and
shape as x, but with all of its elements represented as integers instead
of floating point values.  All the elements of REAL scalar, vector,
matrix or array x must be exact integers with values between -2147483647
and 2147483647 = 2^31-1.

The only use at present for asLong() is to create a long integer
argument to a user function called by User().  When the argument is
returned it is "coerced" to an equivalent REAL variable.  For example,
User("foo", result:asLong(20)) will return a REAL integer scalar value.

asLong(x) is also legal as an argument to print() and write().  For
example, print(asLong(vector(1,3,5,2))) produces the same output as
print(vector(1,3,5,2)).

When assigned (y <- asLong(x)), a LONG variable is "coerced" to a
ordinary REAL variable.  For example, a <- asLong(vector(1,3,5,2))
has the same effect as a <- vector(1,3,5,2).
```

```
                          Cross references
See also topics 'variables' and, in file userfun.hlp, User() (type
userfunhelp(User).
```

2.25 assignment

Usage:

```
  a <- x assigns value of x to a.
  a[J1] <- x, a[J1,J2] <- x, ..., where the J's are valid subscripts,
    replaces the designated elements to corresponding elements of x.
  a[J] <- x, when a is a structure, replaces the designated components
    of a by x, J a valid subscript
  a <-+ x assigns a + x to a and similarly for a <-- x, a <-* x, a <-/ x,
    a <-%% x and a <-^ x.
  When Str is a structure"
    Str$a <- x, Str$a$b <- x, ..., Str[[i]] <- x, Str[[i]][[j]] <- x, ...
    replaces the indicated component of Str by x
```

Keywords: syntax

```
                            Introduction
This topic describes the use of the assignment operator '<-' and
arithmetic assignment operators '<-+', '<--', '<-*', '<-/' and '<-%%'.
It has sections on ordinary assignment, assignment to subscripts,
assignment to structure components and arithmetic assignment operators.

                          Ordinary assignment
You can assign values to a variable using the left pointing arrow '<-'
made up of the two characters "less than" and "minus".  For example,
'foo <- 5' assigns the value 5 to the variable foo.  When foo does not
already exist, it is created; otherwise, its previous value is discarded
and foo is re-defined.

An expression of the form 'y <-3', say, is always interpreted as 'y <-
3' rather than as 'y < -3'.  When you want the latter, be sure to put a
space before '-3'.

The value of such an assignment is the value of the variable after the
assignment.  For example, 'y <- exp(x <- 4)' sets variables x and y to 4
and exp(4), respectively, and 'y <- x <- 4' assigns 4 to both x and y.

This value is normally not printed unless the assignment is the last
command in a compound command {command_1;...;command_k}.

For example, '{y <- 3}' not only assigns the value 3 to y put also
prints the number 3, although 'y <- 3' by itself prints nothing.  For
this reason, it is a often a good idea to terminate compound commands
with ';;', as in '{y <- 3;;}'.  Of course, this is a bad idea if you
want the final value to be printed or if you are assigning the value of
the entire compound statement to a variable.
```

Some "special" variables such as CLIPBOARD can be assigned to. What actually happens depends on the particular variable. You cannot assign to special structure variable GRAPHWINDOWS although you can assign to its components. See topics 'CLIPBOARD', 'GRAPHWINDOWS' and 'graph_assign'.

Assignment to subscripts

You can modify parts of an existing vector, matrix or array y by y[J1] <- x, y[J1,J2] <- x, y[J1,J2,J3] <- x, ... as long as the subscripts are appropriate. You can use positive, negative and LOGICAL vector subscripts or a single matrix subscript, but not an array subscript with more than two dimensions.

x must be the same type variable as y, REAL, CHARACTER or LOGICAL. When x is a scalar (number, T or F or quoted string), it replaces all the elements of y selected by the subscripts. When x is not a scalar, then length(x) must match the number of elements of y selected, but x can be of any shape and is treated as if it were vector(x).

The value of an assignment to subscripts is a vector, matrix or array containing the new elements and having the same shape as the elements of y that were replaced.

It is legal for the subscripts to select the same element of x more than once (for instance, y[vector(1,1,2)] <- vector(3,5,7)). In this case the eventual value for an element selected more than once is the last element in x assigned to that element (in the example, y[1] is set to 5).

See below for using subscripts to change components of an existing structure.

NULL or non-selecting subscripts

y[J1] <- NULL, y[J1,J2] <- NULL, ... are legal provided at least one of the subscripts is NULL or is non-selecting (is all False or is a complete set of negative subscripts). y is not changed and the value of the assignment is NULL. For example, even if all the elements of u are positive, y[u < 0] <- x[u < 0] is legal and does not change y.

Similarly, when x is a scalar of the appropriate type , y[J1] <- x, y[J1,J2] <- x, ... is legal even one or more subscripts are NULL or non-selecting. y is not changed and the value of the assignment is NULL. For example, y[vector(y) > 10] <- 10 is legal even when there are no elements of y greater than 10.

It is an error to assign a non-NULL non-scalar variable to subscripts when there is a NULL or non-selecting subscript.

Vector subscript for matrix or array

Suppose y is a matrix or array and J is a vector such that vector(y)[J] is legal, and x is a scalar or a vector with the same length as vector(y)[J]. Then y[J] <- x is legal and assigns the elements of x to the positions that would be specified by J if y were a vector. The

```
dimensions of y are retained.  For example,
  Cmd> y[vector(abs(y)) > 3] <- ?
replaces all elements of y that exceed 3 in absolute value by MISSING,
without disturbing the dimensioning of y.

Similarly, when x is a scalar or is a vector, matrix or array with
length(x) = length(y), y[] <- x replaces all the values of y by values
from x without changing the dimensions of y.

When y is not a structure, y[[J]] <- x is illegal except when J = 1 in
which case it is equivalent to y <- x.

See below (assignment:"assignment_to_structure_components") for
assignment to components of a structure.

                                Examples
Examples assuming x is a vector of length 5 and y is a 3 by 2 matrix.

  Cmd> y <- x[-run(2)] <- 17
sets all the elements of except x[1] and x[2] to 17 and sets y to
vector(17,17,17).

  Cmd> y <- x[vector(1,4)] <- vector(17,19)
sets x[1] and x[4] to 17 and 19 and y to vector(17,19)

  Cmd> y <- x[vector(1,3,1)] <- vector(17,19,21)
sets x[1] and x[3] to 21 and 19 and y to vector(17,19,21).

  Cmd> y[vector(1,4)] <- run(3)
is illegal because there are 3 elements in run(3), but only 2 elements
are selected in y.

  Cmd> y <- x[-1,] <- run(4) # change all but row 1 of x
changes x[2,1], x[3,1], x[2,2] and x[3,2] to 1, 2, 3 and 4,
respectively, and sets y to the 2 by 2 matrix matrix(run(4),2).

  Cmd> y <- x[hconcat(run(2),run(2))] <- 4 #matrix subscript
sets x[1,1] and x[2,2] to 4 and y to vector(4,4).

  Cmd> y <- x[x>max(x)] <- 3
doesn't change x and sets y to NULL because x > max(x) has value
vector(F,F,F,F,F).

                  Assignment to structure components
You can assign to structure components by name or by number.  In the
following Str is assumed to be an existing structure variable, not the
result of an operation like describe(x).

When a component of Str has name Name, Str$Name <- x replaces that
component by the value of x without changing it's name.  The value of
the assignment expression is x.  If more than one component is named
Name, assignment is to the first such component.  It is an error if Str
has no such component.
```

Str$Name1$Name2 <- x, Str$Name1$Name2$Name3 <- x, ... are also legal, provided the indicated component exists.

Str[J] <- x and Str[[J]] <- x are identical and work similarly to a[J] <- x, when a is a vector, matrix or array, except that entire components are replaced. The names of components in Str are never changed.

There are four cases depending on x and K = number of components selected in Str by J, counting any duplicated subscripts more than once.

1. When K = 1, the selected component is always replaced by a copy of x, whether x is a structure or a non-structure.

2. When K > 1 and x is a structure with ncomps(x) = K, the selected components in Str are replaced by copies of the corresponding components of x and the value is identical to x. When a component of Str is selected more than once, its new value is the highest numbered component of the x that was assigned to it. The value of the assignment is a copy of x.

3. When K > 1 and x is not a structure or is a structure with ncomps(x) != K, each selected component of Str is replaced by a copy of x. The value is structure(x,x,...,x), where there are K copies of x.

4. When K = 0, that is, no component of Str is selected as in Str[rep(F,3)], x is ignored, Str is not changed and the value is NULL.

In addition, you can specify by number components of components to be changed. For example, Str[[3]][[2]] <- x replaces component 2 of the third compoent of Str by x and Str[[3]][[-1]] <- x replaces all but the first component of the third component of Str by x. If x is a structure with the right number of components, each component of Str[[3]][[-1]] is replaced by the correspoding component of x. Otherwise, each component of Str[[3]][[-1]] is replaced by x.

You can nest component specificiation, mixing names and [[...]] subscripts up to 31 deep. All subscripts except possibly the final one must be integer scalars. With nested components, no [...] subscripts are allowed, that is, Str[[1]][3] <- x is illegal; use Str[[1]][[3]] <- x.

Arithmetic assignment operators

There are several arithmetic assignment operators: <-+, <--, <-*, <-/, <-%% and <-^. For example, a <-* b is equivalent to a <- a*b and a <-^ b is equivalent to a <- a^b. '<--' and '<-+' require a following space.

The variable being modified cannot be subscripted or be a structure component. For example, x[3] <-+ 1 and Str$x <-- 1 are illegal. See topic 'arithmetic' for more information.

2.26 atan()

Usage:

```
atan(x [, degrees:T or radians:T or cycles:T]), x REAL or a
  structure with REAL components; value in radians (default), cycles, or
  degrees as specified by option 'angles' or the optional keyword
atan(x,y [, degrees:T or radians:T or cycles:T]), y REAL or a structure
  with real components the same size and shape as x
```

Keywords: transformations

```
                                Usage
atan(x) transforms the elements of REAL vector, matrix, or array x to
inverse tangents (arctangents).  When x is a structure with components
x1,...,xm, atan(x) is a structure with components atan(x1),...,atan(xm).

When any element of x is MISSING, the corresponding element of atan(x)
is MISSING.

atan(x,y) computes theta = arctan(x/y), with the result in the
appropriate quadrant, where x and y must be REAL vectors, matrices, or
arrays with the same dimensions.  Specifically, theta is chosen so that
sin(theta) has the same sign as x, cos(theta) has the same sign as y and
tan(theta) = x/y.

atan(x,y) is also defined when x and y are both structures with the same
number of components, say x is structure(x1,...,xm) and y is
structure(y1,..., ym) .  The result is what would be produced by
structure(atan(x1,y1),...,atan(xm,ym)).

                                Units
The units of the result of atan(x) and atan(x,y) are radians, degrees or
cycles as determined by the value of option 'angles'.  The default is
radians.  See subtopic 'options:"angles"'.

                              Keywords
atan(x, radians:T), atan(x, degrees:T), atan(x, cycles:T), atan(x, y,
radians:T), atan(x, y, degrees:T) and atan(x, y, cycles:T) return values
in the specified units, overriding option 'angles'.

                           Cross references
See topic 'transformations' for more information about atan(), including
its use with a CHARACTER argument.

See also topics 'structures', 'labels'.
```

2.27 atanh()

Usage:

```
atanh(x), x REAL or a structure with REAL components
```

Keywords: transformations

Usage

atanh(x) returns the inverse hyperbolic tangent of the elements of x, when x is a REAL scalar, vector, matrix or array. The result has the same shape as x. In terms of other functions, atanh(x) = .5*log((1+x)/(1-x)).

This transform is sometimes called the Fisher z-transform. When r is a sample Pearson correlation from a bivariate normal sample of size N and population correlation rho, atanh(r) is approximately normal with mean rho and variance 1/(N-2).

When any element of x is MISSING, so is the corresponding element of atanh(x). When any element of x >= 1 or <= -1, the corresponding element of atanh(x) is MISSING. In both cases a warning message is printed.

Structure argument

When x is a structure, all of whose non-structure components are REAL, atanh(x) is a structure of the same shape and with the same component names as x, with each non-structure component transformed by atanh().

Cross references

See topic 'transformations' for more information on atanh().

2.28 attachnotes()

Usage:

attachnotes(x,Notes), Notes a CHARACTER scalar or vector or NULL

Keywords: general, macros, variables

Usage

attachnotes(x, Notes) "attaches" Notes to variable x as descriptive "notes".

When Notes is NULL, any existing notes are removed from x. Otherwise, Notes must be a CHARACTER scalar or vector, usually containing descriptive information about variable or macro x.

x must be an existing variable of any type, including a structure, a macro or a GRAPH variable. You can't attach notes to certain special variables like CLIPBOARD and GRAPHWINDOWS.

You can retrieve notes using getnotes() and append notes to previously attached notes using appendnotes(). You can test whether a variable has notes attached using hasnotes().

Cross references

See also topics 'notes', appendnotes(), getnotes(), hasnotes().

2.29 autoreg()

Usage:

```
autoreg(Phi,A [,reverse:T, limits:vector(i1 [,i2]), start:startVals,\
  seasonal:L]), REAL vector or NULL Phi, REAL vector or matrix A, REAL
  startVals the same size and shape as A, positive integer L
```

Keywords: time series

```
                                Introduction
autoreg() is designed to implement an autoregressive operator as the
term is used in ARIMA time series analysis.  It can also be used to
compute partial sums of a series or, together with movavg(), to find the
power series coefficients of rational functions.

                                    Usage
autoreg(Phi,A) applies the autoregressive operators specified by the
columns of the REAL matrix Phi to the columns of the REAL matrix A.
When ncols(Phi) = 1, Phi is applied to every column of A and if ncols(A)
= 1, each column of Phi is applied to A.  The result is a matrix with
nrows(A) rows and max(ncols(Phi), ncols(A)) columns.  When both Phi and
A have more than one column, they must both have the same number of
columns.

Specifically, assuming for simplicity that both Phi and A are vectors so
that the result x is a vector,
    x[i] = A[i] + sum(Phi[k]*x[i-k],1<=k<=nrows(Phi)),
with x[l] taken to be 0 for l < 1.

When Phi is a vector, movavg(Phi,A) can be expressed in matrix terms as
solve(Phi1, A), where Phi1 is a nrows(A) by nrows(A) matrix.  For
example, when nrows(Phi) = 2,
         [   1       0       0       0   ...    0       0       0 ]
         [-Phi[1]    1       0       0   ...    0       0       0 ]
  Phi1 = [-Phi[2]  -Phi[1]    1       0   ...    0       0       0 ]
         [   0     -Phi[2]  -Phi[1]    1   ...    0       0       0 ]
         [ ..................................................... ]
         [   0       0       0       0   ... -Phi[2]  -Phi[1]    1 ]
See also solve().

NOTE: The sign assumed for Phi is not affected by variable ARSIGN which
is recognized by several macros in file Arima.mac.  Type
arimahelp(MASIGN) for details.

When Phi is NULL, the result is the same as A, stripped of labels or
notes, if any.  Also, the result is a true vector or matrix (ndims = 1
or 2).

                          Inverse difference operator
A common usage is autoreg(1,x), where x is a vector or matrix.  This
```

```
computes the partial sums x[1,], x[1,]+x[2,], ..., sum(x).  A useful
macro might be defined by
    partialsum <- matrix("autoreg(1,$1)")

                            Keyword reverse
autoreg(Phi,A,reverse:T) applies the autoregressive operator in reverse:
    x[i] = A[i] + sum(Phi[k]*x[i+k],1<=k<=nrows(Phi)),
with x[l] = 0 for l > nrows(A).

                            Keyword seasonal
autoreg(Phi,A,seasonal:L [,reverse:T) does the same, except that the
computations are of the forms
    x[i] = A[i] + sum(Phi[k]*x[i-k*L],1<=k<=nrows(Phi)).
or
    x[i] = A[i] + sum(Phi[k]*x[i+k*L],1<=k<=nrows(Phi)) (reverse:T)

                            Start and limits
autoreg(Phi,A,limits:vector(i1,i2),start:StartVals [,reverse:T,
seasonal:L]) is the same except that x[i] is computed as described only
for i1 <= i <= i2, with the remaining values copied before the
computation from rows 1 to i1-1 and rows i2+1 to nrows(A) of matrix
StartVals.  The values in rows 1 through i1-1 of StartVals serve as
"starting values" for the autoregressive operator.  When reverse:T is an
argument, rows nrows(A) through i2+1 serve as starting values.  This
feature is useful for generating out of sample forecasts or "backcasts".

The value for limits can also be a scalar j between 1 and nrows(A).  In
this case, with reverse:T, i1 = 1, i2 = j, and without reverse:T, i1 =
j, i2 = nrows(A).

StartVals must have the same number of columns as A and usually has the
same number of rows.  When nrows(StartVCals) != nrows(A), without
reverse:T, i2 must be nrows(A) and with reverse:T, i1 must be 1.  In
this case, the elements of StartVals, which are used as starting values,
are copied to the rows not included between i1 and i2 and hence
nrows(start) must match nrows(A) - (i2 - i1 + 1).  This feature allows
you to compute autoregressive predictions up to 20 time units ahead,
say, by
  Cmd> autoreg(phi,rep(0,length(x) + 20),limits:length(x)+1,start:x)

autoreg() is the inverse of movavg() and vice versa, in that
   autoreg(phi,movavg(phi,x))  and   movavg(phi,autoreg(phi,x))
both reproduce x, except for rounding error.

                                Examples
Examples:
  Cmd> autoreg(phi,rnorm(400))[-run(100)]
    generates an autoregressive series with normal innovations,
    discarding the first 100 values to avoid transients.
  Cmd> autoreg(phi,matrix(rnorm(4000),400)[-run(100),]
    generates 10 independent autoregressive series at once.
  Cmd> autoreg(hconcat(phi1,phi2),rnorm(400))[-run(100)]
    generates 2 autoregressive series with different coefficients but
```

```
    the same innovations
  Cmd> autoreg(.3, autoreg(-.1,rnorm(230),seasonal:4))[-run(30)]
     generates a (1,0,0)x(1,0,0)-4 seasonal ARMA time series
  Cmd> autoreg(vector(1,1),padto(1,20))
     computes the first 20 Fibonacci numbers F(j) satisfying F(j) =
     F(j-1) + F(j-2) with F(1) = 1, F(0) = 0

                              Cross references
See also movavg().
```

2.30 batch()

Usage:

```
   batch(fileName [,echo:T or F, prompt:string]), CHARACTER scalars
     fileName and string
```

Keywords: syntax, control, files

```
                                    Usage
batch(FileName) executes the commands in the file with name given in the
quoted string or CHARACTER variable FileName.  It must be the last
command in a line or a compound command surrounded by '{' and '}' and
must not be in a loop.

In windowed version, if FileName is "" you will be prompted to enter
the file name in a dialog box.

Lines of the file are read sequentially and executed as if they were
typed at the keyboard.  Normally, each line is printed with the file
name as prompt before it is executed.  You can suppress this by using
keyword phrase echo:F (see below), or by previously executing
setoptions(batchecho:F) (see topics setoptions() and 'options').

The batch file can contain any sequence of MacAnova commands.  This
includes additional batch() commands that do not read from a batch file
currently in use.

                                   Example
Here is an example of a short batch file designed to do cubic regression
of variable y on variable x and do a plot of residuals against x (see
also regress() and plot()):

   xsq <- x * x
   xcub <- x * xsq
   regress("y=x + xsq + xcub")
   plot(x, RESIDUALS, title:"Cubic regression residuals vs x")

                              Treatment of errors
When an error occurs, the default behavior is to terminate all current
batch() commands.  You can use setoptions(errors:N), where N is a
positive integer to increase the number of errors tolerated before
```

```
termination.  See topic 'options"errors"' for details.

                                Keywords
batch(FileName,echo:F) works the same as batch(FileName) except the
prompts and lines read from the file are not printed.  This status is
inherited by batch files invoked from within a batch file.  You can use
setoptions(batchecho:F) to set the default behavior of batch() so that
lines will not be echoed.

batch(FileName,echo:T) forces the printing of prompts and commands, even
if option 'batchecho' has been set False (see subtopic
'options:"batchecho').

NOTE: If you want to suppress printing of both prompts and output, put
  setoptions(quiet:T)
at the start of the file and
  setoptions(quiet:F)
at the end of the file.  See subtopic 'options:"quiet"'.

batch(FileName,prompt:Prompt), where Prompt is a quoted string or
CHARACTER scalar, forces echoing, with command lines starting with
Prompt instead of the file name.  When the batch file contains a
setoptions(prompt:newPrompt) command, newPrompt overrides Prompt.  A
subsequent setoptions(default:T) in the file, restores Prompt.  See
topics setoptions(), 'options'.

                            Cross references
See also topic 'launching'.
```

2.31 bcprd()

Usage:

```
   bcprd(x), REAL matrix x
   bcprd(x1, x2, ... ), x1, x2, ... REAL matrices with the same number of
     rows.
```

Keywords: matrix algebra, glm

```
                                  Usage
bcprd(x) where x is a matrix computes a "bordered" cross product matrix
containing the means of the columns of x and mean-corrected sums of
squares and products of the columns of x.

bcprd(x1,x2,...,xm) yields the same result as bcprd(hconcat(x1,x2,...,
xm)) when x1, x2, ... are all REAL matrices with the same number of
rows.

Specifically, when x is an n by p matrix, bcp <- bcprd(x) sets bcp to
a p+1 by p+1 matrix, where
  bcp[1,1]   = 1/n
  bcp[-1,1]  = a column vector containing the sample mean xbar
```

```
  bcp[1,-1]  = a row vector containing -xbar'
  bcp[-1,-1] = the p by p matrix of mean-corrected sums of squares and
               products of the columns of x

                          Relationship with swp()
bcprd(x) is mathematically equivalent to
  {@TMP <- hconcat(rep(1,nrows(x)),x); swp(@TMP %c% @TMP,1)}.
However, the use of bcrpd() is preferred to the illustrated use of swp()
since it uses a numerically stable algorithm to compute the corrected
sums of squares and products.

                                   Labels
When all the arguments of bcprd() have labels, so will the result of
bcprd() with both row and column labels taken from the column labels of
the arguments.

                               Cross references
See also swp().
```

2.32 bin()

Usage:

```
   bin(x,Bnds [,silent:T,leftendin:T]), x a REAL matrix, Bnds REAL vector,
      Bnds[k] < Bnds[k+1]
   bin(x,vector(anchor,width) [,leftendin:T]), anchor and width > 0 REAL
      scalars
   bin(x,nbins, [leftendin:T]), nbins positive integer.
   bin(x [,leftendin:T])
```

Keywords: categorical data, summary statistics

```
                                     Usage
bin(x,Edges) counts the number of values v in each column of REAL vector
or matrix x in class intervals defined by the elements of REAL vector
Edges with length(Edges) > 2.  Edges must satisfy Edges[k] < Edges[k+1],
k = 1, ..., nclasses = length(Edges) - 1.  When x is a matrix, counts
are computed for each column of x.

The value of bin(x,Edges) is structure(boundaries:edges, counts:M) where
edges is a vector of length nclasses + 1 containing the class limits and
M contains counts.  In this usage, edges is identical to Edges.  When x
is a vector, M[k] = (number of values of x in class k).  When x is a
matrix, M is a matrix with nclasses rows and ncols(x) columns with
M[k,j] = (number of values in column j of x in class k).

The count for class k is the number of values v with Edges[k] < v <=
Edges[k+1], k = 1, ..., nclasses, that is any value equal to the right
end of an interval is counted in that interval.

When any element of x is <= Edges[1] or > Edges[nbins+1], it is not
included in the count and a warning message is printed.
```

```
                          Keyword 'leftendin'
bin(x,Edges,leftendin:T) does the same except a value v is counted in
class k if Edges[k] <= v < Edges[k+1], that is, a value equal to the
left end of a class is counted as being in the class.  v is not counted
when v < Edges[1] or v >= Edges[nbins+1].  'leftendin:T' can be used
with all variants of bin() arguments

                           Keyword 'silent'
bin(x,Edges,silent:T) does the same except any warning messages are
suppressed.  'silent:T' can be used with all variants of bin()
arguments.

                          Equal width classes
bin(x,vector(anchor, width)), where anchor and width are scalars, does
the same, except that the class boundaries are of the form anchor +
k*width, where k is an integer, with the minimum and maximum values of k
selected so as to include all the data in x.  For example, bin(x,
vector(.5,1)) would use class intervals of width 1 centered at integers.

bin(x,nbins) does the same, except that boundaries for nbins classes are
computed so that the classes have equal widths and include all the data.
The class boundaries will normally not be "neat".

bin(x) is equivalent to bin(x, ceiling(log(nrows(x))/log(2))+1), that is
the number of bins is approximately log2(nrows(x)).

You can use bin() to find counts needed to draw a histogram or to
compute a chi-squared test of goodness of fit of a sample to a
theoretical distribution.
```

2.33 bit_ops

Usage:

```
   a %| b, a %^ b, a %& b and %! a, where a and b are REAL or structures
     with REAL components with integer elements >= 0 and <= 4294967295 =
     2^32-1 nbits(x)
```

Keywords: operations, glm, missing values

```
There are 4 operators for working with integers considered as the sets
of 32 bits specified by their binary representations.

                             Bit operations
Bit Operation  Precedence  Meaning
    a %| b         1       Bitwise Or (OR)
    a %^ b         2       Bitwise Exclusive Or (XOR)
    a %& b         3       Bitwise And (AND)
      %!a          4       Bitwise Complement (COMPL)

When an operand x is not an integer or x < 0 or x > 4294967295 = 2^32-1,
```

```
the result of any of these operators is MISSING.

For '%&', a bit of the result is 1 if and only if the corresponding bits
in the operands are both 1.
  Example: 25 %& 19 is 17 because 11001b AND 10011b is 10001b

For '%|', a bit of the result is 1 if and only if at least 1 of the
corresponding bits in the operands is 1.
  Example: 25 %| 19 is 27  because 11001b OR 10011b is 11011b

For '%^', a bit of the result is 1 if and only if exactly 1 of the
corresponding bits in the operands are 1, that is, if the corresponding
bits differ.
  Example: 25 %^ 19 is 10  because 11001b XOR 10011b is 01010b

Operator '%!' operates on the immediately following variable considered
as a collection of 32 bits, changing 1's to 0's and 0's to 1's.
  Examples:
       %! 25 is 4294967270 since COMPL(00000000000000000000000000011001b)
          is 11111111111111111111111111100110b
       %! 0  is 4294967295 since COMPL(00000000000000000000000000000000b)
          is 11111111111111111111111111111111b

When an operand is LOGICAL, it is treated as having value 0 (F) or 1
(T).  The result is always REAL.

When any operand is MISSING, so is the result.

                          Use with modelinfo()
Bit operators were introduced to be useful with the output of
modelinfo(bitmodel:T).  For example, 2^(i-1) %& modelinfo(bitmodel:T)[j]
is non-zero if and only if the j-th term of the model contains the i-th
factor or variate (assuming i <= 32).

The operators were listed above in increasing order of precedence.
Moreover, they have lower precedence that all arithmetic, comparison, or
logical operators which means they are evaluated after all such
operators.

                               Examples
  Examples:
    Expression                Interpretation              Value
    17 %| 29 %^ 91 %& 11      17 %| (29 %^ (91 %& 11))    23
    %!21 %| 97 %& %! 33       (%!21) %| (97 %& (%!33))    4294967274
    1 %& 3 + 4                1 %& (3+4)                  1
    3 %^ 5 != 6               3 %^ (5 != 6)               2
    1 %| 2 == 3               1 %| (2 == 3)               1
    %!0 == 4294967295         %!(0 == 4294967295)         4294967295

To understand the last three examples, note that 5 != 6 is True and is
interpreted as 1, and that 2 == 3 and 0 == 4294967295 are both False and
are interpreted as 0. See topics 'arithmetic' and 'logic'.
```

Cross references

See topic 'arithmetic' for a description of the "shape" of the result when operands are not scalars.

See also modelinfo(), nbits().

2.34 boxcox()

Usage:

boxcox(x,power), x a REAL vector or matrix, power a REAL scalar

Keywords: transformations

Usage

boxcox(var,Pow) computes the Box-Cox transformation of the data in vector or matrix var. When var is a matrix, the transformation is applied to each column separately. If GM is the geometric mean of the values in a vector, boxcox(y,Pow) computes (y^Pow-1)/(Pow*(GM)^(Pow-1)) when Pow != 0, and GM*log(y) when Pow == 0. Boxcox is implemented as a macro.

Cross references

See also topics 'macros' and 'transformations'.

2.35 boxplot()

Usage:

boxplot(x1,x2,...,xk [,vs:indv, boxsize:W] [,vertical:T, excludeM:T, boxtype:m, symbols:outlierSyms, graphics keyword phrases]), x1,...,xk REAL vectors, indv REAL length k vector with no MISSING values, m > 0 integer, W REAL non-negative vector or scalar, outlierSyms CHARACTER scalar or vector of length 2

boxplot(Struc, [,vs:indv, boxsize:W] [,vertical:T, excludeM:T, boxtype:m, symbols:outlierSyms, graphics keyword phrases]), Struc a structure with k REAL vector components

Keywords: plotting, descriptive statistics

Usage

boxplot(x1, x2, ... , xk) produces horizontal parallel Tukey boxplots for the vectors x1 through xk and plotting positions 1, 2, ..., k on the y axis.

boxplot(x1, x2, ... , xk,vertical:T) and boxplot(Struc,vertical:T) do the same except the boxplots are aligned vertically at plotting positions 1, 2, ..., k on the x axis. Pre-defined macro vboxplot() which is used identically to boxplot(), makes use of the feature to make vertical boxplots.

boxplot(x1, x2, ..., xk, vs:Predictor [,vertical:T] ...) does the same except the boxes are aligned with Predictor[1], Predictor[2], ..., Predictor[k] on the y or x axis. Predictor must be a REAL vector with no MISSING values with length(Predictor) = k.

boxplot(x1, x2, ..., xk, boxsize:W, ...) does the same except the thickness of the boxes is determined by non-negative REAL scalar or vector W. See below for details.

boxplot(x1, x2, ..., xk, boxtype:m, ...) does the same except the style of the boxplot is determined by integer m > 0. The default type corresponds to m = 1. Whiskers extend to the most extreme values inside the inner "fences", and values beyond the inner and outer fences are individually plotted with special symbols.

With boxtype:2, the box plot is a 5 number summary box plot, with whiskers with cross bars at the end extending from the ends of the box to the extremes. No outliers are indicated. At present, boxtype:m with m > 2 yields as boxtype:1.

boxplot(Struc, ...) produces parallel box plots for the components of structure Struc, all of which must be REAL vectors. You can use any boxplot() or graphical keywords.

Keyword 'symbols'

Keyword 'symbols' has a different meaning from other plotting commands. You use it to specify symbols for moderate outliers (beyond inner fences and inside outer fences) and extreme outliers (beyond outer fences). The value of 'symbols' must be a CHARACTER scalar or vector of length 2.

```
Cmd> boxplot(x1, x2, x3, symbols:vector("\3", "\5"))
```

uses "\3" (square) as a symbol for moderate outliers and "\5" (triangle) as a symbol for extreme outliers. When the value of 'symbols' is a scalar, the default symbol is used for extreme outliers.

Use with split()

boxplot(split(y,a) [,vertical:T] ...) draws parallel box plots of the data in vector y classified according to levels of factor a. See split().

boxplot(split(y) [,vertical:T], ...) draws parallel box plots of the data in each column of matrix y.

These work because split() returns a structure and each component of a structure gets its own box.

Size of boxes

You can use keyword phrase boxsize:W, where W is a non-negative scalar or vector to specify the "thickness" of the boxes (height for horizontal boxes, width for vertical boxes). When W is not a scalar, length(W) must match the number of boxes. When W is a vector and W[j] = 0, box j

```
is omitted.

When you don't use keyword 'boxsize', the default thickness is such that
about 2/3 of the space between the first and last box is made of up
boxes and 1/3 of interbox space.

                                  Keywords
When keyword phrase 'excludeM:T' is an argument, and the number of
non-MISSING values in a sample is odd, the median is omitted in
calculating the quartiles as the medians of the upper and lower halves.

You may use keywords 'dumb', 'xmin', 'xmax', 'ymin', 'ymax', 'logx',
'logy', 'xlab', 'ylab', 'title', 'xaxis', 'yaxis', 'borders', 'ticks',
'xticks', 'yticks', 'xticklen', 'yticklen', 'xticklabs', 'yticklabs',
'height', 'width', 'pause', 'silent' and 'notes' as for other plotting
commands.  See topics 'graph_keys', 'graph_border' and 'graph_ticks'

When option 'dumbplot' has been set False (see subtopic
'options:"dumbplot"'), the plot will be a low resolution plot unless
'dumb:F' is an argument.

Note: Using 'logx:T' and/or 'logy:T' affects only the scaling used in
plotting, not the determination of outliers.  With logy:T without
vertical:T or logx:T with vertical:T'q, the thickness of the boxes will
be affected since the edges are equally distant from the middle in
arithmetic units but not in logarithmic units

                              Cross references
See topic 'graph_files' for information on how to save a boxplot in a
file using keywords 'file', 'new, 'ps', 'screendump', and 'epsf'.

See also topics showplot(), 'structures', 'graph_keys'.
```

2.36 break

Usage:

```
for(i,run(n)){if(x[i] < 0){break} .... }
for(i,run(n)){for(j,run(m)){if(x[i,j] < 0){break 2} .... }}
```

Keywords: control, syntax

```
                                    Usage
'break' and 'break n' are used to exit prematurely from one or more
enclosing loop, perhaps because an error has been found.

'break', in a 'for' or 'while' loop, exits the loop, skipping any
remaining commands in the loop and resuming execution immediately after
the '}' terminating the loop.  When more than one loop "encloses"
'break', only the innermost one is exited.

'break n', where n is a positive integer, exits from n enclosing 'for'
```

```
or 'while' loops.  For example, 'break 1' is equivalent to 'break' and
will exit the current loop; 'break 2' will exit the current loop and the
loop enclosing it; and so on.  n must be a literal integer ('1', '2',
...)  and not a variable with integer value.

It is an error to use 'break' outside of a loop or to use 'break n' when
not enclosed in at least n loops.

                      In macro or evaluated string
In an evaluated string or out-of-line macro, 'break' and 'break n' can
be used only to exit from a loop that started in the macro or evaluated
string.  It is an error to try to exit from a loop that started outside
the macro or evaluated string.  See evaluate() and 'macros'.

Using 'break' in an in-line macro to exit from a loop that started
outside the macro will work, but is a bad programming practice.

Inside a macro, break n with the appropriate value of n should always be
used instead of breakall.

                                 Examples
Examples:
  for(i,run(100)){... compute x ...;if(x<0){print("x < 0");break;};...;}
If x ever becomes negative, the 'for' loop is terminated.
  for(i,run(10)){for(j,run(5)){...;if(x<0){break 2}}}
If x ever becomes negative, both 'for' loops are terminated.

                            Cross references
See also topics 'if', 'for', 'while', 'breakall', 'next', batch().
```

2.37 breakall

Usage:

```
for(i,run(n)){if(x[i] < 0){breakall} .... }
for(i,run(n)){for(j,run(m)){if(x[i,j] < 0){breakall}}}
```

Keywords: control, syntax

```
'breakall' is used in 'while' and 'for' loops to exit prematurely from
any and all "enclosing" 'while' or 'for' loops.  Execution resumes
immediately after the '}' terminating the most inclusive 'while' or
'for' loop currently in effect.

'breakall' should normally not be used inside a macro, since if such a
macro were invoked in a loop at the prompt level, 'breakall' would exit
from that loop as well as any loops in the macro.  This would seldom be
what you want.  Instead, use 'break n', where n is a positive integer
specifying the number of loops to exit.  See 'break'.

It is an error to use 'breakall' outside of a loop.
```

```
In an evaluated string or out-of-line macro, 'breakall' will exit the
outermost loop that started inside the evaluated string or macro.  It is
an error if there is no such enclosing loop.  See evaluate() and
'macros'.

In an in-line macro (the default), 'breakall' will exit not only loops
which start in the macro, but also any loops which enclose the macro.
If this is not what is wanted (and it usually wouldn't be), use 'break
n', where n is the number of loops enclosing 'break' in the macro.
Use 'return' to exit from a macro prematurely.

                                Example
Example:
  Cmd> for(i,run(m)){
          for(j,run(n)){... compute x ...;if(x<0){breakall};}}

If x ever becomes negative, both 'for' loops are immediately terminated.
Using 'break' instead of breakall would mean that only the inner loop
(for(j,run(n)){...}) would be terminated.

                            Cross references
See also topics 'if', 'for', 'while', 'break', 'next', 'return'.
```

2.38 breakif()

Usage:

```
for(i,run(n)){breakif(x[i] < 0) .... }
for(i,run(n)){for(j,run(m)){breakif(x[i,j] < 0, 2) ...}}
```

Keywords: control, syntax

```
                                Usage
breakif(Logical) is equivalent to if(Logical){break;} .
breakif(Logical,n) is equivalent to if(Logical){break n;} .

It is implemented as a pre-defined in-line macro.  It can be used only
inside a loop and n must not exceed the number of containing loops.

                                Example
Example:
  Cmd> for(i,run(length(x))){breakif(abs(x[i]) > 3)}
computes the index i of the first element on vector x to exceed 3 in
absolute value.

                            Cross references
See also topics 'break', 'breakall', 'next'.
```

2.39 callback_fun

Keywords: general, control

```
This topic is in file userfun.hlp.  Type
  userfunhelp(callback_fun)

It provides a brief introduction to the form of a user function that
makes "call backs" (executes functions internal to MacAnova).

Some other useful entries in userfun.hlp are arginfo_fun and user_fun.
Type
  userfunhelp()
for a complete list of entries.
```

2.40 carapace

Usage:

```
   Type help(carapace) for information about windowed versions of MacAnova
```

Keywords: general

```
Windowed forms of MacAnova from version 5.00 onwards use the Carapace
and wxWidgets libraries to provide graphical user interfaces.  This
help topic describes features common to those versions.

There are several other related help topics that you may wish to
explore.  Topic 'nongui' describes non-windowed MacAnova.  Topics
'dos_windows', 'macintosh', and 'unix' describe specifics for those
platforms.  Topics 'launching' and 'customize' describe the use of
command line options, MacAnova options, and initialization files.

Windowed forms of MacAnova allow multiple command/output and high
resolution graphics windows and also use menus, dialogs, the mouse,
and so forth in the standard way.  A command window has two panes,
a lower pane, into which commands are typed, and an upper pane,
into which results are printed. Output and graphics windows can be
printed and/or saved to files.

When you type into the command window, any unmatched quotes, braces,
brackets, or parentheses will be highlighted.  If you specify a
filename as "", then a dialog box will be opened to allow you to
select the file interactively.

                            The Clipboard
Text in a command window can be copied to the clipboard.  Content of
a graphics window can be copied to the clipboard as a bitmap.  The
MacAnova variable CLIPBOARD is connected to text on the clipboard in
the sense that accessing CLIPBOARD returns a MacAnova string containing
the text content of the clipboard, and assigning to CLIPBOARD writes
```

text to the clipboard.

Command Window Menus

File Menu Items

Open	Open a text file in a new command window.
Save	Save the output pane of this window to a file.
Save Window As	Save, but allow a name change.
Page Setup	Set up for printing.
Print Window	Print the output pane of this command window.
Interrupt	Stop processing the current MacAnova command.
Restore	Restore a MacAnova workspace from a file.
Save Workspace	Save your workspace to a file.
Save Workspace As	Save your workspace to a file, changing the name.
Quit	Quit MacAnova

PLEASE NOTE: saving the window saves your output but not your data. Saving the workspace saves your data but not your output. You may wish to do both.

Edit Menu Items

Undo/Redo	Undo or redo last change in output pane.
Cut	Cut text.
Copy	Copy text.
Paste	Paste text.
Copy to Command	Copy the selection to the command pane
Execute	Copy the selection to the command pane and execute
Back History	Move back through command history
Forward History	Move forward through command history

Windows Menu Items

Hide	Hide this window.
Close	Close this window.
New	Open a new command window.
Output Windows	A submenu allowing you to access command windows.
Graph Windows	A submenu allowing you to access graphics windows.
Set Font	Change the font in this command window.
Scroll to Top	Scroll to the top of the output pane.
Scroll to Bottom	Scroll to the bottom of the output pane.
Move to Command	Move keyboard and pointer focus to the lower pane.

Help Menu Items

Help	General help on MacAnova, or help on the selection.
About	Information about MacAnova.

Keyboard equivalents.
There are keyboard equivalents for many of the menu commands. For example, up and down arrows in the command pane do backward and forward in the command history.

Button equivalents.
Each command window has six buttons across the bottom. Five of these implement the undo, execute, back history, forward history, and interrupt menu items. The sixth is Clear, which clears the text

```
in the command pane.

On the same row with the buttons are two status displays.  The first
indicates when MacAnova is processing a command by displaying the text
"Running".  The second tells you how far back you are in the command
history at any given point.

                          Graphics Window Menus
File Menu Items
    Save Graph As     Save the graph as jpeg, tiff, png, or PostScript.
    Page Setup        Set up for printing.
    Print Window      Print the graph.
    Interrupt         Stop processing the current MacAnova command.
    Quit              Quit MacAnova

Edit Menu Items
    Copy              Copy the graph to the clipboard as a bitmap.

Windows Menu Items
    Hide              Hide this window.
    Close             Close this window.
    Zoom in           Zoom in on the graph
    Zoom out          Zoom out on the graph
    Fit               Redraw the graph to fit the window exactly.
    Fit keep aspect   Redraw the graph but maintain the aspect ratio.
    Set aspect ratio  Set the aspect ratio for the graph.
    Output Windows    A submenu allowing you to access command windows.
    Graph Windows     A submenu allowing you to access graphics windows.
```

2.41 cat()

Usage:

```
   cat(x1,x2,...,xk [,KeyPhrases]) where x1, x2, ... all have the same
     type, REAL, LOGICAL, or CHARACTER, or are structures with components
     all of the same type
   KeyPhrases can be labels:lab and/or silent:T, where lab is a CHARACTER
     scalar or vector.
```

Keywords: variables, combining variables, character variables, null variables

```
cat() is identical to vector().  See vector() for information on its
use.  See topic 'vectors' for general information on vectors.

The use of cat() is deprecated -- that is, it will continue to be
available for the immediate future, but at some point may be disabled.
Use vector() instead.
```

2.42 cconj()

Usage:
```
cconj(cx), cx a REAL matrix representing complex data
```

Keywords: time series, complex arithmetic

```
                                  Usage
cconj(cx) returns the complex conjugates of successive pairs of columns
of the matrix cx, considered as the real and imaginary parts of complex
series.  The real and imaginary parts of the results are in alternating
columns.

When cx has an odd number, say 2*m-1, of columns, the last column is
interpreted as the real part of a complex series with zero imaginary
part.  In the result, an extra column of zeros is added, so the result
has 2*m columns representing m complex series with both real and
imaginary parts.

                             Cross references
See also hconj(), hreal(), himag(), creal(), cimag().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.43 cdivc()

Usage:
```
cdivc(cx1 [, cx2]), cx1 and cx2 REAL matrices representing complex data
```

Keywords: time series, complex arithmetic

```
                                  Usage
cdivc(cx1, cx2) computes the element wise complex ratio of fully complex
(pairs of columns constitute real and imaginary parts) matrices cx1 and
cx2.  When either of cx1 or cx2 has an odd number of columns, it is
augmented with a column of zeros before division.

Any ratio of the form (0 + 0i)/(0 + 0i) is returned as 0 + 0i.  The ratio
of a non-zero element of cx1 and 0 + 0i is MISSING + MISSING*i.

cdivc(cx) is equivalent to cdivc(cx,cx), returning a result all of whose
elements are 1 + 0i, except for (0 + 0i)/(0 + 0i) ratios which are 0 +
0i.

When nrows(cx1) > 1 and nrows(cx2) > 1, nrows(cx1) = nrows(cx2) is
required.  Otherwise, the single row in the short argument is implicitly
duplicated to match the number of rows in the other argument.
```

```
When cx1 represents a single complex series (ncols(cx1) <= 2), that
series is divided by all the series in cx2.  Similarly when ncols(cx2)
<= 2, all the series in cx1 are divided by cx2.

                                Examples
Examples:
  When ncols(cx1) = 2 and ncols(cx2) = 5, cdivc(cx1,cx2) is eqivalent to
  cdivc(hconcat(cx1,cx1,cx1),hconcat(cx2,rep(0,nrows(cx2)))).

  cdivc(1,cx) computes the complex reciprocal of cx.

                            Cross references
See also cdivcj(), hdivh(), hdivhj(), cprdc(), cprdcj(), hprdh(),
hprdhj(), hconcat(), rep(), nrows(), ncols().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.44 cdivcj()

Usage:

```
cdivcj(cx1 [, cx2]), cx1 and cx2 REAL matrices representing complex data
```

Keywords: time series, complex arithmetic

```
                                  Usage
cdivcj(cx1, cx2) computes the element wise complex ratio of fully
complex (pairs of columns constitute real and imaginary parts) matrices
cx1 and cconj(cx2).  When either cx1 or cx2 has an odd number of
columns, it is augmented with a column of zeros before division.

Any ratio of the form (0 + 0i)/(0 + 0i) is returned as 0 + 0i.  The ratio
of a non-zero element of cx1 and 0 + 0i is MISSING + MISSING*i.

cdivcj(cx) is equivalent to cdivcj(cx,cx).

When nrows(cx1) > 1 and nrows(cx2) > 1, nrows(cx1) = nrows(cx2) is
required.  Otherwise, the single row in the short argument is implicitly
duplicated to match the number of rows in the other argument.

When cx1 represents a single complex series (ncols(cx1) <= 2), that
series is divided by the complex conjugates of all the series in cx2.
Similarly when ncols(cx2) <= 2, all the series in cx1 are divided by
cconj(cx2).

                                Examples
Examples:
```

```
  When ncols(cx1) = 2 and ncols(cx2) = 5, cdivcj(cx1,cx2) is eqivalent
  to cdivcj(hconcat(cx1,cx1,cx1),hconcat(cx2,rep(0,nrows(cx2)))).

  cdivcj(1,cx) computes the complex reciprocal of cconj(cx).

                            Cross references
See also cdivc(), hdivh(), hdivhj(), cprdc(), cprdcj(), hprdh(),
hprdhj(), cconj(), hconcat(), rep(), nrows(), ncols().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.45 ceiling()

Usage:

```
   ceiling(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                  Usage
ceiling (x) rounds the elements of the REAL variable x to the next
integer in the positive direction, producing a vector, matrix, or array
with the same shape as x.

                                 Example
Example:
  Cmd> ceiling(vector(3.1416, -3.1416, 12))
  (1)           4          -3          12

                           Argument too large
When x > 4503599627370495 or x < -4503599627370495, ceiling(x) is set to
MISSING because of the impossibility of exact representation of integers
beyond these limits.  These limits may be different on some computers.

                           Structure argument
When x is a structure consisting of REAL components, so is ceiling(x).
If the i-th component of x is xi, the i-th component of ceiling(x) is
ceiling(xi).

                            Cross references
See also topics floor(), round(), 'structures'.
```

2.46 cellstats()

Usage:

```
   cellstats(Term), Term a CHARACTER scalar of form "A.B. ...", where A, B,
      ...  are factors in current GLM model.
```

Keywords: descriptive statistics, anova

```
                                Usage
cellstats(Term) computes statistics for each cell of the multiway layout
indicated by the term in the CHARACTER variable Term. The term must be
made of factors in the model used by the most recent GLM (generalized
linear or linear model) command such as regress(), anova(), or
poisson().  It omits all cases for which there is any MISSING data in
either the left or right hand sides of the model.

                        Difference from tabs()
cellstats() and tabs() do almost the same thing and generally tabs() is
to be preferred.  When Term is, say, "a.b.c" where a, b, and c are
factors in the most recent GLM, and y is the response variable in the
model, cellstats(Term) is almost equivalent to tabs(y, a, b, c).

cellstats() and tabs() will differ only when (a) the response variable y
is multivariate (has more than 1 column) and (b) there are MISSING data
in y.  cellstats() omits completely any row of y that contains any
MISSING data; tabs() uses all non-MISSING data available and thus the
cell count can differ among the columns of y.  Except in this case, you
should probably use tabs() in preference to cellstats() since tabs() has
additional options and can be used independently of any GLM command.
Even in this case, tabs() is preferable if you want cell statistics that
use all the data.

Of course, both cellstats() and tabs() omit all cases for which any of
the factors are MISSING (how could they determine the cell?).

                                Example
Example:
  Cmd> anova("y=a+b+c+b.c") ; cellstats("b.c") # or tabs(y,b,c)
gives cell statistics for the b.c term.

                            Cross references
See also topics 'glm', tabs().
```

2.47 cft()

Usage:

```
   cft(cx [,divbyT:T]), cx a REAL matrix representing complex data
```

Keywords: time series, complex arithmetic

```
                                Usage
cft(cx) where cx is a REAL vector or matrix, computes the fully complex
form of the discrete Fourier transforms of successive pairs of columns
of cx, considered as the real and imaginary parts of complex series.
The real and imaginary parts of the results are in alternating columns.
```

Any MISSING values in cx are replaced by 0 in computing the result and a warning message is printed.

cft(cx,divbyt:T) does the same except the transform is divided by the number of rows of cx.

Inverse transform

cconj(cft(cconj(cx),divbyt:T)) is the inverse of cft() in the sense that cx and cconj(cft(cconj(cft(cx)),divbyt:T)) are equal except for rounding error.

Limitation on length

The largest prime factor of nrows(cx) must not exceed 29. You can use primefactors() to find the maximum factor of nrows(cx) and goodfactors() to find a length >= nrows(cx) which has no prime factors > 29. In addition, the product of all the "unpaired" prime factors can't be too large. For example N = 3*5*7*11*13*17*M^2 = 255255*M^2, where M is an integer, breaks the algorithm and hence is not allowed.

Cross references

See topic 'complex' for discussion of complex matrices in MacAnova.

See also hft(), rft(), cconj(), primefactors(), goodfactors().

See subtopic 'matrices:"complex_matrices" for a list of macros for working with complex matrices.

2.48 changestr()

Usage:

```
changestr(Struc,name,x), Struc a structure, name a CHARACTER  scalar, x
  a defined variable
changestr(Struct,n,x), integer n, 1 <= n <= ncomps(Struct) + 1
changestr(Struct,name:x)
changestr(Struc, -n), n a positive integer
```

Keywords: structures

Change component by name

changestr(Str,Name,x) makes a copy of structure Str except that the value of component Name is changed to x. Name must be a quoted string or CHARACTER scalar of no more than 12 characters. It is an error if Name contains a space, '$' or any "control character" (ASCII code <= 31 or 127). If there is no component with name Name, x will be added as a new component. It will have name Name unless x is a keyword phrase

changestr(Str,Name,newname:x) does the same, except the changed or added component has name 'newname'.

changestr(Str,compname:x) is the same as changestr(Str,"compname", x).

```
compname must have no more than 10 characters.

If you want to change a named component of a structure, say component
var of structure stats, it is better to use 'stats$var <- x' than
stats <- changestr(stats,"var", x).

                        Change component by number
changestr(Str,CompNumber,x), where CompNumber is a positive integer,
does the same, except the component to be changed is specified by number
rather than by name.  When CompNumber = ncomps(Str) + 1, a new component
with value x is added.  It is an error if CompNumber > ncomps(Str)+1.

changestr(Str,CompNumber,newname:x) does the same except the changed or
added component will have name 'newname'.

To change a component of a structure by number, it is preferable to us
Str[CompNumber] <- x rather than Str <- changestr(Str,CompNumber,x).

Moreover, you can change more than one component at a time by assigning
to subscripts:
  Cmd> Str[run(2)] <- vector(Pi,PI^2)
changes components and and 2 of Str without changing there names.

                           Deleting component
changestr(Str,-CompNumber) produces a new structure omitting component
CompNumber.  It is illegal to delete the only component of a structure.

Preferable to this usage is Str[-CompNumber].  Moreover, you can omit
several components by, say, Str[-run(2)].

                      Modificication using assignment
When Str is a structure with a component Name, Str$Name <- x changes
component Name, without making a temporary copy of Str.

When J is a positive integer <= ncomps(Str), Str[J] <- x changes
component J, without making a temporary component of Str.  You can
change more than component of Str when J is a vector of subscripts.  See
topic 'assignment'.

                                 Examples
Examples:  In each of the following groupings, all the commands return
the same structure:
  Cmd> changestr(structure(a:run(10),b:"Hello"),"a",PI)
  Cmd> changestr(structure(a:run(10),b:"Hello"),1,PI)
  Cmd> structure(a:PI,b:"Hello")

  Cmd> changestr(structure(a:run(10),b:"Hello"),"a",pi:PI)
  Cmd> changestr(structure(a:run(10),b:"Hello"),1,pi:PI)
  Cmd> structure(pi:PI,b:"Hello")

  Cmd> changestr(structure(a:run(10),b:"Hello"),3,c:"Dolly")
  Cmd> changestr(structure(a:run(10),b:"Hello"),c:"Dolly")
  Cmd> structure(a:run(10),b:"Hello",c:"Dolly")
```

```
  Cmd> changestr(structure(a:run(10),b:"Hello"),-1)
  Cmd> structure(b:"Hello").

                          Cross references
See also topics 'structures', 'keywords', structure(), strconcat().
```

2.49 cholesky()

Usage:

```
   cholesky(x [,pivot:T or force:T , nonposok:T]), x a positive definite
     square REAL matrix with no MISSING values
```

Keywords: matrix algebra

```
                                  Usage
cholesky(A) returns the Cholesky decomposition of the positive definite
REAL symmetric matrix A.  Its value is the REAL upper triangular matrix
r of the same size as A such that r' %*% r = A.  It is an error if A is
not positive definite.

                          Keyword 'nonposok'
cholesky(A, nonposok:T) does the same, except that a non positive
definite A is not considered to be an error, but results a value of NULL
being returned.  This makes it possible for a macro to take corrective
action when a matrix is not positive definite.  See topics 'macros' and
'NULL'.

                           Keyword 'pivot'
cholesky(A,pivot:T [,nonposok:T]) reorders the rows and columns as the
computation proceeds so as to obtain the most stable computation.  It
returns a structure with components 'r', a REAL upper triangular matrix,
and 'pivot', a REAL vector of integers describing the reordering.  After
result <- cholesky(A,pivot:T), result$r' %*% result$r should equal
A[result$pivot, result$pivot] except for rounding error.

                           Keyword 'force'
cholesky(A,force:Vec [,nonposok:T]), where Vec is a REAL vector whose
length is nrows(A), enables pivoting, but allows some control on
reordering.  The elements of Vec should be 1, -1, or 0, since only the
signs are used.  Before factoring, rows and columns of A, if any, with
index j such that Vec[j] > 0 are moved to rows and columns 1, 2, ...,
(initial columns) but are not further moved.  All rows and columns with
Vec[j] < 0 are moved to rows and columns nrows(A), nrows(A) - 1, ...,
(final columns) but are not further moved.  Rows and columns, if any,
with Vec[j] == 0 (pivoted columns), are free to be reordered, but will
follow the initial columns and precede the final columns.  Again the
result is a structure with components 'r' and 'pivot'.

                          Cross references
See also qr().
```

2.50 chplot()

Usage:

```
chplot(x,y [, symbols:c] [, add:T, lines:T, impulse:T] [,graphics
  keyword phrases]), where x is a REAL vector or scalar, y is a REAL
  vector or matrix and c is a integer or CHARACTER scalar, vector, or
  matrix
chplot([Graph,] [x,y, symbols:c], keys:str), str a structure whose
  component names are graphics keywords such as 'add', 'lines' and
  'impulse'
```

Keywords: plotting

```
                                  Usage
chplot(x,y,symbols:c) makes a scatter plot of REAL vector or matrix y
versus REAL vector x using plotting symbols as specified by CHARACTER or
REAL vector c.

It is not an error when x or y is NULL; a warning message is printed and
no plotting occurs.

For backward compatibility with earlier versions, you can omit keyword
'symbols', as in chplot(x,y,c).

chplot(Struc,symbols:c), where Struc is a structure with at least two
REAL components, is equivalent to chplot(Struc[1], Struc[2], symbols:c).
For example, chplot(x,y,symbols:c) and chplot(structure(x,y),symbols:c)
are equivalent.  Any components beyond the first two are ignored.

                          Graph variable argument
chplot(graph,x,y,symbols:c) or chplot(graph,Struc,symbols:c), where
graph is a GRAPH variable, draws the plot encapsulated in graph, adding
to it the new information.  See topic 'graph' for details on adding
information to a plot.

                               Symbols used
When c is REAL, each element c[i] must be an integer with 0 <= c[i] <=
999 and the plotting symbol will be the number centered at the plotting
point.

When c is a CHARACTER scalar with value "###", the characters plotted
are the same as when symbols:c is omitted; see below.

When c is CHARACTER other than the scalar "###", up to 3 characters from
each c[i] will be drawn centered at the plotting point.

                             Default symbols
Argument symbols:c may be omitted.  In this case the default plotting
characters are as follows:
  y a vector:                   The row number of an element y
```

y a matrix with ncols(y) > 1: The column number of an element

Keyword 'add'

chplot(x,y [,symbols:c], add:T, ...) does the same as chplot(LASTPLOT, x, y [,symbols:c), that is, plotted points are combined with the data already in LASTPLOT.

'lines', 'impulse' and 'dumb' keywords

chplot([graph,] x,y, lines:T [,symbols:c]) makes a character plot, connecting the points by lines similarly to lineplot().

chplot([graph,] x,y, impulse:T [,symbols:c] [,lines:T]) does the same except that vertical lines will be drawn to the points from the x = 0 line.

When option 'dumbplot' has been set False (see subtopic 'options:"dumbplot"'), the plot will be a low resolution plot unless 'dumb:F' is an argument.

Symbol variable shape

When c has more than 1 column then you must have ncols(c) = ncols(y) and the elements in c[,j] will be used to plot y[,j], reusing the rows of c cyclically if nrows(c) < nrows(y).

When c is a vector of length ncols(y), c[j] will be used to plot all elements of the column y[,j]

Otherwise, if c is a vector with length(c) != ncols(y), c[i] will be used to plot all elements in the row y[i,], reusing the rows of c cyclically if nrows(c) < nrows(y).

Drawn plotting symbols

Drawn plotting symbols

When the first character of an element of a CHARACTER c has ASCII code V between 1 and 31, it designates a specially drawn character. There are 8 basic shapes in three sizes, diamond (V=1, 9, 17), plus sign (V=2, 10, 18), square (V=3, 11, 19), cross (V=4, 12, 20), triangle (V=5, 13, 21), star (V=6, 14, 22), dot (V=7, 15, 23) and circle (V=8, 16, 24). Codes 1 - 8 are the standard sizes; codes 9 - 16 are about 2/3 standard size and 17 - 24 are about 1/2 standard size. There is only one size dot. Codes 25 - 31 "wrap around" to 1 - 7.

You can specify these special ASCII codes using quoted strings "\1", "\2", "\3", "\4", "\5", "\6", "\7", "\10", "\11", ... ,"\17", "\20", ..., "\27", "\30", ..., "\37". The digit or digits are the octal representations of the codes. For example, "\3" represents an ASCII 3 and specifies a standard size square, "\10" represents an ASCII 8 and specifies a standard size circle, and "\27" represents 22, the smallest size star. They can also be specified using escaped hexadecimal codes "\x01", "\x01", "\x02", ..., "\x09", "\x0a", ..., "\x1f".

You can also specify these codes by name, using makesymbols(). For example, chplot(x,y,symbols:makesymbols("diamond",medium:T)) makes the

```
same plot as chplot(x,y,symbols:"\11").  See makesymbols() for details.

See topic 'graphs' for the use of a scalar or length 2 vector for x.

                              Keyword 'add'
Use keyword phrase 'add:T' or commands addchars(), addlines(),
addpoints() and addstrings() to add information to a plot.

                              Keywords
Keywords 'dumb', 'lines', 'linetype', 'thickness', 'impulse', 'xmin',
'xmax', 'ymin', 'ymax', 'logx', 'logy', 'xlab', 'ylab', 'title',
'xaxis', 'yaxis', 'borders', 'ticks', 'xticks', 'yticks', 'xticklen',
'yticklen', 'xticklabs', 'yticklabs', 'height', 'width', 'pause',
'silent' and 'notes' may be used as for other plotting commands.  See
topics 'graph_keys', 'graph_border' and 'graph_ticks'

                            Cross references
See topic 'graph_assign' for information on another way to make plots.

                              Keyword 'keys'
chplot([Graph,] keys:structure(x:x,y:y,symbols:c [other keyword
phrases)) is equivalent to chplot([Graph,] x:x,y:y, symbols:c [other
keyword phrases]).  See topic 'graph_keys' for details.

                              File keywords
See topic 'graph_files' for information on how to save a plot in a file
using keywords 'file', 'new, 'ps', 'screendump', and 'epsf'.

                                Examples
Examples:
  Cmd> chplot(x,y,symbols:"*")
makes a plot of y vs x with "*" as plotting symbol.

Suppose x[,1] contains integers 1, 2, or 3.  Then

  Cmd> chplot(X2:x[,2],X3:x[,3], symbols:vector("A","B","C")[x[,1] ],\
       title:"X3 vs X2")

makes a plot of column 3 of x against column 2, using plotting symbols
"A", "B", or "C", according as the value in column 1 of x is 1, 2 or 3.
Axes are labeled 'X2' and 'X3' and a title is printed.

  Cmd> chplot(X:1,run(20)^(.2*run(5)'),\
       symbols:vector(".2",".4",".6",".8","1.")', ylab:"Powers of X",\
       title:"X^.2, X^.4, X^.6, X^.8, and X",lines:T)

draws line connected plots of x^.2, , x^.4, x^.6, x^.8 and x vs x, using
plotting symbols ".1", ".4", ...,"1.0" for each line.  X:1 is equivalent
to X:run(20).  See subtopic graphs:"specification_of_data" for details.

                            Cross references
See also topics 'graphs', plot(), lineplot(), showplot(), addchars(),
addlines(), addpoints(), colplot(), rowplot(), tek(), vt().
```

2.51 cimag()

Usage:

```
cimag(cx), cx a REAL matrix representing complex data
```

Keywords: time series, complex arithmetic

```
                                Usage
cimag(cx) computes the imaginary part of the fully complex matrix cx.
For example, cimag(matrix(run(10),5)) is vector(6,7,8,9,10).

                            Cross references
See also hconj(), cconj(), hreal(), himag(), creal().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.52 CLIPBOARD

Usage:

```
CLIPBOARD <- x   or   x <- CLIPBOARD  or  vecread(string:CLIPBOARD)
x <- fromclip([ncols]), integer ncols > 0
toclip(x), x REAL scalar, vector, matrix or array
SELECTION <- x  or  x <- SELECTION (GTK only)
Type help(CLIPBOARD) for more information.
```

Keywords: syntax, character variables, input, output

```
                      Special variable CLIPBOARD
A special CHARACTER variable CLIPBOARD is always defined.  When used in
an expression or as an argument to a function, CLIPBOARD behaves just
like any other variable.  It can be printed, written to a file, or
assigned to a regular variable.

In windowed versions, CLIPBOARD allows direct access to the system
Clipboard.

                        Assignment to CLIPBOARD
CLIPBOARD <- x assigns a CHARACTER representation of x to CLIPBOARD.  x
must be a CHARACTER, REAL or LOGICAL variable.

CLIPBOARD[1] <- x and CLIPBOARD[T] <- x do the same, provided x is a
scalar variable.  It is an error if x is not a scalar.

CLIPBOARD[rep(1,k)] <- x is permissible, in which case x must be a
```

vector of length k and CLIPBOARD will contain a CHARACTER representation of x[k].

In the windowed versions, CLIPBOARD <- x and CLIPBOARD[1] <- x copy the new value of CLIPBOARD to the system Clipboard. And when any text is copied to the system Clipboard using the Edit Menu in MacAnova or any other program, that text becomes the value of CLIPBOARD.

In non-windowed versions, CLIPBOARD <- x and CLIPBOARD[1] <- x do nothing beyond setting CLIPBOARD to CHARACTER representation of x (to x when x is CHARACTER).

Details of the CHARACTER representation

After CLIPBOARD <- x, the contents of variable CLIPBOARD (and of the system Clipboard in the Windowed versions) are as follows:

When x is a CHARACTER scalar, CLIPBOARD will contain that scalar. This is also the case after CLIPBOARD[1] <- x.

When x is a REAL or LOGICAL scalar, CLIPBOARD will contain a character representation of x. This is also the case after CLIPBOARD[1] <- x. For example CLIPBOARD is "3.1415926535897931" after either CLIPBOARD <- PI or CLIPBOARD[1] <- PI and is "T" after CLIPBOARD <- PI > 3 or CLIPBOARD[1] <- PI > 3.

When x is a vector of length N, CLIPBOARD will contain N lines separated by "\n", with line i containing a x[i], when x is CHARACTER, or a CHARACTER representation of x[i] otherwise.

When x is a matrix, CLIPBOARD will contain nrows(x) lines, with line i containing CHARACTER representations of x[i,j], j=1,...ncols(x), with the elements of each row separated by the tab character "\t".

When x is an array with 3 or more dimensions greater than 1, it is treated as if it were a matrix with nrows(x) = first dimension > 1.

Stated more technically, the value of CLIPBOARD after CLIPBOARD <- x is what would be produced by

```
Cmd> CLIPBOARD <- paste(x,multiline:T,missing:"?",sep:"\t",\
              linesep:"\n", format:"0.17g")
```

See topic paste() for more information on paste(x,multiline:T,...).

Since MISSING is coded as '?', after CLIPBOARD <- x, where x is REAL vecread(string:CLIPBOARD), should produce vector(x), including MISSING values.

Two pre-defined macros, toclip() and fromclip(), are useful when working with CLIPBOARD. In particular, toclip() allows for different coding of MISSING and user specified field separators. See topics fromclip() and toclip().

Value of CLIPBOARD

In the windowed versions, because the value of CLIPBOARD is whatever text the system Clipboard currently contains, CLIPBOARD may not be the same as what was most recently assigned to it. This can happen because of subsequent use of Edit menu items in MacAnova or another program. This feature allows easy importing of data from other programs, especially from spreadsheets. See topic fromclip().

Similarly, in the windowed versions you can easily export data from MacAnova to another application like a spreadsheet by assigning the value of a variable to CLIPBOARD. See toclip().

You can free up the memory used by the contents of CLIPBOARD by delete(CLIPBOARD). This has no effect on any system Clipboard.

You probably should not use CLIPBOARD as a name for a structure component or as a keyword on a computer with an actual Clipboard, as every mention of CLIPBOARD, even in contexts like str$CLIPBOARD or CLIPBOARD:T, refreshes special variable CLIPBOARD with stuff from the Clipboard.

Variable SELECTION in GTK

In GTK, selecting text with the mouse provides another method of communicating between programs. Briefly, if you select text and then click in a window using the middle button on the mouse, what was selected is inserted there. The GTK version of MacAnova has a special CHARACTER variable SELECTION which is connected to the current selection in the same way CLIPBOARD is connected to the Clipboard. Immediately after

```
  Cmd> SELECTION <- x
```

if you click in a window with the middle button, a character representation of x is "pasted" into the window. SELECTION[1] <- x is permissible if x is a scalar.

Similarly, if you select text in a window,

```
  Cmd> charx <- SELECTION
```

creates a CHARACTER variable charx containing the text; if the text is numerical data,

```
  Cmd> x <- vecread(string:SELECTION)
```

creates a REAL vector x. Use of this feature is somewhat tricky, since clicking in a window can change the selection. When you assign something to SELECTION, you should retrieve it with a middle button click before doing anything else. And if you want to assign from or read from SELECTION, you should type the command without a terminating Enter, then select what you want to copy, click on the frame of the MacAnova window, and then press Ctrl+E followed by Return to execute the command.

Cross references

See also topics vecread(), read(), matread(), macroread().

2.53 clipreaddata

Usage:

```
clipreaddata(name1,...,namek [,factors:T] [,keyword phrases]),
  name1,... names, quoted or unquoted variable names
clipreaddata(vector("name1",...,"namek") [,factors:F][,keyword phrases])
clipreaddata([factors:F] [,keyword phrases])
```

Keywords: input

Introduction

Macro clipreaddata() creates data vectors from information on the clipboard. It uses macro readdata() to "read" the clipboard and its usage is identical, except that you don't specify a file name. Like readdata(), it can handle data sets in which some data columns are non-numerical and can get variable names from the first line of the clipboard.

Any line on the clipboard that starts with skip character '#' is automatically skipped and is echoed to output by default.

clipreaddata() is particularly useful for importing data from a spread sheet program, particularly if the first line contains variable names. In the spread sheet program, you select a rectangular set of cells containing numbers and copy them to the Clipboard by selecting Copy on the Edit menu. After switching to MacAnova, you then use clipreaddata() to create MacAnova variables from the data in each column you selected. For instance, if the selection contains 10 rows and 5 columns of data, clipreaddata() will make 5 length 10 vectors. Caution: Before copying to the clipboard, fill any empty cells with one of "?", "*", "." or "NA". clipreaddata() will read these as MISSING values.

Usage

clipreaddata(name1,name2,...,namek) uses vecread() to read one or more columns of data from variable CLIPBOARD, creating variables. name1, name2, ..., namek. The variable names can be either quoted ("weight") or unquoted (weight).

Text on the clipboard should consist of k columns of "words" separated by spaces, commas or tabs. A column consisting entirely of numbers, possibly with MISSING values (indicated by '?', '.', '*' or 'NA') is read as a REAL vector. A word is any set of consecutive characters not including a comma, space or tab.

Non numerical data

When the first data item in a column is not a number or a code for MISSING, the entire column is normally read as a factor, with a level for each distinct word in the column. The factor has the original words in the file as row labels.

clipreaddata(name1,...,factors:F) does the same except a variable starting with a non-numerical word is read as a CHARACTER vector rather than translated into a factor.

Names from first line

clipreaddata([,factors:T]) does the same except the names for the variables are assumed to be in the first non-skipped line of the clipboard with the data starting in the second non-skipped line.

For all usages, the number of variable names, whether given as arguments or taken from the first line of the clipboard, must divide the total number of data values. And of course the names must be legal MacAnova variable names.

By default, for each variable, clipreaddata() prints a line containing the variable name and information on its type, REAL, factor or CHARACTER. You can suppress this by including 'quiet:T' as an argument.

Keywords

You can use most vecread() keywords 'quiet', 'silent', 'stop', 'skip', 'skipthru', 'go', 'quiet', 'echo', and 'n', but not 'bypass', 'bywords', 'bylines', 'bychars', 'byfields' and 'realorchar'. See topic 'vecread_keys'.

Cross references

See also fromclip(), toclip(), readcols(), vecread(), 'vecread_files'.

2.54 clipwritedat()

Usage:

```
clipwritedat(x1,x2,... [,missing:M] [, putNames:F] \
  [,fieldwidth:w or format:fmt]), vectors x1, x2, ..., all with same
  length, CHARACTER scalars M, fmt, integer w > 0
```

Keywords: output

Introduction

clipwritedat() is a macro designed to copy to the clipboard vectors of arbitrary in columnar form. By default, the columns are headed by the variable names.

Usage

clipwritedat(x1,x2,...) transforms data vectors x1, x2, ... to character form and then puts them side by side as columns in special variable CLIPBOARD. In windowed systems this also copies them to the system clipboard. Effectively, it sets CLIPBOARD to an "image" of the file that would be written by writedata(fileName, x1, x2,...).

x1, x2, ... can be of any type. If any vector is a factor and has row labels consistent with the factor levels, the row labels are written instead of the factor levels.

REAL data are formatted using the current default format as returned by getoptions(format:T), except that integer values are written as integers with no decimal point.

```
CHARACTER data is written right justified in a field whose width is
taken from the default format.

LOGICAL data are written as "T" or "F" and then written like CHARACTER
data.

MISSING values are written as "?".

If any data vector is specified by a keyword phrase (x1:X[,1], for
example), the keyword is used as the vector name.  For this usage, the
keyword may not be 'keep', 'new', 'fieldwidth' or 'missing'.

                                Keywords
The keywords for clipwritedat() are the same as those for writedata(),
except that 'new' is ignored.  See writedata() for details.

                             Cross references
See also clipreaddata(), fromclip(), toclip(), 'clipboard'.
```

2.55 cluster()

Usage:

```
cluster(x [, nclust:n, standard:F, method:name, keep:charVec, print:T,\
  tree:T or F, classes:T or F, reorder:T]), x a REAL matrix, name a
  character scalar (one of "single", "complete", "average", "ward",
  "mcquitty", "centroid", or "median"), charVec a CHARACTER vector
  with elements "all", "classes", "criterion", or "distances"
cluster(dissim:d [, ...]), d a square REAL matrix
cluster(similar:s [, ...]), s a square REAL matrix
```

Keywords: multivariate analysis

```
                                  Usage
cluster(x) performs a hierarchical cluster analysis of cases (rows) of
the data matrix x.  The default method is average linkage and the
default maximum number of clusters described in the output is 9.  It
produces a table of cluster membership with one line per case and a
dendrogram, with the join points labeled with the value of the criterion
used.  There must be at least 2 rows in x.

Distances between cases in x are computed as squared Euclidean distance
after standardization by dividing by standard deviations.
Standardization can be suppressed by including 'standard:F' as an
argument.  NOTE: This is a change in behavior of cluster() from version
3.1 to version 3.3.

                       Keywords 'dissim' and 'similar'
cluster(dissim:d) uses the upper triangle of the square matrix d as
dissimilarity or distance measure.  Matrix d must have at least 2 rows
and is treated algorithmically as if it were unsquared Euclidean
```

distance.

cluster(similar:s) uses the upper triangle of the square matrix s as a similarity matrix. Matrix sqrt(2*(max(vector(s))-s)) is used as a distance matrix. Matrix s must have at least 2 rows.

Other Keywords

Keyword phrase	Default	Meaning
		Keyword 'method'
method:Name	"average"	The clustering method used. Legal values are "ward", "single", "complete", "average", "mcquitty", "median" and "centroid". Name must be a quoted string or CHARACTER variable.
		Keyword 'nclust'
nclust:m	9	The number (>= 2) of clusters to be described in the output. When m > 25, the class membership table requires more than 80 columns for printing, and if m > 22 the dendrogram requires more than 80. m > 50 is illegal when either the class membership table or the dendrogram is to be printed.
		Keyword 'standard'
standard:F	T	suppresses the standardization of the data matrix to unit standard deviations before computing distances. Not legal with 'dissim' or 'similar'.
		Keyword 'distance'
distance:Dname	"euclid"	Specifies the distance measure used to label the dendrogram. Legal values are "euclid" and "euclidsq". It has no effect on the clustering produced. Dname must be a CHARACTER variable or quoted string. Not legal with keywords 'dissim' or 'similar'.
		Keyword 'keep'
keep:charVec	none	Specifies which, if any, results should be returned as the value of cluster(). charVec must be a quoted string or a CHARACTER vector or scalar. Legal values for elements of charVec are "distances" (the computed distances are returned), "classes" (the computed n by nclust-1 class membership matrix is returned), "crit" (the criterion values at each of the final nclust - 1 merges are saved), and "all" (all three are returned). When only one item is to be returned, it is returned as a matrix or vector. Otherwise, items are components in a structure with names 'distances', 'classes', and 'criterion'. The use of 'keep' suppresses

```
                        printing the table of class membership and the
                        dendrogram, unless print:T, tree:T, or
                        classes:T are arguments.  When 'keep' is not
                        used, cluster() has a NULL value.

                            Keyword 'print'
  print:T               Forces printing output, even when 'keep' is
                        used.  Default is F when 'keep' is used;
                        otherwise the default is T.

                            Keyword 'tree'
  tree:T                Forces printing of dendrogram.
  tree:F                Suppresses printing of dendrogram.
                        Default is F when 'keep' is used; otherwise T.
                        Must come later than 'keep' in argument list.

                           Keyword 'classes'
  classes:T             Forces printing of table of class membership.
  classes:F             Suppresses printing of table of class
                        membership.  Default is F when 'keep' is used;
                        otherwise T.  Must come later than 'keep' in
                        argument list.

                           Keyword 'reorder'
  reorder:T     F       Directs that the rows of the printed table of
                        class membership be reordered so that cases in
                        the same clusters are adjacent.  It does not
                        affect the returned value if keep:"classes"
                        appears.  The reordering is the same as that
                        implied in the dendrogram.  A warning message
                        is printed if you use reorder:T together with
                        classes:F.

                                Example
Example:
  Cmd> results <- cluster(x,nclust:15,keep:vector("classes","crit"),\
        method:"median",classes:T, reorder:T)
computes the last 15 stages of clustering, using the so called median
method, returns the class membership table and the criterion in a
structure, and prints the reordered class membership table.
```

2.56 cmplx()

Usage:

```
   cmplx(Re,Im), Re and Im REAL matrices with same size and shape.
   cmplx(Re)
```

Keywords: time series, complex arithmetic

```
                                Usage
cmplx(re,im) combines matrices re and im considered as the real and
```

imaginary parts of a complex matrix. The j-th columns of re and im become the 2j-1-th and 2j-th columns of the result. Re and im must have the same size and shape and the output has the same number of rows and twice the columns. For example, if re and im are both 5 by 2, cmplx(re,im) is equivalent to hconcat(re[,1],im[,1],re[,2],im[,2]).

cmplx(re) is equivalent to cmplx(re,0*re), that is, it produces a complex matrix with 0 imaginary part.

Cross references

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for working with complex matrices.

2.57 coefs()

Usage:

coefs([Term] [, errorTerm:ErrorTerm, se:T, coefs:F, byterm:F, silent:T]), Term a CHARACTER scalar, a positive integer, or a factor or variate in the current GLM model, ErrorTerm a CHARACTER scalar or positive integer. Use byterm:F only when Term and coefs:F omitted, and se:T included

Keywords: glm, anova, regression

Usage

coefs(Term) returns the model effects or regression coefficients for term Term in the current GLM model. These are determined from information computed by the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson().

Term is usually a quoted string or CHARACTER variable such as "a.b" which exactly matches a term in the most recent model, that is, "a.b" is not the same as "b.a". An interaction term produces a matrix or array with the leftmost subscript corresponding to the leftmost factor in Term. When a model term contains {expr} where expr is a MacAnova expression, '{' and '}' are part of the term name and must be included.

For a term which consists of a single factor or variate, Term can be its unquoted name.

Alternatively, Term can be a integer between 1 and the number of terms, excluding the final error term. For example, unless the model contained "-1", coefs(1) gets the estimated intercept or grand mean.

coefs() (no Term specified) computes coefficients for all terms in the model as a structure with one component for each term. The component names are taken from the term names, truncated if necessary to 12 characters. When any truncation is necessary, the complete term names are attached to the result as labels. See topic 'labels'.

```
                          Keyword 'silent'
coefs(Term, silent:T) and coefs(silent:T) do the same, but certain
warning and advisory messages are suppressed.  'silent:T' can be used
with any other keywords.  This feature is useful in a macro when warning
messages might confuse the user, or in a simulation.  The default value
of 'silent' is False unless the value of option' 'warnings' is False.

                            Keyword 'se'
coefs(Term,se:T) and coefs(se:T) also compute standard errors and are
equivalent to secoefs(Term) and secoefs(), respectively.  You can also
use keywords 'error' and 'byterm' in this case.  See secoefs().

Caution: After anova(), manova() and regress(), standard errors are
computed using the final error mean square in the model.  This may not
be appropriate with mixed models, including split plot designs.

                          Multivariate use
coefs(Term,Varno) or coefs(,Varno) computes coefficients only for
variable number Varno in the case of a multivariate dependent variable.
When present, Varno must be the second argument and any keywords must
follow it.

                            Limitations
coefs() does not work after screen() or after a GLM command with
'coefs:F' as an argument.

                              Example
Example: After anova("y= a + b + a.b")
   coefs(a), coefs("a"), or coefs(2) will compute the main effect
     coefficients for factor a
   coefs("a.b") or coefs(4) will produce a matrix of the a by b
     interaction coefficients.
   coefs() will produce all coefficients, including the constant.

                          Cross references
See also secoefs(), contrast(), modelinfo(), popmodel(), pushmodel()
```

2.58 colplot()

Usage:

```
colplot(x [, graphics keyword phrases]), x a REAL matrix
```

Keywords: plotting

```
                              Usage
colplot(x) makes an "interaction" plot of the data in the REAL matrix x.
The plotting positions are the row numbers and the values in x.  Points
within each column are joined by lines.  Any keywords useable in
chplot() may follow x.  colplot() is implemented as a pre-defined macro.
```

```
When option 'dumbplot' has been set False (see subtopic
'options:"dumbplot"'), the plot will be a low resolution plot unless
'dumb:F' is an argument.

See topic 'graph_keys', 'graph_border' and 'graph_ticks' for information
on other keywords that can be used with colplot().

                                Example
Example:
  Cmd> colplot(run(20)^(.2*run(5)'),xlab:"X",\
        title:"X^.2, X^.4, X^.6, X^.8, X")

                            Cross references
See also topic rowplot().
```

2.59 comments

Usage:

```
   [command1; command2 ...] # comment which will be ignored
```

Keywords: syntax

```
                                Usage
Anything following a '#' on a line is ignored unless it is part of a
string quoted with '"'.  You can use this feature to add comments to
spooled output or usage information to macros.  The '#' and everything
following up to the end of the line or a terminating '\' are skipped.

You can use macrousage() to print any comment lines (lines starting with
"#") in a macro.  It is good practice to include comment lines
describing the usage.  They should not be confused with header lines
starting with ')' in a file which may also give usage information.  See
topics macrousage(), 'macros', 'files'.

                                Example
Example:
  Cmd> xbar <- sum(x)/nrows(x) # compute mean as a row vector
is just the same as
  Cmd> xbar <- sum(x)/nrows(x)

                            Cross references
See also spool().
```

2.60 complex

Usage:

```
cmplx(re,im), hprdhj(hx1,hx2), hprdh(hx1,hx2), hdivhj(hx1,hx2),
hdivh(hx1,hx2), cprdc(cx1,cx2), cprdcj(cx1,cx2), cdivc(cx1,cx2),
cdivcj(cx1,cx2), cmatmultc(cx1,cx2), csolve(cx), ceigen(), cdiag(cx),
ctrace(), ctranspose(), hpolar(hx), cpolar(cx), hrect(hx), crect(cx),
hreal(hx), himag(hx), creal(cx), cimag(cx), csubscr()
```

Keywords: time series, complex arithmetic

Introduction

MacAnova stores complex matrices in two forms, fully complex and packed Hermitian.

Fully complex

The fully complex form has alternating columns R1, C1, R2, C2, ..., containing the real and imaginary parts of the columns of the complex matrix represented. When such a matrix has an odd number of columns, there is an implied additional column of zeros. Thus columns 1, 3, 5, ... are the real parts of complex series and columns 2, 4, 6, ... are the corresponding imaginary parts.

Packed hermitian

The packed Hermitian form is meaningful only for matrices whose columns represent periodic complex frequency functions with Hermitian symmetry sampled at frequencies 0,1/n,2/m,...,(n-1)/n cycles, where n is the number of rows. Hermitian symmetry for such a function g(f) means g(-f) = g(1-f) = conj(g(f)). This implies that g(0) and g(.5) are real and g((n-k)/n) = conj(g(k/n)), and hence only n real numbers are required to represent such series.

The n numbers are stored in the following order, the packed Hermitian form:

```
 z[0],Re(z[1]),...,Re(z[floor((n-1)/2)]),{z[n/2]},Im(floor(z[(n-1)/2])),
     ...,Im(z[1]),
```

where z[k] = g(k/n). z[n/2] is omitted when n is odd.

Functions for complex data

You can create a fully complex matrix from its real and imaginary parts Re and Im using cmplx(Re,Im). cmplx(Re) is equivalent to cmplx(Re,0*Re), returning a complex matrix with 0 imaginary part.

You can extract the real and imaginary parts and compute the complex conjugate of a fully complex matrix cx by creal(cx), cimag(cx), and cconj(cx).

You can do the same for a packed Hermitian matrix hx by hreal(hx), himag(hx), and hconj(hx).

You can switch between the two representations of a periodic Hermitian series by htoc(hx) and ctoh(cx).

You can transform both types of complex matrices to and from polar form

```
(with the modulus being stored as the real part and the argument or
phase as the imaginary part of fully complex or packed hermitian
matrices) by cpolar(cx), crect(cx), hpolar(hx), and hrect(hx).

You can multiply them element by element by cprdc(cx1,cx2), cprdcj(cx1,
cx2), hprdh(hx1,hx2), and hprdhj(hx1,hx2).

You can divide them element by element by cdivc(cx1,cx2), cdivcj(cx1,
cx2), hdivh(hx1,hx2), and hdivhj(hx1,hx2).

                        Macros for complex matrices
There are macros for working with complex matrices A and B represented
as real matrices a and b in fully complex form.  These are cmatmultc()
(A %*% B, A %c% B, A %C% B), ctrace() (trace(A)), cdiag() (diag(A)),
csolve() (inverse of A), ctranspose() (A'), cjtranspose() (conj(A)'),
ceigen() (eigenvalues and eigenvectors of a square Hermitian A) and
csubstr() (extract elements as if by subscripts).

                             Cross references
See also subtopic 'matrices:"complex_matrices"', polyroot(), rft(),
hft(), cft().
```

2.61 compnames()

Usage:

```
compnames(S), S a structure
```

Keywords: structures, character variables

```
                                   Usage
compnames(struc) returns a CHARACTER vector containing the names of the
components of STRUCTURE struc.

                                  Example
  Cmd>compnames(structure(a:1,b:2,c:17)) # returns vector("a","b","c").
  (1) "a"
  (2) "b"
  (3) "c"

                             Cross references
See also topics structure(), strconcat(), changestr(), 'structures',
nameof(), varnames()
```

2.62 console()

Usage:

```
y <- console()
```

Keywords: files, input

```
                                  Usage
y <- console() uses a pre-defined macro to read a vector using variable
CONSOLE as the filename.  This results in a request to type the data in,
terminating it with '!'.  This has the advantage over y <- vector(1.3,
4.5, 6.7, 2.5, ... ), say, in that the commas do not need to be typed.
Variable CONSOLE can be used with any operation that reads or writes a
file.  Its value is ignored.

Command vecread() is used by console() so you can use '?' to specify
missing data.

On a windowed versions, you enter data into a dialog box.  Pressing
Return or clicking on the OK button ends a line.  The Done button
terminates the data.

                               Keyword 'echo'
y <- console(echo:F) suppresses any echoing of lines read.  Such echoing
is the default behavior on windowed versions, or when using console() in
a batch file on any system.

                              Cross references
See vecread().
```

2.63 contrast()

Usage:

```
   contrast(Term,Coefs [,Byvar] [,errorTerm:ErrorTerm] [,silent:T]), Term a
     factor in the most recent GLM model or a character SCALAR or positive
     integer specifying a term and ErrorTerm a CHARACTER scalar or positive
     integer, Coefs REAL, Byvar a factor in the most recent GLM or a
     CHARACTER scalar specifying such a factor
```

Keywords: glm, anova, comparisons

```
                                  Usage
contrast(Term,Coefs) computes the estimated value, sum of squares, and
standard error for the contrast in the levels of the term specified by
Term.  The term specified must be made up exclusively of factor
variables in the model used by the most recent GLM (generalized linear
or linear model) command such as anova() or poisson().

Term is usually a CHARACTER scalar or quoted string such as "a" or
"b.c".  When the term contains just one factor, you can use just the
unquoted factor name.  Instead of a name, Term can be an integer between
1 and the number of terms in the model, counting CONSTANT if it is in
the model.  For example, after anova("a+b+a.b"), contrast(b,Coefs),
contrast("b",Coefs) and contrast(3,Coefs) are equivalent.
```

When any of the variables in the model are of the form {expr}, where expr is a MacAnova expression, you must specify the variable in Term in the same way or by number. See topic 'models'.

Result

The result is a structure with components 'estimate', 'ss', and 'se'. For example, when factor a in a model has 5 levels, contrast("a", vector(2,2,2,-3,-3)/6) compares the average effect of the first 3 levels of factor a with the average effect of the last 2.

Multi factor case

When Term contains more than one factor (for example "a.b"), Coefs must be an array with dimensions matching the number of levels in the factors of Term. The contrast coefficients must sum to zero, but MacAnova does not check to see if the contrast lies in any particular subspace.

You can use outer() to create multidimensional contrast coefficients that are products of one dimensional contrasts. For example, if factors a and b have 2 and 3 levels, respectively, after anova("y = a*b"),

Example

```
Cmd> contrast("a.b", outer(vector(1,-1), vector(2,-1,-1)))
```

compute results for a product contrast that forms part of the a.b sum of squares. See outer().

By variables

contrast(Term,Coefs,Byvar) computes the contrast, sum of squares, and standard error separately for each level of the factor variable (the "byvariable") given in the CHARACTER or quoted string variable Byvar.

For example, after anova("y=a*b"), where a has three levels, contrast("a",vector(-1,0,1),b) computes the indicated contrast for each level of b. Byvar can be specified by a quoted name but not by a positive integer.

Keyword 'errorterm'

contrast(Term, Coefs [,Byvar], errorterm:ErrTerm) does the same except the mean square error used in computing standard errors comes from the term specified by ErrTerm. ErrTerm can either be a positive integer specifying a term number, or a CHARACTER scalar or quoted string containing the name of a term in the model ("a.b.c", for example).

Keyword 'silent'

contrast(Term, Coefs, [,Byvar] [, errorterm:ErrTerm], silent:T) does the same, except that certain warning or advisory messages are suppressed. The default value of 'silent' is False unless the value of option' 'warnings' is False.

Limitations

contrast() does not work after screen() or after any GLM command with 'coefs:F' as an argument.

```
                              Unbalanced models
For unbalanced models, contrasts are computed as follows.

No byvariable:
  When Term corresponds to a term in the current anova model, the sum of
  squares is that for removal of the contrast degree of freedom from the
  complete model.  It is actually computed as a linear combination of
  the non-aliased coefficients associated with the term.

  When Term is not present in the model, the sum of squares is the
  incremental sum of squares obtained when adding the contrast df to the
  complete model, that is, the contrast is adjusted for all terms in the
  model.

Byvariable specified:
  The contrast is unadjusted for any other terms in the model, that is,
  it is computed as if the contrast degree of freedom at that level of
  the byvariable is the only degree of freedom in the model).

In all three cases, the MSE used in computing the standard error is the
error mean square (or other mean square as specified by 'errorterm')
from the most recent GLM command.

                            After nonlinear GLM
After logistic(), probit(), poisson(), and other GLMs fit iteratively by
glmfit() (but not robust()), contrast() computes the estimated value and
the deviance associated with the contrast based on full model weights.
Key word 'byvar' may not be used, but deviances are otherwise computed
as after anova().  Standard errors are computed using a scale parameter
of 1, or the value specified by keyword 'scale' on the GLM command.
Instead of 'ss', the second component of the result has name 'deviance'.

After robust(), the contrast value is computed based on coefficients
from the full model.  The "ss" computed is the square of the ratio of
the estimated contrast to its estimated standard error multiplied by the
error mean square.  Keyword 'byvar' cannot be used.

Contrast() does not work following fastanova(), ipf(), or screen() or
after a GLM command with coefs:F.

                                  Examples
Example:
  Cmd> anova("y=a+b+a.b")
  Cmd> contrast("b",vector(-1,1,0)) # assumes b has 3 levels
  Cmd> contrast("b",vector(-1,1,0),"a") # a is byvariable
  Cmd> contrast("a.b",matrix(vector(-1,1,0,0,1,-1),2))# assumes 2 by 3

                              Cross references
See also coefs(), secoefs(), modelinfo(), popmodel(), pushmodel()
```

2.64 convolve()

Usage:

```
convolve(wts, x [, reverse:T, decimate:M]), wts a REAL vector, x a REAL
  vector or matrix, M a positive integer
```

Keywords: time series

```
                                Usage
convolve(a,x) performs a circular convolution of the values in vector a
with each of the columns of vector or matrix x.  If we index rows
starting with 0, so that a contains elements a[0], a[1], ..., a[p-1],
and a column of x contains x[0], x[1], ..., x[n-1], the corresponding
column of the result is computed as follows:

  d[k] = sum(a[j]*x[k-j],j=0,min(k,p-1)) + sum(a[j]*x[k-j+n],j=k+1,p-1),

where the second sum is omitted when k >= p - 1.

                           Keyword 'reverse'
convolve(a,x,reverse:T) computes sums of circularly lagged products of
the elements of a and each column of x.  Explicitly, with the same
indexing,

 d[k] = sum(a[j]*x[j-k+n],j=0,min(k-1,p-1)) + sum(a[j]*x[j-k],j=k,p-1)

where the first sum is omitted when k = 0 and the second sum is
omitted when k >= p.

                          Keyword 'decimate'
convolve(a,x,decimate:M) and convolve(a,x,reverse:T,decimate:M) "thin"
the result by a factor of about 1/M.  M must be a positive integer.

Specifically, if d is the result of an unthinned convolution, the value
returned is a K by ncols(x) matrix where K = floor((nrows(x)-1)/M) + 1
with d[1 + (j-1)*M,] in row j.  This option is useful if a is the
impulse response function of a smoothing filter.

                           Cross references
See also autoreg(), movavg().
```

2.65 copyright

Keywords: general

```
                                Authors
MacAnova is conceived and programmed by Gary W. Oehlert and Christopher
Bingham, School of Statistics, University of Minnesota, and is Copyright
(C) 1994 - 2001 by them. Their e-mail addresses are kb@stat.umn.edu and
gary@stat.umn.edu.
```

GNU public license

MacAnova is distributed under the terms of the GNU Public License, Version 2 (see file COPYING distributed with MacAnova).

There is no warranty of any kind for MacAnova, either expressed or implied. MacAnova is distributed "as is". See file Copyint.txt for a more complete statement.

Home page

The MacAnova WWW home page is
http://www.stat.umn.edu/macanova
Executable versions of MacAnova are available there, along with source and PDF versions of the User's Guide and other documentation. An up-to-date mirror of these files is maintained by statlib at
http://lib.stat.cmu.edu/

Reports of bugs should be emailed to kb@stat.umn.edu.

Software used

The Carapace versions of MacAnova (Windows, Mac OS X, GTK Linux) make use of the wxWidgets cross-platform library available at http://www.wxwidgets.org. The wxWidgets license is very similar to the LGPL. Carapace is using wxWidgets versions 2.4.2 (Windows and GTK) and 2.5.3 (Mac OS X); we plan to move to 2.6 when is becomes available.

The extended memory MSDOS version (DJ) is compiled using a version of Gnu gcc developed and copyrighted by D. J. Delorie (DJGPP) and distributed under the terms of the GNU Public License. Source and executable for DJGPP can be found at http://www.delorie.com/djgpp

The Mac OS 9 (classic) version uses TransSkel 3.12, a transportable Macintosh application skeleton placed in the public domain by Paul Dubois (dubois@primate.wisc.edu).

Plotting is done using a modification of GNUplot, Copyright (C) 1986, 1987 Thomas Williams, Colin Kelley.

The Unix/Linux version and the extended memory DOS version (DJGPP) allow command line editing and history maintenance using the GNU Readline Library, Copyright (C) 1988, 1991 Free Software Foundation, Inc., distributed under the terms of the GNU public license. A compressed tar archive of version 2.0 (used in the Unix/Linux version) is available through the MacAnova home page. The version used in the DOS DJGPP version was included with the source for gdb4.12 found on ftp://oak.oakland.edu/ which has been reorganized since we retrieved it.

Fortran programs used

Included in MacAnova's distribution are modified translations from Fortran to C of the following programs written by others.

Program screen and related subroutines for computing regressions by leaps and bounds by G.M.Furnival and R.W.Wilson supplied by Sanford

Weisberg. See their paper, Regression by Leaps and Bounds, Technometrics 16 (1974) 499-511.

Subroutines rebak, reduc, rsg, tql2, tqlrat, tred1, tred2, svd, tridib, and tinvit from the Eispack library.

Subroutines dchdc, dgeco, dgedi, dgefa, dgesl, and dqrdc from the Linpack library.

Subroutines for computing mixed radix fast Fourier transforms written by Gordon Sande at the University of Chicago circa 1968.

Program hc and related subroutines for computing hierarchical cluster analysis by F. Murtagh, retrieved from statlib.

Subroutines for making stem and leaf displays from the book ABCs of EDA by David Hoaglin and Paul Velleman, Duxbury 1981.

Subroutines to compute the roots of real polynomials from Algorithm 493 published in TOMS retrieved from netlib.

Code to compute the cumulative normal adapted from W. J. Kennedy and J. E. Gentle, Statistical Computing, Marcel Dekker, 1980, pp 90-92, which is based on W. J. Cody, Rational Chebyshev approximations for the error function, Math. Comp 23 (1969) 631-637.

Code to compute the inverse of a normal distribution from Algorithm AS 111 by J.D. Beasley and S. G. Springer, Appl. Statist. 26 (1977), 118-121 retrieved from statlib.

Code to compute the inverse Student’s t-distribution from CACM Algorithm 396, by G. W. Hill retrieved from netlib.

Code to compute the (central) Beta distribution from a subroutine of W. Fullerton, Los Alamos, based on Bosten and Battiste, Remark on Algorithm 179, CACM 17 (1974) p. 153

Code to compute the inverse Beta distribution from Algorithm AS 109 by G. W. Cran, K. J. Martin and G. E. Thomas, Appl. Statist. 26 (1977), 111-114 retrieved from statlib.

Code to compute the non-central Beta distribution from Algorithm AS 226 by R. V. Lenth, Appl. Statist. 36 (1987) 241-244, incorporating changes by H. Frick, Appl. Statist. 39 (1990) 311-12, retrieved from statlib

Code to compute the gamma and chi-squared cumulative distributions from Algorithm AS 91 by D. J. Best and D. E. Roberts, Appl. Statist. 24 (1975), 385-388, incorporating revisions by B. L. Shea, Appl. Statist. 40 (1991), 233-235), retrieved from statlib.

Code to compute the non-central chi-squared cumulative distribution

from Algorithm AS 275 by Cherng G. Ding, Appl. Statist. 24 (1992), 478-482, retrieved from statlib.

Code to compute the non-central Student's t cumulative distribution from Algorithm AS 243 by Russell V. Lenth, Appl. Statist. 38 (1989), 185-189, retrieved from statlib.

Code to compute the cumulative distribution function and its inverse for the Studentized range from Algorithm AS 190 by R. E. Lund and J. R. Lund, Appl. Statist. 32 (1983) 204-210, incorporating corrections by Lund and Lund, Appl. Statist. 34 (1985) 104 and I. D. Hill, Appl. Statist. 36 (1987) 119, retrieved from statlib.

Code for a combined uniform pseudo-random number generator for 32 bit machines in P. L'Ecuyer 1988 Comm. ACM, retrieved from netlib.

Code implementing the Singleton quicksort algorithm (Comm. ACM Algorithm 347) adapted from ssort.f in cmlib.

Code computing the cumulative distribution for Dunnett's t was adapted from Algorithm AS 251 by C. W. Dunnett, Appl. Statist. 38 (1989) 564-579 incorporating a correction by C. W. Dunnett, Appl. Statist. 42 (1993) p. 709, and subroutine mvstud, also by Dunnett, that is part of the AS 251 distribution from statlib.

Code generating a pseudo-random Poisson variable adapted from a Fortran program in C. D. Kemp and W. A. Kemp, Poisson random variate generation, Appl. Statist. 40 (1991) 143-158.

Code generating a pseudo-random binomial variable adapted from Algorithm 678, Transactions on Math. Software 15, 394-397 by Voratas Kachitvichyanukul and Bruce Schmeiser.

Code implementing varimax rotation from subroutine varmx supplied by Douglas Hawkins (doug@stat.umn.edu).

Code implementing k-means clustering from subroutine trwcla supplied by Douglas Hawkins (doug@stat.umn.edu).

Code used to compute inverses to cumulative distributions from subroutine fsolve supplied by Douglas Hawkins (doug@stat.umn.edu). It is used by invchi() to compute the inverse of non-central chi-squared and by invdunnett() to compute probability points of Dunnett's t.

Macros from fortran

Certain macros are also based on Fortran code:

Macro levmar() in file Arima.mac is based on a Fortran program of Ken Brown. See Brown,K.,M. and Dennis,J.,E., Derivative free analogues of the Levenberg-Marquardt and Gauss algorithms for nonlinear least squares approximation. Numerische Mathematik, Vol. 18, pp. 289-297 (1972)

Macro neldermead() in file Math.mac is based on Fortran subroutine

MINIM by D. E. Shaw, CSIRO, Division of Mathematics & Statistics, with amendments by R. W. M. Wedderburn, Rothamsted Experimental Station, and Alan Miller, CSIRO, Division of Mathematics & Statistics. See also Nelder & Mead, The Computer Journal 7 (1965), 308-313. MINIM was retrieved from statlib.

Macro contourplot() and associated macros contour(), _Follow() and findcontour() are based on Fortran routines by Dan LaLiberte, implementing methods in Crane, C.M.(1972), Contour plotting algorithm, 'The Computer Journal', Vol. 15, pp. 382-384 and Cottafava, G., Andle Moli, G. (1969). Automatic Contour Map, 'Comm. ACM', Vol. 12, pp. 386-391.

2.66 cor()

Usage:

cor(x1 [,x2,...]), x1, x2, ... REAL vectors or matrices all with the same number of rows

Keywords: descriptive statistics

Usage

cor(x) computes a correlation matrix for data in REAL matrix x. Any row of x that contains missing data is entirely omitted. It is an error to have missing data in all rows.

cor(a,b,c,...), where a, b, c, ... are vectors or matrices with the same number of rows, is equivalent to cor(hconcat(a,b,c,...)) . See hconcat().

When any column in the input is constant (all values the same), the entire corresponding row and column of the output is set MISSING, including the diagonal, and a warning message is printed. In particular this occurs when the number of rows in the input is 1.

Non-MISSING elements of the diagonal are exactly 1.

2.67 cos()

Usage:

cos(x [, degrees:T or radians:T or cycles:T]), x REAL or a structure with REAL components x in radians (default), cycles, or degrees as set by option "angles" or the optional keyword

Keywords: transformations

Usage

cos(x) computes the cosine of the values of the elements of x, where x is a REAL scalar, vector, matrix or array. The result has the

same shape as x.

Units

The argument is considered to be in units of radians, degrees or cycles as determined by the value of option 'angles'. The default is radians. See topic 'options'.

cos(x, radians:T), cos(x, degrees:T), cos(x, cycles:T) interpret x as in the indicated units, regardless of the value of option 'angles'.

Missing or too large argument

When any element of x is MISSING or is too large (> 5000000*PI radians in absolute value), the corresponding element of the result is MISSING and a warning message is printed.

Structure argument

When x is a structure, all of whose non-structure components are REAL, cos(x [,UNITS:T]), where UNITS is one of 'radians', 'degrees' or 'cycles', is a structure of the same shape and with the same component names as x with each non-structure component transformed by cos().

Cross references

See topic 'transformations' for more information on cos(), including its use with a CHARACTER argument.

2.68 cosh()

Usage:

cosh(x) returns the hyperbolic cosine of the elements of x, when x is a REAL scalar, vector, matrix or array. The result has the same shape as x. In terms of other functions, cosh(x) = (exp(x) + exp(-x))/2.

When any element of x is MISSING or > 710.4758600739439 in absolute value, the corresponding element of cosh(x) is MISSING and a warning message is printed.

When x is a structure, all of whose non-structure components are REAL, cosh(x) is a structure of the same shape and with the same component names as x, with each non-structure component transformed by cosh().

See topic 'transformations' for more information on cosh().

Keywords: transformations

See topic 'transformations' for information on cosh().

2.69 cpolar()

Usage:

```
cpolar(hx [,unwind:F or crit:val]), hx a REAL matrix representing
  complex data, val a REAL scalar, 0.5 < val <= 1
```

Keywords: time series, complex arithmetic

```
                                Usage
cpolar(cx) computes the polar form of the fully complex matrix cx,
storing it in pseudo fully complex form, with the amplitude or absolute
value as the real part and the phase as imaginary part.  Thus
creal(cpolar(cx)) and cimag(cpolar(cx)) return REAL matrices whose
columns are the amplitudes and phases of the complex series represented
by pairs of columns of cx.

                             Angle units
The value of the computed phase is in radians, degrees or cycles
depending on the value of option 'angles'.  See subtopic
'options:"angles"'.  By default the phase is "unwound" so as to minimize
discontinuities arising from wrap-around.

                      Keywords 'crit' and 'unwind'
cpolar(cx,crit:Val), where .5 <= Val < 1 changes the criterion
controlling "unwinding".  The default is .75.  See unwind() for details.

cpolar(cx,unwind:F) suppresses the unwinding.

                           Cross references
See also hpolar(), crect(), hrect().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.70 cprdc()

Usage:

```
cprdc(cx1 [, cx2]), cx1 and cx2 REAL matrices representing complex data
```

Keywords: time series, complex arithmetic

```
                                Usage
cprdc(cx1, cx2) computes the element wise complex multiplication of
fully complex (pairs of columns constitute real and imaginary parts)
matrices cx1 and cx2.  When cx1 or cx2 has an odd number of columns, it
is augmented with a column of zeros before multiplication.

cprdc(cx) is equivalent to cprdc(cx,cx).
```

```
When cx1 or cx2 represents a complex scalar (nrows(cx1) = 1 and
ncols(cx1) <= 2 or nrows(cx2) = 1 and ncols(cx2) <= 2), cprdc()
multiplies every element of the other argument by that scalar.  For
example,

  Cmd> i_times_cx <- cprdc(vector(0,1)',cx)

multiplies cx by i = sqrt(-1).

When nrows(cx1) = 1 or nrows(cx2) = 1, cprdc() multiplies each row in
the other argument by that single row.

When cx1 or cx2 represents a single complex series (ncols(cx1) <= 2 or
ncols(cx2) <= 2), cprdc() multiplies each series in the other argument
by that series.

It is an error when both nrows(cx1) > 1 and nrows(cx2) > 1 and
nrows(cx1) != nrows(cx2), or when both ncols(cx1) > 2 and ncols(cx2) > 2
and ceiling(ncols(cx1)/2) != ceiling(ncols(cx2)/2) (the number of
complex series represented differs).

                              Examples
Examples:
  When ncols(cx1) = 2 and ncols(cx2) = 5, cprdc(cx1,cx2) is equivalent to
  cprdc(hconcat(cx1,cx1,cx1),hconcat(cx2,rep(0,nrows(cx2))))

  When a and b are REAL scalars, cprdc(cmplx(a,b),cx) computes the
  complex scalar product of a + b*i and cx.

                           Cross references
See also cprdcj(), hprdh(), hprdhj(), cdivc(), cdivcj(), hdivh(),
hdivhj(), hconcat(), rep(), nrows(), ncols().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.71 cprdcj()

Usage:

```
cprdcj(cx1 [, cx2]), cx1 and cx2 REAL matrices representing complex data
```

Keywords: time series, complex arithmetic

```
                                  Usage
cprdcj(cx1, cx2) computes the element wise complex multiplication of
fully complex (pairs of columns constitute real and imaginary parts)
matrices cx1 and cconj(cx2).  When cx1 or cx2 has an odd number of
```

```
columns, it is augmented with a column of zeros before multiplication.

cprdcj(cx) is equivalent to cprdcj(cx,cx) and returns as output a matrix
with the squared moduli of the elements of cx, considered as complex
numbers, in the odd columns (real part) of the result, with zero's in
the even columns (imaginary part).

When cx1 or cx2 represents a complex scalar (nrows(cx1) = 1 and
ncols(cx1) <= 2 or nrows(cx2) = 1 and ncols(cx2) <= 2), cprdcj()
multiplies every element of the other argument by that scalar.  For
example,

  Cmd> i_times_cx <- cprdcj(vector(0,1)',cx)

multiplies cconj(cx) by i = sqrt(-1).

When nrows(cx1) = 1 or nrows(cx2) = 1, cprdcj() multiplies each row in
the other argument by that single row.

When cx1 or cx2 represents a single complex series (ncols(cx1) <= 2 or
ncols(cx2) <= 2), cprdcj() multiplies each series in the other argument
by that series.

It is an error when both nrows(cx1) > 1 and nrows(cx2) > 1 and
nrows(cx1) != nrows(cx2), or when both ncols(cx1) > 2 and ncols(cx2) > 2
and ceiling(ncols(cx1)/2) != ceiling(ncols(cx2)/2) (the number of
complex series represented differs).

                              Examples
Examples:
  When ncols(cx1) = 2 and ncols(cx2) = 5, cprdcj(cx1,cx2) is equivalent
  to cprdcj(hconcat(cx1,cx1,cx1),hconcat(cx2,rep(0,nrows(cx2))))

  When a and b are scalars, cprdcj(cmplx(a,b),cx) computes the complex
  scalar product of a + b*i and cconj(cx).

                           Cross references
See also cprdc(), hprdh(), hprdhj(), cdivc(), cdivcj(), hdivh(),
hdivhj(), cconj(), hconcat(), rep(), nrows(), ncols().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.72 creal()

Usage:

```
creal(cx), cx a REAL matrix representing complex data
```

Keywords: complex arithmetic, time series

```
                                Usage
creal(cx) computes the real part of the fully complex matrix cx.  For
example, creal(matrix(run(10),5)) is vector(1,2,3,4,5).

When cx has an odd number, say 2*m-1, of columns, the last column is
interpreted as the real part of a complex series with zero imaginary
part and the result has m columns.

                            Cross references
See also hconj(), cconj(), hreal(), himag(), cimag().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.73 crect()

Usage:

```
crect(cx), cx a REAL matrix representing complex data
```

Keywords: time series, complex arithmetic

```
                                Usage
crect(cx) is the inverse operation to cpolar().  Matrix cx is assumed to
represent the polar form of a fully complex series, with amplitudes or
absolute values in the real part and phases in the imaginary part.  The
result contains the real and imaginary parts of that series in fully
complex form.  See topic 'complex' for discussion of complex matrices in
MacAnova.

                              Angle units
The phases are assumed to be in units of radians, degrees or cycles
depending on the value of option 'angles'.  See subtopic
'options:"angles"'.

                            Cross references
See also cpolar(), hpolar(), hrect().
```

2.74 ctoh()

Usage:

```
ctoh(cx), cx a REAL matrix representing complex data
```

Keywords: time series, complex arithmetic

Usage

ctoh(cx) returns the packed Hermitian symmetrized form of the REAL matrix cx, considering its columns in pairs as representing unrestricted Complex series. When cx is m by 2*n-1, or m by 2*n, ctoh(cx) is m by n, with column i containing the Hermitian symmetrized form of columns 2*i-1 and 2*i. When ncols(cx) is odd, the final complex series is assumed to have imaginary part zero.

When cx actually has Hermitian symmetry, ctoh(cx) represents the same matrix in packed form. When cx does not have such symmetry, ctoh(cx) represents the symmetrized matrix obtained by averaging elements that should be equal and discarding the imaginary parts of elements that should be real.

Cross references

See topic 'complex' for discussion of complex matrices in MacAnova.

See also htoc(), cconj(), hconj(), hreal(), himag(), creal(), cimag().

See subtopic 'matrices:"complex_matrices" for a list of macros for working with complex matrices.

2.75 cumbeta()

Usage:

cumbeta (x,alpha,beta[,lam] [,upper:T or lower:F]), x, alpha, beta, and lam REAL, elements of alpha, beta > 0, lam >= 0

Keywords: probabilities

Usage

cumbeta(Val,a,b) computes the probabilities that a beta random variable with parameters a and b would be <= the elements of the vector, matrix, or array Val.

cumbeta(Val,a,b,lam) computes similar probabilities for non-central beta with noncentrality parameter lam.

Any of Val, a, b, or lam that are not scalars (single numbers) must be vectors, matrices, or arrays with the same size and shape which will also be the size and shape of the result.

cumbeta(Val,a,b [,lam] ,upper:T) and cumbeta(Val,a,b [,lam] ,lower:F) do the same, except P(beta >= Val) is computed.

The elements of a, b and lam must be positive REAL numbers.

Example

Example:

```
Cmd> cumbeta(.173,1.5,2.5,upper:T) # upper tail prob,a = 1.5,b = 2.5
(1)      0.79252
```

```
  Cmd> 1 - cumbeta(.173,1.5,2.5) # same
  (1)      0.79252

                            Cross references
See also cumF(), invF(), invbeta().
```

2.76 cumbin()

Usage:

```
   cumbin(x,N,P [,upper:T or lower:F]), x, N and P REAL, elements of N
     integers > 0, elements of P between 0 and 1
```

Keywords: probabilities

```
                                    Usage
cumbin(Val,N,P) computes the probabilities that a binomial random
variable with N trials at probability P would be <= elements of the
vector, matrix or array Val.  When Val is not integral, the result is
the same as cumbin(floor(Val),N,P).

Any of Val, N, and P that are not scalars (single numbers) must be
vectors, matrices, or arrays with the same size and shape which will
also be the size and shape of the result.

The elements of N must be positive integers less than 100,000 and the
elements of P must be between zero and one.

cumbin(Val,N,P,upper:T) and cumbin(Val,N,P,lower:F) compute the
probability that the binomial random variable is >= elements of Val.
This is mathematically the same as 1 - cumbin(ceiling(Val - 1),N,P), not
1 - cumbin(Val,N,P)

Note that when Val is an integer, P(x = Val) is included in both
cumbin(Val,N,P) and cumbin(Val,N,P,upper:T).

                             Computing P values
If x_obs is an observed number of successes in a sequence of n
independent trials with constant p = P(success), you can use cumbin() to
compute P-values for a test of H_0: p = p_0 as follows:
    H_a      P-value
  p > p_0    cumbin(x_obs,n,p_0,upper:T)
  p < p_0    cumbin(x_obs,n,p_0)
  p != p_0   2*min(cumbin(x_obs,n,p_0),cumbin(x_obs,n,p_0,upper:T))

                                  Example
Example:
  Cmd> 1 - cumbin(7,13,.25) # P(x > 7) with n = 13, p = .25
  (1)   0.0056493

  Cmd> cumbin(8,13,.25,upper:T) # or cumbin(8,13,.25,lower:F), same
```

```
  (1)   0.0056493

                            Cross references
See also cumpoi().  See invbeta:"binomical_confidence_interval" for
information on computing a confidence interval for p based on a binomial
random variable
```

2.77 cumchi()

Usage:

```
   cumchi(x,df [,upper:T or lower:F]), x and df REAL, elements of df > 0
   cumchi(x,df,lam [,upper:T or lower:F]), same x, df, lam REAL with 0 <=
     lam[i] < 1419.56542578676
```

Keywords: probabilities

```
                                    Usage
cumchi(Val,df) computes P(x <= Val) where x is a chi-square random
variable with df degrees of freedom.  When Val is a vector, matrix or
array, the probabilities are computed for each element.  When Val and df
are both not scalars, they must be the same size and shape which will
also be the size and shape of the result.

The elements of df must be positive (fractional degrees of freedom are
allowed).

cumchi(Val, df, upper:T) and cumchi(Val, df, lower:F) do the same except
the upper tail probability P(x >= Val) is computed.  It is
mathematically the same as 1 - cumchi(Val, df), although a more accurate
value may be computed.

                                    Example
  Cmd> 1 - cumchi(sum((obs-e)^2/e), nrows(obs) - 1) # P value

  Cmd> cumchi(sum((obs-e)^2/e), nrows(obs) - 1, upper:T) # same

                             Non central chi squared
cumchi(Val,df,lam) computes P(x <= Val) where x is a non-central
chi-square random variable with df degrees of freedom and non-centrality
parameter lam.  Both cumchi(Val,df,lam,upper:T) and cumchi(Val,df,lam,
lower:F) compute P(X >= Val).  All non-scalar arguments must be the same
size and shape which will also be the size and shape of the result.  The
elements of Val must be non-negative and less than 1419.56542578676.

                                    Example
Power of 5% Chi-squared test of test of H_0: p[1] = p0[1], ..., p[k] =
p0[k] when p[j] = p1[j], j = 1,...,k based on a multinomial sample of
size n:
  Cmd> cumchi(invchi(.05,k-1,upper:T),k-1,n*sum((p1-p0)^2/p0),upper:T)

                            Cross references
```

```
See also cumgamma(), invchi(), invgamma().
```

2.78 cumdunnett()

Usage:

```
cumdunnett(x, ngroup, errorDf [,groupSizes][,onesided:T][,epsilon:eps]
  [,upper:T or lower:F]) x REAL, elements of ngroup integers >= 2,
  elements of errorDf >= 1, elements of groupsizes >= 0, eps > 0,
  default = .00001.
```

Keywords: probabilities, comparisons

```
                                  Usage
cumdunnett(x, K, Df) computes the probability that Tmax <= x, Tmax =
max(abs(t21), abs(t31), ..., abs(tK1)), where t21, t31, ..., tK1 are K-1
t-statistics of the form tI1 = (xbarI-xbar1)/ stderr(xbarI-xbar1), I =
2,...,K.  xbar1, xbar2, ..., xbarK are the means of independent normal
random samples of the same size with identical population means and
variances, and the standard errors are computed using an independent
estimate of error variance with Df degrees of freedom.  The value is 0
for any x <= 0.  For x >= 0, when K = 2 the value is the same as
2*cumstu(x,Df) - 1.  See cumstu().

cumdunnett(x, K, Df, upper:T) and cumdunnett(x, K, Df, lower:F) do the
same, except they compute the upper tail probability P(Tmax >= x).

                            Keyword 'onesided'
cumdunnett(x, K, Df, onesided:T) computes the probability that Tmax<=x,
where Tmax is now the maximum of t21, t31, ..., tK1, not of their
absolute values.  When K = 2 the value is the same as cumstu(x,Df).
With 'upper:T' or 'lower:F' it computes the upper tail probability
P(Tmax >= x).

See below for computing probabilities when the sample sizes differ.

x, K and Df must be REAL.  The elements of K must be integers >= 2, and
the elements of Df must be >= 1, not necessarily integers.

Any of the arguments x, K or Df that are not scalars must all be
vectors, matrices or arrays of the same size and shape; the value has
the same size and shape.

                             Keyword 'epsilon'
cumdunnett(x, K, Df [, onesided:T] [, upper:T or lower:F], epsilon:eps),
where eps is a small positive number (default .00001) which controls the
accuracy to which the probability is computed; the computed probability
should be no farther than eps from the true probability.

                           Multiple comparisons
cumdunnett() is primarily used to compute P values for a multiple
comparisons procedure due to C. W. Dunnett wherein a control group
```

(group 1) is compared to K-1 other treatment groups using K-1 t-tests.

For a completely randomized design with k treatment groups of size n, the P value is computed as cumdunnett(maxt, k, k*n - k, upper:T [,onesided:T]). See invdunnett() for computing critical values of the Dunnett test.

Caution: cumdunnett() is very computation intensive and may be unacceptably slow on an older computer. On one Macintosh 68000 computer with no math coprocessor, a single value took about 7 minutes to compute.

Example

```
Cmd> cumdunnett(3.27, 5, 5*8 - 5, upper:T) # P(Tmax >= 3.27)
(1)     0.00866
```

computes P(Tmax > 3.27) for a completely randomized design with 5 groups, all with sample size 8.

Differing sample sizes

cumdunnett(x, K, Df, groupSizes [,onesided:T, epsilon:eps, upper:T or lower:F]) computes probabilities for Tmax, with REAL argument groupSizes specifying the sample sizes.

In the simplest usage, groupSizes is a vector (ndims(groupSizes) = 1), with elements >= 0. When groupSizes is a matrix or array (ndims(groupSizes) > 1), it is treated as if it were a vector, matrix or array, with one less dimension, each of whose elements is a vector with length = last dimension of groupSizes.

The first ndims(groupSizes) - 1 dimensions of groupSizes must match the dimensions of any of x, K, or DF which is not a scalar. In particular, a m by 1 matrix, which is treated as a vector of length m by most MacAnova functions, is interpreted by cumdunnett() as a set of m vectors of length 1.

In computing an element of the result based on a vector of group sizes (either all of groupSizes when it is a vector, or a row or "slice" of groupSizes when ndims(groupSizes) > 1), cumdunnett() uses up to k of the non-zero leading values in the vector, where k is the corresponding element of K. When there are fewer than k non-zero values, the last one is replicated as many times as needed. It is an error to have a zero value followed by a nonzero value or to have all values zero.

When there is only 1 non-zero value in a row or "slice" of groupSize, the replication of this element means the group sizes are assumed to be equal. In particular, this is the interpretation when groupSizes is a scalar or a m by 1 matrix.

Examples

```
Cmd> cumdunnett(3.27, 4, 12 - 4, vector(6,2,2,2), upper:T)
(1)     0.03070
```

```
computes P(Tmax >= 3.27) for a completely randomized design with 4
groups and sample sizes 6, 2, 2 and 2.

  Cmd> cumdunnett(3.27, vector(3,4), vector(12 - 3, 12 - 4),\
       matrix(vector(6,3,3,0, 6,2,2,2),4)', upper:T)
  (1)     0.01825     0.03070

computes P(Tmax >= 3.27) for two completely randomized designs, one with
3 groups and sample sizes 6, 3, and 3, the other with 4 groups with
sample sizes 6, 2, 2, and 2.  Because trailing values in the rows of
groupSizes are replicated, matrix(vector(6,3, 6,2),2)' would be an
equivalent way to specify the group sizes.

  Cmd> cumdunnett(3.27, 4, 12 - 4, 3)
  (1)     0.97182

computes the same result as cumdunnett(3.27, 4, 12 - 4), because
groupSizes is a scalar.

                      Ratios of samples sizes used
Only the ratios of non-zero elements of groupSizes are relevant.

For example, for 5 groups (K=5), the following groupSizes are
equivalent: vector(1,2,3,3,3), vector(1,2,3), vector(1,2,3,0,0),
vector(2,4,6).

vector(1,2,3,0,3) and vector(1,2,3,0,0,1) would be errors because a
non-zero value follows a zero.

Caution: cumdunnett() is somewhat computation intensive.  On a slow
computer you may have to wait several seconds for a result.  Using a
somewhat larger value for epsilon, for example, epsilon:.0001, may speed
up the calculation at the cost of loss of accuracy.

                            Cross references
See also invdunnett(), cumstudrng().
```

2.79 cumF()

Usage:

```
   cumF(x,df1,df2 [,lam] [,upper:T or lower:F]), x, df1, df2 and lam REAL,
     elements of df1 and df2 >i 0 and lam >= 0
```

Keywords: probabilities

```
                                  Usage
cumF(Val,df1,df2) computes the probabilities that an F random variable
with df1 and df2 degrees of freedom would be less than the elements of
the vector, matrix, or array Val.

cumF(Val,df1,df2,upper:T) and cumF(Val,df1,df2,lower:F) compute upper
```

```
tail probabilities.  For large Val, the result may preserve significant
digits that are lost when computing the upper tail probability by 1 -
cumF(Val,df1,df2).

                              Non central F
cumF(Val,df1,df2,lam [,upper:T or lower:F]) computes similar
probabilities for non-central F with noncentrality parameter lam.

Any of Val, df1, df2, or lam that are not scalars (single numbers) must
be vectors, matrices, or arrays with the same size and shape which will
also be the size and shape of the result.

The degrees of freedom must be positive REAL numbers (not necessarily
integers).  Upper tail areas of F can be computed as 1 - cumF().

                                 Example
Examples:
Compute P-value for F-statistic following anova():
  Cmd> cumF((SS[2]/DF[2])/(SS[5]/DF[5]), DF[2], DF[5], upper:T)

Compute 2-tail P-value for test of sigma_1 = sigma_2 based on sample
standard deviations s1 and s2 from independent normal samples.

  Cmd> 2*min(cumF(s1^2/s2^2, n1-1, n2-1), \
           cumF(s1^2/s2^2, n1-1, n2-1, upper:T)) # two tail P-value

                            Cross references
See also invF(), cumbeta(), invbeta().
```

2.80 cumgamma()

Usage:

```
   cumgamma(x,alpha [,upper:T or lower:F]), x and alpha REAL, elements of
     alpha > 0
```

Keywords: probabilities

```
                                  Usage
cumgamma(Val,alpha) computes the probabilities that a gamma random
variable with shape parameter alpha would be less than the elements of
the vector, matrix, or array Val.

When Val and alpha are both not scalars, they must be vectors, matrices,
or arrays with the same size and shape which will also be the size and
shape of the result.

The elements of alpha must be positive.

cumgamma(Val,alpha,upper:T) and cumgamma(Val,alpha,lower:F) compute
upper tail probabilities.  For large Val, this may provide more
significant digits than the mathematically equivalent 1 -
```

```
cumgamma(Val,alpha).

cumchi(x,df [,upper:T or lower:F]) is equivalent to cumgamma(x/2,df/2
[,upper:T or lower:F]).

                                Example
Example:
  Cmd> 1 - cumgamma(13.27, 15.5) # P(x >= 13.27)
  (1)      0.69507

  Cmd> cumgamma(13.27, 15.5, upper:T) # the same
  (1)      0.69507

                            Cross references
See also invgamma(), cumchi(), invchi().
```

2.81 cumnor()

Usage:

```
   cumnor(x [,upper:T or lower:F]), x REAL
```

Keywords: probabilities

```
                                  Usage
cumnor(Z) computes the probabilities that a standard normal (mean 0,
variance 1) random variable would be less than the elements of the
vector, matrix, or array Z.  The size and shape of the result is the
same as that of Z.

cumnor(Z, upper:T) and cumnor(Z, lower:F) compute upper tail
probabilities.  For large positive Z, these may preserve significant
digits lost by the mathematically equivalent 1 - cumnor(Z).

Two-tailed P values associated with an observed value of z, may be
computed as 2*cumnor(abs(z),upper:T) or 2*(1 - cumnor(abs(z))).

                                Example
Example:
Two-tail normal P-value for z_obs = 1.841
  Cmd> 2*cumnor(1.841,upper:T) # or 2*(1 - cumnor(1.841))
  (1)     0.06562

                            Cross references
See also invnor().
```

2.82 cumpoi()

Usage:

```
cumpoi(x,mu [,upper:T or lower:F]), x and mu REAL, elements of mu > 0
```

Keywords: probabilities

```
                                  Usage
cumpoi(Val,mu) computes the probability that a Poisson random variable
with mean mu is <= the elements of the vector, matrix or array Val.
When Val is not integral, the result is the same as cumpoi(floor(Val),
mu).

When Val and mu are both not scalars, they must be the same size and
shape which will also be the size and shape of the result.

All elements of mu must be non-negative

cumpoi(Val,mu,upper:T) and cumpoi(Val,mu,lower:F) compute the
probability that the Poisson random variable is >= elements of Val.
This is mathematically the same as 1-cumpoi(ceiling(Val-1),mu), not
1-cumpoi(Val,mu).

Note that when Val is an integer, P(x = Val) is included in both
cumpoi(Val,mu) and cumpoi(Val,mu,upper:T).

                           Computing P values
If x_obs is an observed value of a Poisson random variable with mean mu,
you can use cumpoi() to compute P-values for a test of H_0: mu = mu_0 as
follows:
     H_a       P-value
  mu > mu_0    cumpoi(x_obs,mu_0,upper:T)
  mu < mu_0    cumpoi(x_obs,mu_0)
  mu != mu_0   2*min(cumpoi(x_obs,mu_0),cumpoi(x_obs,mu_0,upper:T))

                                  Example
Example:
  Cmd> 1 - cumpoi(13,9.5) # P(x > 13) = 1 - P(x <= 13), mu = 9.5
  (1)      0.10186

  Cmd> cumpoi(14,9.5, upper:T) # same
  (1)      0.10186

See also cumbin().  See invgamma:"poisson_confidence_interval" for
information on computing a confidence interval for a Poisson mean.
```

2.83 cumstu()

Usage:

```
cumstu(x,df [,upper:T or lower:F]), x and df REAL, elements of df > 0
cumstu(x,df,delta [,upper:T or lower:F]), x, df > 0, delta REAL
```

Keywords: probabilities

```
                                  Usage
cumstu(Val,df) computes P(t <= Val) where t is a Student's t random
variable with df degrees of freedom.  Val and df can be scalars,
vectors, matrices or arrays, but must have the same size and shape if
neither is a scalar.  When both are scalars, the result is a scalar; if
there is a non-scalar argument, the result has the same size and shape
as that argument.

The degrees of freedom must be positive, but not necessarily integral.

cumstu(Val,df,upper:T) and cumstu(Val,df,lower:F) compute upper tail
probabilities P(t >= Val).  The result is mathematically equivalent to 1
- cumstu(Val,df) but may be more accurate for large Val.

Two tailed P values for an observed t statistic Val can be computed with
the macro twotailt(Val,df) or as 2*cumstu(abs(Val),df,upper:T)).

                                  Example
Compute two-tail P-value of H_0: mu = 10 using sample mean xbar and
standard deviation s from sample of size n:
  Cmd> 2*cumstu(abs(sqrt(n)*(xbar-10)/s), n-1, upper:T) #2-tail P-value

                               Non central t
cumstu(Val,df,delta) computes P(t <= Val) where t is a non-central
Student's t random variable with df degrees of freedom and noncentrality
parameter delta.  All three arguments can be scalars, vectors, matrices
or arrays, but any non-scalar arguments must have the same size and
shape which will be the size and shape of the result.

cumstu(Val,df,delta,upper:T) computes the upper tail probability
P(t >= Val).

When Val = (xbar - mu_0)/(s/sqrt(n)) is computed from a random sample of
size n from N(mu_a, sigma^2), delta = sqrt(n)*(mu_a - mu_0)/sigma.

                                  Example
Compute the power of a one-tail 5% t-test of H_0: mu = 10 vs H_a: mu >
10 when mu = 15:
  Cmd> cumstu(invstu(.05,n-1,upper:T),n-1,sqrt(n)*(15-10)/sigma,upper:T)

                              Cross references
See also twotailt(), invstu(), subtopic cumF:"non_central_F", power()
and power2().
```

2.84 cumstudrng()

Usage:

```
cumstudrng(x, ngroup, errorDf [,epsilon:eps] [,upper:T or lower:F])
  where x is REAL, elements of ngroup integers >= 2, elements of errorDf
  >= 1, eps > 0 small
```

Keywords: probabilities, comparisons

Usage

cumstudrng(x, K, Df) computes the probability that Q <= x, where Q is a Studentized range based on K normal variates and an independent estimate of variance with Df degrees of freedom. All three arguments must be REAL. K must consist of integers >= 2, and the elements of Df must be >= 1, not necessarily integers. The value is 0 for any x <= 0. For any element of Df > 1000, the asymptotic value (Df = infinity) is used.

Any of the arguments x, K or Df that are not scalars must be vectors, matrices or arrays all of the same size and shape.

cumstudrng(x,2,Df) should be the same as 2*cumstu(x/sqrt(2),Df) - 1 except for computational error.

cumstudrng(x, K, Df, upper:T) and cumstudrng(x, K, Df, lower:F) compute the upper tail probability P(Q >= x). The result is equivalent to 1 - cumstudrng(x,K,Df).

Test of ANOVA hypothesis

When you have K independent normal samples of size n, all with the same variance, you can test the null hypothesis that all means are equal by the studentized range statistic computed as Q <- (max(xbars) - min(xbars))/sqrt(Ssq/n). This is an alternative to the ANOVA F-statistic.

You can compute the P-value based on Q as cumstudrng(Q,K,K*(n-1), upper:T). Here xbars is a vector containing the K sample means and Ssq is the pooled estimate of variance. See invstudrng() for computing critical values for Q.

Keyword 'epsilon'

cumstudrng(x, K, Df, epsilon:eps [,upper:T or lower:F]), where eps is a small positive scalar, does the same computation with accuracy influenced by eps. The smaller the value of eps, the more accurate the result should be, but the longer it will take to compute it. The default value of eps is 0.0000001.

Cross references

See also invstudrng(), cumstu().

2.85 customize

Keywords: control, general

```
                                Introduction
You can alter how MacAnova works either by using command line options
when launching MacAnova or by setting options once MacAnova has started.
You can automate these approaches by setting an environmental variable
MACANOVA and/or by preparing a startup file MacAnova.ini.txt.

One common reason doing this is to add additional search paths for
macros, help, and files.  Also, on Unix/Linux, it may be necessary to
set the directory where MacAnova lives by using the -appdir option.

                        Environmental variable MACANOVA
MacAnova looks for an environmental variable MACANOVA.  Its value should
be a list of command line options such as '-l 26 -w 75 -q' (see topic
'launching' for details on command line options).  These are, in effect,
prepended to any command line options that you use and thus are can be
overridden by options on the command line.  Some care may be needed
when quoting arguments with embedded spaces.  Also, it may be necessary
to double any backslashes used in directory paths.

DOS/Windows example: add another search directory and set dumb plots to
be 26 lines and 75 columns.  On Windows NT/2000/XP, right click on
"My Computer", then select "Properties", then select "Environment" (you
may need to select "Advanced" on some systems).  Then add a new variable
MACANOVA with value -l 26 -w 75 -path c:\macanova\macros
On earlier systems you can add a line like the following in your
AUTOEXEC.BAT file.
  SET MACANOVA=-l 26 -w 75 -path c:\macanova\macros

Unix/Linux example: set the application directory and add a directory to
the search path.  The method for setting environment variables depends
on the shell that you use.  For csh or a variant such as tcsh, add a line
similar to the following to the .chsrc file in your home directory:
  setenv MACANOVA '-path ~/mymacros -appdir /usr/local/macanova'
For sh, bash, or ksh, add a line similar to the following to the .profile
file in your home directory:
  MACANOVA='-path ~/mymacros -appdir /usr/local/macanova';export MACANOVA

Another use for MACANOVA is to make sure some standard macros are read
in or variables created by including '-e Command' in the environmental
variable (see 'launching').  For example, in Unix/Linux, one of
  setenv MACANOVA "-e getmacros(ffplot,tsplot,spectrum,quiet:T)" [csh]
or
  MACANOVA="-e getmacros(tsplot,ffplot,quiet:T)";export MACANOVA [sh]
ensures macros tsplot() and ffplot() will always be available.  Warning:
The command cannot contain any spaces or tabs.

                    Customizing by using a startup file
When MacAnova is launched, it searches for a "startup" file with a
special name: MacAnova.ini.txt  MacAnova searches in the default
directory and any directories listed in character vector DATAPATHS.
```

If it is found, MacAnova assumes the initialization file contains MacAnova commands and executes it silently as a batch file before the first prompt (see batch(), launching).

See topic 'launching' for information on how to specify an alternative startup file using command line flag -f. See topic 'DATAPATHS' for a description of setting the search path.

The use of a startup file is completely optional. If you have one, you can put commands in it to set options such as the default output formatting, file names to replace the default values of variables DATAFILE, MACROFILES, DATAPATHS (see topic 'DATAPATHS'), and HOME (see topic 'file_names') and the units (radians, degrees, or cycles) to be used by trigonometric functions (see topic 'options'). You can also include commands to create macros or read in macros that will thus always be available whenever you launch MacAnova.

Note: setoptions(prompt:newprompt) has no effect in a startup file. See topics setoptions(), 'options'.

The version of the startup file distributed with MacAnova does nothing as it stands, since every action in it is in an if(F){...} clause. You can activate actions in the file by editing it to change some or all of the if(F){...} to if(T){...} using any text editor. If you use a word processor, the file must be saved as a text or ASCII file.

Tektronix emulation

If you run a Unix/Linux version of MacAnova through a terminal emulating program that can switch into and out of Tektronix 4014 emulation mode, you may want to use a startup file that sets option 'tektest' to specify the character strings that control such switches. See subtopic 'options:"tektest"'. This is not necessary when you are running in an xterm window.

Example startup file

Here is a simple example of a possible MacAnova startup file (the line numbers are for reference but are not part of the file)

```
Line #
  1    setoptions(nsig:6,angles:"cycles",pvals:T,fstats:T,restoredel:F)
  2    DATAFILE <- "timeser.dat"
  3    addmacrofile("mytser.mac")
  4    adddatapath("C:\\Time Series\\Data") #Windows form
  5    ls <- macro("listbrief($0)")
  6    if(isdefined(DEGPERRAD)){delete(DEGPERRAD,lockedok:T)}
```

If this were your startup file, it would have the following effects:

Line 1:

Output will be printed with 6 significant digits (option 'nsig')

Trigonometric functions will assume that angles are measured in cycles with 1 equivalent to 2*pi (option 'angles')

Output from GLM functions such as anova(), regress(), and glmfit() will include P values, and where appropriate, F-statistics (options

```
    'pvals' and 'fstats')
  Command restore() will not delete existing variables unless they are
    overwritten or unless keyword phrase 'delete:T' is used on restore()
    (option 'restoredel')
Line 2:
  Pre-defined CHARACTER variable DATAFILE will be redefined to be
    "timeser.dat".  This will result in getdata() retrieving data
    from file "timeser.dat".
Line 3:
  Prepend "mytser.mac" to pre-defined CHARACTER vector MACROFILES,
    ensuring that getmacros() will search file mytser.mac before the
    standard macro files.
Line 4:
  Prepend "C:\\Time Series\\Data" to predefined CHARACTER variable
    DATAPATHS making it the first search directory.  On Unix/Linux
    might be something like "~/TimeSeries/Data", and on Mac OS 9
    it might be "MyDisk:Time Series:Data".  See topic 'DATAPATHS'.
Line 5:
  Macro ls will be an "alias" for command listbrief()
Line 6:
  Pre-defined REAL constant DEGPERRAD with value 180/pi will be deleted.
  'lockedok:T' is required since DEGPERRAD is a locked variable.
```

2.86 data_files

Keywords: variables, files, input, output

```
                              Introduction
Any data file that can be read by MacAnova must be a plain text or ascii
file.  If you create it in a word processor, be sure to save it as a
text or ascii file.

All versions of MacAnova correctly read text files in DOS and Windows
format (lines separated by carriage return and linefeed code), in
Macintosh format (lines separated only by carriage return code), and in
Unix/Linux format (lines separated only by linefeed code).

                           Option 'maxlinelen;
The only known limitation is that no more than a fixed number of
characters will be scanned in any single line.  This number is
determined by option 'maxlinelen' whose default in most versions is
2000.  If by some chance, you need to read a file containing lines
longer than this, you can increase it by, say

  Cmd> setoptions(maxlinelen:5000)

                            Cross references
See topic 'vecread_file' for information on files containing a single
unstructured REAL or CHARACTER data set that can be read by vecread()
and readcols().
```

See topic 'matread_file' for information on files of named REAL, LOGICAL, or CHARACTER data sets or structures that can be read by matread() and read(). Data sets may contain coordinate labels and have descriptive notes. No single line can have more than 50 data items.

See topic 'macro_files' for information on files that can be read by macroread() and read().

See topic 'files' for technical information on file names, default directories or folders, and abbreviated file names of the form "~/filename".

See topic 'DATAPATHS' for information on where MacAnova searches for files.

See also matread(), read(), vecread(), readcols(), and macroread().

2.87 DATAPATHS

Keywords: files, input, output

Description

This topic describes how variable DATAPATHS determines where MacAnova searches for files when you use a command such as read(), matread(), vecread() or macroread() to read a file. For all such commands, the first argument is a CHARACTER scalar or quoted string specifying a file name. See topic 'file_names'.

DATAPATHS must be a CHARACTER scalar or vector. Each element must be the name (path) of a directory, ending with a separator character (you may use '/' on all platforms, '\' in Windows or MSDOS, and ':' on Mac OS 9). See topic 'file_names' for information on path names.

Search behavior

MacAnova first looks for the file in the default directory or folder (see topic 'files').

If the file is not found in the default directory, MacAnova searches for it in the directory or folder whose name is in DATAPATHS[1], then in DATAPATHS[2] and so on. If the file is not found in any of these, you are informed that the file could not be opened.

Macro adddatapath()

You can use macro adddatapath() to add directories or folders to the start or end of DATAPATHS. See adddatapath().

Pre defined value

On windowed versions of MacAnova, DATAPATHS is initialized to have two elements. The first element is a standard place for MacAnova files in the user's directory. This is the MyMacAnovaFiles directory in the user's home directory. (The home directory is well defined on Mac OS X

```
and Unix/Linux systems; various versions of Windows have home
directories for users, often in the Documents and Settings directory.)
The second element is a standard directory in the MacAnova installation.
(See the option -appdir under topic 'launching'.)

                          Changing the value
You can change the defaults by setting DATAPATHS in a start up file, or
by setting environmental variable MACANOVA, or by command line options.
See topics 'customize' and 'launching'.

For compatibility with earlier versions of MacAnova, if variable
DATAPATHS does not exist, variable DATAPATH is substituted, if it
exists.

                               Examples
                          Windows/DOS example
Windows and DOS:
  When DATAPATHS is vector("A:/SURVEY/", "C:/MACANOVA/"),
  read("mydata.dat") first tries to read MYDATA.DAT in the default
  directory or folder and then, if not successful, tries to read
  A:\SURVEY\MYDATA.DAT and C:\MACANOVA\MYDATA.DAT.

                     Unix/Linux and Mac OS X example
Unix/Linux:
  When DATAPATHS is vector("/users/joe/survey/", "/usr/lib/macanova/"),
  read("mydata.dat") first tries to read mydata.dat in the current
  default directory, and then, if not successful, it tries to read file
  /users/joe/survey/mydata.dat and /usr/lib/macanova/mydata.dat.
```

2.88 delete()

Usage:

```
delete(var1[ ,var2, ...] [,all:T, real:T or F,char:T or F,\
  logical:T or F, structure:T or F,macro:T or F, graph:T or F,\
  lockedok:T, silent:T]), F's used only with all:T
delete(var, return:T [,invisible:T])
```

Keywords: variables, character variables

```
                                  Usage
delete(var1,var2,...,vark) deletes the variables given as arguments and
frees the memory they use for other purposes.  The variables can be of
any type including MACRO or GRAPH.  It is an error if any arguments are
'locked'.  See subtopic 'variables:"locked_variables"' and topics
lockvars() and unlockvars().

delete(var1,var2,...,vark, lockedok:T) does the same, even if one or
more of the variables are locked.

                          Deleting by attributes
You can use keywords 'real', 'char', 'logical', 'structure', 'macro',
```

'graph' and 'all' with LOGICAL values to delete classes of variables.

For example, delete(real:T, logical:T [,lockedok:T]) deletes all REAL and all LOGICAL variables and delete(all:T, macros:F [,lockedok:T]) deletes everything except macros. It is illegal to have both keyword phrases specifying attributes and variables as arguments.

Keywords 'return' and 'invisible'

delete(var, return:T) deletes variable var but returns a copy as value. This usage is most common as the last command in a macro when the value the macro is supposed to be var, as in the following

```
  Cmd> mymacro <- macro("@tmp <- ($1)+($2)
         print(describe(@tmp))
         delete(@tmp,return:T)", dollars:T)
```

mymacro(x,y) prints descriptive statistics for x+y and returns x+y as value, automatically deleting the temporary variable. See also macro(), 'macro_syntax' and 'macros'.

delete(var, return:T, invisible:T) does the same except the value is returned as an "invisible" variable (see 'variables'). The value can be assigned to a variable but will not automatically print when it is not assigned. 'invisible:T' is illegal without 'return:T'. See topic 'variables:"invisible"'.

delete(var, return:F) is also legal and has the same effect as delete(var).

Keyword 'silent'

delete(..., silent:T) suppresses all warning messages. 'silent:T' must be last argument.

Deleting special variables

delete(CLIPBOARD) does not actually delete special variable CLIPBOARD but clears its contents without affecting the system clipboard in any way. In the GTK version, delete(SELECTION) behaves the same. See topic 'CLIPBOARD'.

delete(GRAPHWINDOWS) does not actually delete special structure GRAPHWINDOWS, but sets all its components to NULL, changing their names. It does not affect what is displayed in the corresponding graphics windows. See topic 'GRAPHWINDOWS'.

Cross references

See also topics list(), listbrief().

2.89 describe()

Usage:

```
describe(data [, silent:T, excludeM:T, all:T, fivenum:T, n:T|F, min:T|F,
  max:T|F, q1:T|F, q2:T|F, median:T|F, mean:T|F, var:T|F, stddev:T|F,
  m3:T|F, m4:T|F, g1:T|F, g2:T|F, iqr:T|F, range:T|F]), where data is
  REAL or a structure with REAL components; F's should be used only with
  all:T or fivenum:T.
```

Keywords: descriptive statistics

```
                                  Usage
describe(Data) computes statistics describing the data in the REAL
vector or array Data.

The value of describe(Data) is a structure with following components:
        n            sample size, excluding MISSING values
        min          minimum
        q1           Q1 = lower quartile
        median       M = median
        q3           Q3 = upper quartile
        max          maximum
        mean         average
        var          variance (with divisor of n-1)

By default Q1 and Q3 are computed as the medians of the lower and upper
halves of the data, *including* the median in both halves when n is odd.
Keyword phrase excludeM:T (see below) changes this definition so that Q1
and Q3 are computed as medians of the lower and upper halves *excluding*
the median.

describe(Data, silent:T) does the same, but any warning messages about
MISSING values or overflows are suppressed.

describe() can compute additional statistics including the standard
deviation and the interquartile range.  See below.

                      Computing specific statistics
You can specify particular statistics to compute using keyword phrases.
For example, describe(Data, mean:T) has the same result as
describe(x)$mean, except that no unwanted statistics are computed, and
describe(Data, mean:T,var:T) returns a structure with components 'mean'
and 'var' without computing other statistics.

When only one statistic is requested, the result is a REAL variable and
not a structure.

You can use 'm1' instead of 'mean' and 'q2' instead of 'median' when
specifying what to compute; however, when other statistics are also
computed, the components still have names 'mean' and 'median'.  For
example, describe(x,m1:T,q2:T) is equivalent to describe(x,mean:T,
median:T).

                          Keyword 'fivenum'
```

```
describe(x, fivenum:T) is equivalent to describe(x,min:T,q1:T,median:T,
q3:T,max:T), that is, it computes the five number summary consisting of
the extremes and quartiles.

If you want additional statistics, say, the mean, use describe(x,
fivenum:T,mean:T).

If you want to suppress one or more of the five numbers, say the
extremes, use describe(fivenum:T,max:F,min:F).

                          Additional statistics
There are other statistics that can be computed only by using keyword
phrases.
       stddev        standard deviation = sqrt(var)
       m2            sum((x-xbar)^2)/n = s2/n = (n-1)*var/n
       m3            sum((x-xbar)^3)/n = s3/n
       m4            sum((x-xbar)^4)/n = s4/n
       g1            coefficient of skewness (see below)
       g2            coefficient of kurtosis (see below)
       range         maximum - minimum
       iqr           IQR = Q3 - Q1 = interquartile range

Some text books give m2 = s2/n as the definition of sample variance
instead of the value of var = s2/(n-1).

Example:
  Cmd> describe(x, g1:T, g2:T)

returns a structure containing the skewness and kurtosis of x in
components g1 and g2.  See below for their exact definitions.

                             Keyword 'all'
describe(Data, all:T) returns a structure with the 8 standard components
plus components stdev, m2, m3, m4, g1, g2, range and iqr.  You can
suppress any component by, for example, median:F.

Example:
  Cmd> describe(Data, all:T, q1:F, median:F, q3:F, silent:T)

returns a structure containing all statistics except the median and
quartiles.  Because 'silent:T' is an argument, no warning is printed if
Data contains MISSING values.

                           Keyword 'excludeM'
describe(Data, excludeM:T ...) is a variant, except that Q1 and Q3 are
computed as medians of the lower and upper half of the data, *excluding*
the median when n is odd, and the IQR is computed from these modified
quartiles.  This corresponds with the definition of quartiles in some
statistical texts, including David S. Moore, The Basic Practice of
Statistics.

'excludeM:T' modifies results only when the number n of non-MISSING
values is odd and one or more of Q1, Q3 or IQR is computed.
```

```
                          Case of keyword names
The case, upper or lower, of letters is ignored in describe() keyword
names.  For example, Q1:T and q1:T are equivalent.  This currently
differs from the behavior of most other functions.  The names of
components are all lower case.

                       Multidimensional argument
When Data is multidimensional (a matrix or array) with dimensions n1,
n2, ..., nk, each component of the result (or the result itself when
only one statistic is requested) is an array with dimensions n2, n3,
..., nk, that is, it has one fewer dimensions than Data.  Each statistic
describes all values with the last k-1 subscripts fixed (a column when
Data is a matrix).  In particular, when Data is a true matrix (exactly 2
dimensions), the component is a vector.  For example, when Data is a
true matrix, describe(Data, mean:T) is a vector, but
sum(Data)/nrows(Data) is a row vector (matrix with 1 row).

When Data is a vector or matrix, you can also use tabs(Data [,keywords])
to compute some of the statistics computed by describe() (not q1,
median, q3, m2, m3, m4, g1, g2, range or iqr).  See tabs().

                          Structure argument
When Data itself is a structure, each component of the result (or the
result itself when only one statistic is requested) is itself a
structure with the same shape as Data, whose components contain summary
values for the corresponding component of Data.

                                 Examples
Examples:
  Cmd> xbar <- describe(x, mean:T); sx <- describe(x, stddev:T)
compute the mean and standard deviation of x.

  Cmd> medians <- describe(split(y,a), median:T) # or MEDIAN:T
and
  Cmd> medians <- describe(split(y,a))$median # not $MEDIAN
both compute a structure, each of whose elements is the median of the
values of y corresponding to a level of factor a.  The first does less
computing of results you aren't saving.

  Cmd> describe(x, mean:T, var:T) # or Mean:T, VAR:T
and
  Cmd> describe(x)[vector(7,8)]

are equivalent, except the latter does much unnecessary computing
because it computes and then discards the extremes, the quartiles and
the median.

                          Skewness and kurtosis
Skewness g1 = k3/k2^1.5 and kurtosis g2 = k4/k2^2 are computed from
Fisher's k-statistics k2, k3 and k4 defined as

  k2 = var = s2/(n - 1)
```

```
  k3 = n*s3/((n - 1)*(n - 2)), and
  k4 = (n*(n + 1)*s4 - 3*(n - 1)*s2^2)/((n - 1)*(n - 2)*(n - 3))

g1 and g2 similar to, but not identical to, sqrt(beta1) = m3/m2^1.5 and
beta2 = m4/m2^2 - 3 which are also used to measure skewness and
kurtosis.

Expressed in terms of sqrt(beta1) and beta2:

  g1 = (sqrt(n*(n - 1))/(n - 2))*sqrt(beta1)
  g2 = ((n^2 - 1)/((n - 2)*(n - 3)))*(beta2  +  6/(n + 1))

When n <= 2, g1 is computed to be 0.  When n <= 3, g2 is computed to be
0.

g1 and g2 are sometimes used to test the null hypothesis that a sample
comes from a normal population.  If the data are a random sample from a
normal distribution, then g1 and g2 have mean 0 and

  V[g1] = 6*n*(n - 1)/((n - 2)*(n + 1)*(n + 3))
  V[g2] = 24*n*(n - 1)^2/((n - 3)*(n - 2)*(n + 2)*(n + 5))

                            Cross references
See also topics 'structures', 'keywords'.
```

2.90 descriptive()

Usage:

```
descriptive(data [options to describe()] [structure:T|F])
```

Keywords: descriptive statistics

```
                                    Usage
descriptive() is a temporary front end to describe(), adding
the single options structure:T|F.  If structure:T is chosen,
the output is just as if you had given the arguments to
describe().  If structure:F is chosen, then the result is
formed as a labelled vector or matrix rather than as a
structure.

descriptive() will only analyze one variable at a time.

See also describe().
```

2.91 design

Keywords: anova, glm, regression

```
Functions for sample size and power computations:
  power(), power2(), samplesize(), cumF(), cumstu(), cumchi()

Type, for example, 'usage(power)' or 'help(power)', to get a thumbnail
sketch or a complete description of power().

                    Contents of macro file Design.mac
File design.mac, distributed with MacAnova, contains the following
macros:
                  Macros Useful in Designing Experiments
  aliases2     Gets aliases in fractioned 2 series
  aliases3     Gets aliases in fractioned 3 series
  allaliases2  Complete aliases structure in fractioned 2 series
               factorial
  choosegen2   Chooses generators for a 2 series fractional factorial
  choosedef2   Chooses defining contrasts for a blocked 2 series
               factorial
  confound2    Confounds a 2 series factorial into blocks
  confound3    Confounds a 3 series factorial into blocks
  ems          Computes the expected mean squares for the terms in the
               ANOVA of a model some of whose terms are random
  ffdesign2    Determines factor/level combinations to use for
               fractioned 2 series

                       Macros for Statistical Analysis
  all3anova    Fits all hierarchical models in a three factor anova and
               sorts them by Cp
  all4anova    Fits all hierarchical models in a four factor anova and
               sorts them by Cp
  boxcoxvec    Gets the error SS for several boxcox transformations
  interactplot Make interaction plots of marginal means in a factorial
  interblock   Recover interblock information in an incomplete block
               design
  mixed        Computes an "ANOVA" table for a model some of whose
               factors are random
  pairwise     Does paired comparisons and generates an "underline"
               diagram
  quadmax      Finds the location of the maximum of a quadratic
               function, with optional linear equality and inequality
               constraints on the solution
  randsign     Does a permutation paired t-test
  randt        Does a permutation two-sample t-test
  randt2       Carry out a permutation two sample t-test combining
               values from two fectors (uses randt)
  rscanon      Does canonical analysis of 2nd order response surface
               design
  reml         Perform restricted maximum likelihood analysis of a model
               with fixed and random terms
  sidebyside   Make a side-by-side plot of effects
  varcomp      Computes the "ANOVA" estimates of random effects
               variances in mixed effects analysis of variance

                 Auxiliary macros used by other macros
```

```
  colproduct   Computes all element-wise products of the columns of two
               matrices (used by ems)
  makemat      Computes various basis matrices (used by ems)
  quadmaxlin   Finds the maximum of x'Ax + b'x subject to Cx = y (used
               by quadmax)

These can be retrieved by, for example, getmacros(boxcoxvec) or
boxcoxvec <- macroread("design.mac","boxcoxvec").  In a standard
installation of macanova, they are available simply by using them.

You can get information on these macros by help() or usage() or, for
example, by designhelp(makemat, varcomp) or designhelp(makemat, varcomp,
usage:T).  See topic designhelp() for details.

                            Cross references
See help() for information on direct use of help() to retrieve help
information from design.hlp.

See also topics 'macros', macroread(), getmacros(), help(), usage(),
macrousage().
```

2.92 designhelp()

Usage:

```
designhelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
designhelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
  Subtopics
designhelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
designhelp(key:Key), CHARACTER scalar Key
designhelp(index:T [,scrollback:T])
```

Keywords: general, anova, glm, regression

```
                                  Usage
designhelp(Topic1 [, Topic2, ...]) prints help on topics Topic1, Topic2,
... related to macros in file design.mac.  The help is taken from file
design.hlp.

designhelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these macros.

designhelp(index:T) or simply designhelp() prints an index of the topics
available using designhelp().  Alternatively, help(index:"design") does
the same

designhelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER
scalar or vector, prints subtopics of topic Topic.  With subtopic:"?", a
list of subtopics is printed.

designhelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where
Suptopics1 and Subtopics2 are CHARACTER scalars or vectors, prints the
```

```
specified subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to designhelp().  In windowed
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

                              Keyword 'key'
designhelp(key:key) where key is a quoted string or CHARACTER scalar
lists all topics cross referenced under Key.  designhelp(key:"?") prints
a list of available cross reference keys for topics in the file.

designhelp() is implemented as a predefined macro.

                            Cross references
See help() for information on direct use of help() to retrieve
information from design.hlp.
```

2.93 det()

Usage:

```
det(x [, mantexp:T, quiet:T]), where x is a REAL square matrix with no
  MISSING values.
```

Keywords: matrix algebra

```
                                  Usage
det(x) computes the determinant of the square REAL matrix x.  MISSING
values are not allowed in x.  When x is singular within rounding error,
det(x) is 0 and a warning message is printed.

det(x,mantexp:T) does the same, but returns the result in base 10
mantissa and exponent form.  For instance, when det(x) = -5.67832e+123,
det(x,mantexp:T) returns vector(-5.67832, 123).  This allows you to find
a determinant whose value is either too large or too close to 0 to be
represented.

det(x [,manexp:T], quiet:T) does the same, but suppresses the warning
message when x is singular.

                            Cross references
See also topics 'matrices', trace().
```

2.94 diag()

Usage:

```
diag(A), A a matrix.
```

Keywords: matrix algebra, variables

```
                                Usage
diag(a) creates a vector consisting of the diagonal elements a[1,1],
a[2,2], ... of matrix a, which does not need to be square.  The length
of the result is min(nrows(a),ncols(a)).  Matrix a may be REAL (most
common), LOGICAL or CHARACTER.

diag(a) is defined for an array with more than 2 dimensions, as long as
there are only two dimensions with size > 1, interpreted as the number
of rows and columns.  For example, after manova(model) command,
diag(SS[3,,]) is the diagonal of the sums of squares and cross-products
matrix associated with term 3 in the model.

                            Cross references
See also dmat(), trace(), 'matrices'.
```

2.95 digamma()

Usage:

```
digamma(x), x REAL with positive elements or a structure with REAL
  components with positive elements.
```

Keywords: transformations

```
                                Usage
digamma(x) returns the digamma function (first derivative of
log(gamma(x))) of the elements of x, when x is a REAL scalar, vector,
matrix or array with positive elements.  The result has the same shape
as x.

digamma(x) is equivalent to polygamma(x,0).

                          Structure argument
When x is a structure, all of whose non-structure components are REAL
with positive elements, digamma(x) returns a structure of the same shape
and with the same component names as x with each non-structure component
transformed by digamma().

                          Character argument
digamma(x) can also be used when x is a CHARACTER variable.  The result
is a CHARACTER variable of the same shape as x describing the
transformation.  See the example below.

Any element of x that is "" or starts with '@', '(', '[', '{', '<', '/'
or '\' is not modified.  This can be useful for creating labels for a
transformed variable.

                                Examples
Examples:
  Cmd> digamma(run(10)) # or polygamma(run(10),0) or polygamma(run(10))
```

```
  (1)    -0.57722     0.42278     0.92278      1.2561      1.5061
  (6)      1.7061      1.8728      2.0156      2.1406      2.2518

  Cmd> digamma(vector("x","y")) # CHARACTER argument
  (1) "digamma(x)"
  (2) "digamma(y)"

                            Cross references
See also polygamma(), lgamma(), 'transformations'
```

2.96 dim()

Usage:

```
   dim(x), x a scalar, vector, matrix, array, structure, macro or GRAPH
     variable.
```

Keywords: variables, null variables

```
                                    Usage
dim(x) returns a vector containing the dimensions of x.  For example,
when x is a vector of length 10, dim(x) has value 10.  When x is a 5 by
7 matrix, dim(x) has value vector(5,7) and dim(x)[1] and dim(x)[2] are 5
and 7, respectively.  When x is an array with k dimensions, dim(x) is a
vector of length k.

When x is a macro or GRAPH variable, dim(x) is 1.

When x is a NULL variable, dim(x) is 0. This is a change from version
4.07 and earlier when dim(NULL) was NULL.

                            Structure argument
When x is a structure, dim(x) is a structure, each of whose components
is a structure with the same shape as x.  Suppose N is the largest
number of dimensions of any component of x or is 1 if all non-structure
components of x are NULL.  Then dim(x) has N components named 'dim1',
'dim2', ....  When xcomp is a non-NULL and non-structure component of x,
the corresponding component of dim(x)[j], that is, x$dimj, is
dim(xcomp)[j] when ndims(xcomp) <= j or 0 when ndims(xcomp) < j.  When
xcomp is NULL, the corresponding component of dim(x)[j] is 0.

                                   Examples
Examples:
  dim(4) is 1
  dim(run(6)) is 6
  dim(matrix(run(20),5)) is vector(5,4)
  dim(array(run(24),2,3,4)) is vector(2,3,4)
  dim(structure(run(6),structure(matrix(run(6),3),NULL)) is
   structure(dim1:structure(6,structure(3,0)),
     dim2:structure(0,structure(2,0)))

                            Cross references
```

See also topics length(), ndims(), 'structures', 'NULL'.

2.97 dmat()

Usage:
dmat(n,val), n > 0 integer, val a REAL, CHARACTER or LOGICAL scalar
dmat(vec), vec a REAL, CHARACTER or LOGICAL vector.

Keywords: matrix algebra, variables

Usage

dmat(n,val), where n is a positive integer and val is a scalar (length(val) = 1), produces an n by n diagonal matrix with val down the diagonal.

dmat(a) where a is a vector of length n, a n by 1 matrix, or a 1 by n matrix, produces a n by n diagonal matrix with the elements of a down the diagonal.

Note: This is a sort of inverse to diag() which extracts the diagonal elements of a matrix.

Example

Example:

```
  Cmd> iden5 <- dmat(5,1); iden5 # or dmat(rep(1,5))
  (1,1)           1           0           0           0           0
  (2,1)           0           1           0           0           0
  (3,1)           0           0           1           0           0
  (4,1)           0           0           0           1           0
  (5,1)           0           0           0           0           1
```

This is the 5 by 5 identity matrix

Cross references

See also diag().

2.98 dos_windows

Keywords: general

Introduction

There are two released versions of MacAnova for Windows. The first runs under Windows 98/NT/XP and includes the familiar multi-window features, menus, dialogs, etc. The second runs in a "command prompt" window (MS-DOS window) and does not use any Windows features. Both have the full range of MacAnova commands.

This help topic first describes features or limitations specific to each version and then describes things they have in common that are specific

to Windows systems.

Windows version (CP)

This version is compiled using the Carapace library which in turn uses the WxWidgets library. It requires a Win32 system such as Windows 98/NT/XP. It allows multiple command/output and high resolution graphics windows and uses menus, dialogs, the mouse, and so forth in the usual way. Commands are typed into the lower pane of a command window, and output appears in the upper pane. Output and graphics windows can be printed and/or saved to files.

Text in a command window can be copied to the clipboard. Content of a graphics window can be copied to the clipboard as a bitmap. The MacAnova variable CLIPBOARD is connected to text on the clipboard in the sense that accessing CLIPBOARD returns a MacAnova string containing the text content of the clipboard, and assigning to CLIPBOARD writes text to the clipboard.

See topic 'carapace' for details about MacAnova for Windows.

The executable file is usually named MACANOCP.EXE.

shell(command,interact:F) and shell(command,keep:T) do not work under Windows 3.1 and Windows 95. It is possible that they work under Windows NT, but that has not been tested. shell(command,interact:T) and lines prefixed with '!' appear to behave somewhat differently, depending on the operating system. Problems remain to be worked out. See shell().

DOS version (DJ)

The DOS version is compiled using the DJGPP development suite and requires a 80386 or better processor. Although it runs under DOS, it can access all available memory and has no limits on variable sizes.

The executable file is usually named MACANODJ.EXE.

The extended memory version works with a variety of graphic displays, including VGA. Keyword phrase screendump:fileName on plotting commands allows you to create PCX files which can be edited under Windows and included in word processor documents.

It has command editing implemented using the arrow keys and editor commands based on either the Emacs or Vi editor commands (Emacs and Vi are Unix/Linux editors). See topic 'unix' for details. The only difference from the Unix/Linux version is that the special file for customizing keymaps must be "INPUTRC" (not ".inputrc") in the same directory as MACANODJ.EXE.

You can execute DOS commands by prefixing the line with '!' in the first position after the prompt or by using the command shell(). shell(cmd,keep:T) returns output from the program executed. You must use shell(cmd,interact:T) if the program executed requires any input. When in doubt, use interact:T. This feature appears to work somewhat differently under Windows 95.

```
Macro edit() (see topic edit()) is predefined to allow easy editing of
macros and data without exiting Macanova.  By default it uses the DOS
program Edit but this can be changed by setting CHARACTER variable
EDITOR to the path name of a different editor.

            Features common to all Windows and DOS versions
MacAnova recognizes '/' in file names (for instance, c:/mv/macanova.dat
instead of c:\mv\macanova.dat).  This is desirable, since to use '\' in
a quoted string it must be doubled ("c:\\mv\\macanova.dat").  See
topic 'syntax'.

Various command line arguments are recognized, allowing automatic
restoring of a workspace, suppressing the banner, changing default file
names, etc.  On the Windows version, they allow initializing the
command/output window with the contents of a file.  See topic
'launching'.

MacAnova uses any default options or file or path names in environmental
variable MACANOVA.  See topic 'customize'.

The startup file is MacAnova.ini.txt in you MyMacAnovaFiles directory.
See topic 'customize'.

Unless you use command line option -home (see 'launching') or include
-home in environmental variable MACANOVA (see 'customize'), MacAnova
pre-defines CHARACTER variable HOME to be your home directory (for
example, c:\Documents and Settings\username on Win XP). HOME is used to
expand file names of the form "~/path" or "~\\path" by substituting the
value of HOME for '~'.  This allows you to refer to files such as
MacAnova.txt.ini as "~/MyMacAnovaFiles/Macanova.ini.txt", even if you
have changed directories.  If you redefine HOME, it changes the
expansion of "~/" and "~\\".  See topic 'files'.

Pre-defined variable DATAPATHS is initialized with two path names.  The
first is effectively "~/MyMacAnovaFiles" and the second is
"foo/SharedSupport", where "foo" is the full path to the directory that
includes the MacAnova program file.  If you use command line argument
"-appdir goo", then the "foo" in the second element of DATAPATHS is
replaced by "goo".  You may prepend a search path to DATAPATHS at the
command line by using the option "-path pathName".  See topics
'DATAPATHS' and 'customize'.
```

2.99 edit()

Usage:

```
edit(obj [, T] [,stripdol:F]), obj a macro or a REAL variable
edit(0)
edit([stripdol:T])
```

Keywords: general

Usage

edit(obj) where obj is a REAL vector, matrix, array, or a macro, writes a temporary file and invokes an editor to edit that file. When editing is done, edit() returns as value the edited contents of the file. For example, 'mymacro <- edit(mymacro)' allows you to change or correct macro mymacro(). edit() is implemented as a macro which is pre-defined only in some non-windowed versions of MacAnova.

When you change the dimensions of a vector, matrix, or array, you must edit the header line to reflect the change before quitting the editor.

edit(obj,T) is equivalent to 'obj <- edit(obj)'.

Keyword 'stripdol'

By default, when editing a macro, trailing '$$' on temporary variables are stripped off for editing and then restored before returning. To suppress this, used edit(mymacro [,T], stripdol:F)

Creating new macro

mynewmacro <- edit([stripdol:F]) (with no non-keyword arguments) invokes an editor on a temporary file with a header specifying a macro with a single blank line. You can enter and edit a macro, being sure to change the number on line 1 to reflect the actual number of lines in the macro. Unless stripdol:F is an argument, '$$' will be appended to any names starting with '@'.

Creating new vector

x <- edit(0) invokes an editor on an empty temporary file and then performs vecread() on that file, under the assumption that the user has edited in data that may be read in this manner. This allows you to use an editor to enter data.

Changing default editor

You can specify the editor to be used by creating a CHARACTER variable with name 'EDITOR' by, for example, EDITOR <- "emacs". When EDITOR has not been set, the default editor is "vi" on Unix/Linux and "edit" for DOS.

Macro edit() is pre-defined only in the Unix/Linux versions and in the extended memory DOS version of MacAnova. Two forms of edit(), one for Unix/Linux and one for DOS, are included with names editunix() and editpc(), in file MacAnova.mac.txt distributed with MacAnova.

Editing in windowed versions

To edit a macro, say mymacro(), in a windowed version, first open a new command/output window using Open on the File menu. Then print the macro in the new window in a form you can edit by:

```
macrowrite(CONSOLE, mymacro, stripdol:T)
```

Then edit the macro, copy to the clipboard the entire macro including the header line (mymacro MACRO DOLLARS) and trailer line (%mymacro%), and then do the following:

```
  mymacro <- macroread(string:CLIPBOARD)

Then switch back to your original window to test it and use it.  It's
not necessary to open the extra window, but it makes it easier since you
can separate your editing from the use of the macro.

                            Cross references
See also topics macrowrite(), macroread(), 'CLIPBOARD'.
```

2.100 eigen()

Usage:

```
   eigen(x [,maxit:N, nonconvok:T]), x a REAL symmetric matrix with no
     MISSING values, integer N > 0
```

Keywords: matrix algebra

```
                                    Usage
eigen(x) computes an eigenvector/eigenvalue decomposition of the
REAL symmetric matrix x.  The result is a structure with two REAL
components, 'values' and 'vectors'.  It an error if x contains any
MISSING values.

Vector eigen(x)$values contains the eigenvalues in decreasing order
(eigen$values[i] >= eigen$values[i+1]).  If all you need are the eigen-
values, use eigenvals(x).

The columns of square matrix eigen(x)$vectors are the eigenvectors of x
with eigen$vectors[,j] corresponding to eigen$values[j].  The eigen-
vectors are orthonormal, even when there are repeated eigenvalues.

                                  Properties
From the properties of the eigenvalue/eigenvector decomposition of a
matrix,
  eigen(x)$vectors %*% dmat(eigen(x)$values) %*% eigen(x)$vectors'
should be the same as x, except for rounding error.

                              Non-convergence
It is possible for the algorithm used by eigen() not to converge,
although it rarely happens.  When it happens, the message
  ERROR: algorithm to compute eigenvalues in eigen() did not converge
is printed.  Keywords 'maxit' and 'nonconvok' may be helpful in this
situation.

                      Keywords 'maxit' and 'nonconvok'
eigen(x maxit:N), where N > 0 is an integer, computes the eigenvalues
and eigenvectors, but sets the maximum number of iterations in the
algorithm to N.  The default value is 30.  By using N > 30, this may
allow you to compute eigenvalues and vectors you can't otherwise
```

```
eigen(x [,maxit:N] ,nonconvok:T) does the same, except failure to
converge is not an error.  When convergence does not occur, no message
printed and NULL is returned.  You can use this in a macro to make it
possible to recover from failure to converge, perhaps by invoking
eigen() again using 'maxit' to increase the number of iterations.

Keyword phrases 'maxit:T' and 'nonconvok:T' may also be used on
eigenvals(), releigen() and releigenvals().

                              Cross references
See also eigenvals(), trideigen(), releigen(), and releigenvals().
```

2.101 eigenvals()

Usage:

```
   eigenvals(x [,maxit:N, nonconvok:T]), x a REAL symmetric matrix with no
     MISSING values, integer N > 0
```

Keywords: matrix algebra

```
                                    Usage
eigenvals(x) computes a REAL vector containing the eigenvalues of REAL
symmetric matrix x in decreasing order.  These are identical to
eigen(x)$values.  If you need eigenvectors, use eigen().

See eigen() for information on keyword phrases 'maxit:N' and
'nonconvok:T'.

                              Cross references
See also det(), trideigen(), releigen(), and releigenvals().
```

2.102 else

Usage:

```
   if (Logical1){command1;command2;...} [elseif (Logical2){...}] else{...}
```

Keywords: syntax, control

```
                                    Usage
'else' is a syntax element used in conjuction with 'if' and optionally
with 'elseif'.

A typical usage would be
  Cmd> if(x > 1){y <- 1}elseif(x < 0){y <- -1}else{y <- 0}

The immediately following '{' must be on the same line as 'else'.

Type help("if") for complete information.
```

2.103 elseif

Usage:
```
if (Logical1){command1;command2;...} elseif (Logical2){...}[else{...}]
```

Keywords: syntax, control

```
                                  Usage
'elseif' is a syntax element used in conjuction with 'if' and optionally
with 'else'.

A typical usage would be
  Cmd> if(x > 1){y <- 1}elseif(x < 0){y <- -1}else{y <- 0}

The condition tested (x < 0 in the example) must be a scalar LOGICAL
variable or expression.  The immediately following '{' must be on the
same line as elseif.

Type help("if") for complete information.
```

2.104 enter()

Usage:
```
x <- enter(val1 val2 val3 ...) , valI a number or ?, no separating
  commas
```

Keywords: input

```
                                  Usage
x <- enter(val1 val2 ...) is equivalent to x <- vector(val1, val2, ...)
allowing easy entry of data without the need to type commas.  val1,
val2, ... should be numbers or ?, not expressions or function values.

                                 Example
Example:
  Cmd> wts <- enter(145.2 162.1 133.5 121.9 99.8 188.9)
or
  Cmd> wts <- enter(1452 1621 1335 1219 998 1889)/10
These are equivalent, but the second avoids the need to type decimal
points.

enter() is implemented as a pre-defined macro using vecread().

                            Cross references
See also topics vecread(), enterchars().
```

2.105 enterchars()

Usage:

```
x <- enterchars(str1 str2 str3 ...) , strI a non-quoted sequence of
   visible characters, separated by spaces.
```

Keywords: input

```
                                  Usage
x <- enterchars(Str1 Str2 ...) is equivalent to x <- vector("Str1",
"Str2", ...)  allowing easy entry of CHARACTER data without the need to
type quotes (") or commas (,).  Str1, Str2, ... must consist of visible
characters which are not commas or '#' and be separated by spaces, tabs
or commas.

You cannot use enterchars() to enter CHARACTER elements which contain
commas or any kind of invisible characters such as spaces or tabs.

                                 Example
Example:
  Cmd> names <- enterchars(Henry Susan Bill Rene)

enterchars() is implemented as a pre-defined macro using vecread().

                             Cross references
See also topics vecread(), enter().
```

2.106 equal()

Usage:

```
equal(a, b [,chknames:F,chklabels:T,chknotes:T,fuzz:eps,relative:T,\
  explain:T), a and b any two defined variables except GRAPH variables,
  eps > 0 a small REAL scalar
```

Keywords: general, variables

```
                               Introduction
You can use equal() to test for exact or approximate equality of two
defined variables of any type except GRAPH.

                                  Usage
equal(arg1, arg2) returns True or False depending on whether or not arg1
and arg2 are exactly equal.  arg1 and arg2 can be any defined variables
except GRAPH variables, including macros, NULL variables, and
structures.

See below for keywords 'fuzz' and 'relative' which allow for approximate
equality and for keywords 'chknames', 'chklabels' and 'chknotes' that
control what non-data items are checked for equality.
```

If x is defined, equal(x,x) always returns True, even when x is a GRAPH variable.

Two NULL variables are always treated as equal.

When arg1 and arg2 are structures, the are considered equal only if every component or subcomponent is equal and component names are the same.

equal(arg1, arg2, explain:T) makes the same comparison but returns structure(equal:T or F, why:Explanation), where Explanation is a CHARACTER scalar that is either "Arguments are equal" or an explanation of the inequality such as "Argument values differ" or "Component dimensions differ". In case of inequality, the first applicable reason is given. If arg1 and arg2 are unequal structures, the number of the component or subcomponent where inequality was detected is part of the explanation.

Keywords 'fuzz' and 'relative'

When you are comparing numerical results of computation, exact equality may not be appropriate because real numbers are not perfectly represented in the computer. You can specify that REAL variables and components are to be considered equal if the differences between their elements are small enough, either in an absolute or relative sense.

equal(arg1, arg2, fuzz:epsilon), where epsilon > 0 is small, returns True if all differences d = x1 - x2 of elements of REAL variables or structure components satisfy abs(d) <= epsilon.

equal(arg1, arg2, fuzz:epsilon, relative:T) returns True if all differences d = x1 - x2 satisfiy abs(d) <= epsilon*(abs(x1)+abs(x2)).

Keywords 'chkxxxx'

By default, equal() compares structure component names for equality but ignores coordinate labels or attached notes. You can change this behavior by including any of 'chknames:F', 'chklabels:T' and 'chknotes:T' as arguments.

Examples

Examples:

```
  Cmd> x1 <- .001*run(3); x2 <- x1 + 1e-10

  Cmd> equal(x1,x2) # exact equality test
  (1) F

  Cmd> equal(x1,x2,fuzz:1e-9) # absolute differences small enough
  (1) T

  Cmd> equal(x1,x2,fuzz:1e-9,relative:T) # relative difference too big
  (1) F

  Cmd> equal(asLong(run(3)), asLong(run(2,4)))
  (1) F
```

```
  Cmd> a <- run(3); b <- run(4); c <- vector(a,label:"A")

  Cmd> equal(a,b) # different shapes
  (1) F

  Cmd> equal(a,b,explain:T)
  component: equal
  (1) F
  component: reason
  (1) "Argument dimensions differ"

  Cmd> equal(a,c) # labels are ignorded by default
  (1) T

  Cmd> equal(a,c,chklabels:T,explain:T) # check labels for equality
  component: equal
  (1) F
  component: why
  (1) "Argument labels differ"

  Cmd> equal(structure(a,b),structure(c,b),explain:T)
  component: equal
  (1) F
  component: why
  (1) "Structure component [1] names differ"

  Cmd> equal(structure(a,b),structure(c,b),chknames:F)#ignore comp names
  (1) T

  Cmd> A <- structure(PI,structure(a)) # nested structures

  Cmd> B <- structure(PI,structure(b)) # nested structures

  Cmd> equal(A, B, explain:T) # compare nested structures
  component: equal
  (1) F
  component: why
  (1) "Structure component [2][1] dimensions differ"

In the last example, the explanation identifies the inequality as being
in structure component 1 of structure component 2 (A[2][1] differs from
B[2][1]).

                            Cross references
See also topics 'variables', 'arithmetic', 'logic', 'notes', 'labels',
structure(), 'structures'.
```

2.107 error()

Usage:

```
error(a, b, ...[,format:Fmt or nsig:m, header:F, labels:F,
  missing:missStr, zero:zeroStr, macroname:F]), CHARACTER scalars Fmt,
  missStr and zeroStr, integer m > 0
```

Keywords: output

```
                                  Usage
error() is almost identical to print().  It differs in ways which make
it useful for reporting errors recognized in a macro.

(i) Use of error() immediately terminates execution of the current line
or macro.

(ii) error("Oops!") will print "ERROR: Oops!", where "Oops!" is a single
quoted string or CHARACTER scalar that does not start with "ERROR:" or
"WARNING:".  When the message starts with "ERROR:" or "WARNING:" error()
prints it unchanged.

(iii) In macro mymacro(), say, error("Oops!") prints
  "ERROR: Oops! in macro mymacro".

                           Keyword 'macroname'
To avoid appending the macro name when using error() in a macro, use
keyword phrase 'macroname:F'.  This is helpful when the error message
itself contains the macro name is in
   error("argument 2 to mymacro >= 1",macroname:F)

                                  Example
Example:
  Cmd> for(i,run(nrows(x))){if(x[i] >= 0){y[i] <- sqrt(x[i]);;}else{
        error("attempt to take square root of number < 0")}}

This will terminate the loop on the first x[i] < 0.  The message printed
will actually be "ERROR: attempt to take square root of number < 0".

                           Cross references
See also topics print(), write(), paste(), 'macros', 'macro_syntax',
'for', 'if'.
```

2.108 evaluate()

Usage:

```
evaluate(cmds), cmds a quoted string or CHARACTER scalar.
```

Keywords: general, syntax, macros, control

```
                                  Usage
evaluate(Cmds), where Cmds is a quoted string or CHARACTER scalar
```

```
consisting of one or more MacAnova commands, executes the commands and
returns the value of the last one.  Unlike macro expansion, symbols
starting with '$' in Cmds have no special significance.

As an argument to evaluate(), Cmds is called an "evaluated string".

                                  Example
  Cmd> b <- evaluate("a <- PI;sqrt(a)");print(a,b)
  a:
  (1)       3.1416
  b:
  (1)       1.7725

In most situations, including the example just given, <<Cmds>> is
equivalent to evaluate(Cmds).  See topic 'syntax'.

                                Limitations
An evaluated string can contain any MacAnova commands except batch() and
quit() (and its synonyms; see topic 'quitting').  There are also
restrictions on the use of 'break', 'breakall', 'next' and 'return',

  'break' and 'next' can be used only to exit or skip to the end of a
  loop that was started in the evaluated string.

  'return' can be used only to exit a macro that was invoked in the
  evaluation string.  'return' cannot be used to exit from the
  evaluated string.

  'breakall' exits the outermost loop that started in the evaluated
  string.

                              Recursive use
You can use evaluate() recursively, in the sense that evaluate() can be
one of the commands in the evaluated string.  The depth D of such a
recursion must satisfy D <= 49 - M, where M is the number of out-of-line
macros currently being evaluated.  For example, because there are no
out-of-line macros in use (M = 0), the last value of i printed by

  Cmd> i <- 0; cmd <- "i <- i+1;print(i);evaluate(cmd)";evaluate(cmd)

will be 49, followed by an error message.

                            Cross references
See also topic 'macros'.
```

2.109 exp()

Usage:

```
exp(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                  Usage
exp(x) returns the exponential function (e^x) of the elements of x, when
x is a REAL scalar, vector, matrix or array.  The result has the same
shape as x.

                       Missing or too large argument
If any element of x is MISSING or > 709.782712893, the corresponding
element of exp(x) is MISSING and a warning message is printed.

                            Structure argument
When x is a structure, all of whose non-structure components are REAL,
exp(x) is a structure of the same shape and with the same component
names as x, with each non-structure component transformed by exp().

                             Cross references
See topic 'transformations' for more information on exp().
```

2.110 factor()

Usage:

```
   factor(n1 [, n2, ...]) where n1, n2, ... are REAL scalars or vectors,
     all of whose elements are positive integers.
```

Keywords: glm, anova

```
                                  Usage
factor(A), where A is a vector of positive integers or MISSING, creates
a vector with contents identical to A except that the new vector is
marked as a "factor" with number of factor levels = max(A).

The non-MISSING elements of A must be positive integers <= 32767.

Since the number of factor levels is the largest integer in A, both
factor(vector(1,2,4)) and factor(vector(1,2,3,4)) produce factors marked
as having four levels, although only three of the levels are present in
factor(vector(1,2,4)).

Argument A can also be a matrix or array when isvector(A) is True, that
is, when all dimensions beyond the first must be 1.  In that case the
result has the same dimensions as A.

factor(a1, a2, ... ak) is equivalent to factor(vector(a1, a2, ... ak))
where a1, ..., ak are all scalars or vectors.

                            Logical argument
When A is a LOGICAL vector, factor(A) is equivalent to factor(A+1), that
is, False and True are translated to levels 1 and 2, respectively.  The
result is always marked as having 2 factor levels, even if every element
of A is False.
```

```
                                Purpose
The purpose of marking a variable as a factor is to ensure that, when it
is a variable in a model for a non-regression GLM (generalized linear or
linear model) command such as anova() or poisson(), its values are
interpreted as specifying levels of a categorical (non-quantitative)
variable, that is, classes or categories.

A vector in a model that has not been marked as a factor using factor()
is called a "variate" and its values are taken to specify quantities,
even if they are all positive integers.  In regress(), and screen()
factors are treated the same as variates -- that is the levels are
viewed as quantitative.

In a model which includes both factors and variates, the variates are
often referred to as "covariates".

                             Common mistake
A common mistake in using GLM commands is to forget to use factor() to
turn vectors of factor levels into factors.  This error results in their
being treated as variates with single degrees of freedom.

                           Subscripted factor
When A is a factor with k levels and J is an appropriate subscript for A
(for example, J might be A != 3, vector(1,run(3,length(A))) or -2), A[J]
is also marked as a factor with k levels, even if max(A[J]) < k.

                    Assignment to subscripted factor
When A is a factor with k levels, A[j] <- newvalue is legal only if
newvalue is an integer between 1 and k.  The number of levels associated
with A will not change even if max(A) < k after the replacement.

                     Maximum level less than nominal
In both these last two situations, subscripting a factor or assigning to
a subscripted factor, it is possible to create a factor whose actual
maximum level is less than k.  However, the actual maximum factor level
will be used in any analysis.

                             Cross references
See also topics makefactor(), 'models'.
```

2.111 fastanova()

Usage:

```
fastanova([Model] [,print:F or silent:T,fstats:T,pvals:T])
```

Keywords: glm, anova

```
                                  Usage
fastanova(Model) computes the analysis of variance table for the model
```

given in the CHARACTER variable Model. No variates (only factors) can be in the model. It uses an iterative algorithm rather than the modified Gram-Schmidt used by anova().

Caution: If there are empty cells in the design, the degrees of freedom and hence the mean squares may be in error.

fastanova(), with no Model specified, uses the model from the most recent GLM command such as anova() or poisson() or the model in STRMODEL.

Keywords

fastanova(Model,maxiter:N,epsilon:eps) where N is a positive integer and eps is a small REAL scalar, iterates no more than N times (default is 25) on each fit and uses the value of eps (default is 1e-6) to determine when convergence has occurred. Either keyword can appear without the other.

Other keyword phrases that can be used with fastanova() are 'print:T', 'silent:T', 'fstats:T' and 'pvals:T'. See topic 'glm_keys' for details. See topic 'options' for information on changing the default values of 'fstats' and 'pvals'.

Keyword phrases 'coefs:F' and 'marginal:T' cannot be used with fastanova().

Cross references

See topic 'models' for information on specifying Model.

Caveats

Coefficients may be retrieved by coefs(); standard errors are not available.

contrast() does not work properly after fastanova(). coefs() may or may not give the correct answers; fastanova() will warn you when coefs() will fail. cellstats() results include the estimated values for any missing data.

Method used

The iterative fitting method used by fastanova() is faster than Gram-Schmidt for large unbalanced data sets. The time used is roughly proportional to length(y)*nterms, where y is the response variable, and nterms is the number of terms in the model (the not the model degrees of freedom). Thus, fastanova() gives greatest speed improvements for models with relatively few terms, each with relatively many degrees of freedom. fastanova() is least effective for models with many terms, each with few degrees of freedom. In fact, it may be slower than anova() for such models.

2.112 file_names

Keywords: files, input, output

```
                              Introduction
This topic summarizes information about the names of files recognized by
MacAnova, absolute and relative path names, and the use of '~' and
variable HOME in abbreviating file names.

See topic 'files' for information on the default directory or folder.

See topic 'DATAPATHS' for information on CHARACTER vector DATAPATHS and
how it specifies where MacAnova searches for files.

                               File names
You specify a file name by a quoted string or CHARACTER scalar (see
topic 'scalars') such as "macanova.dat".  It must be a legal file name
for your system.  When MacAnova decides it is not a legal name it prints
the message
   ERROR: improper file name xxxxxxx.

On some systems, the checks exclude some technically legal file names.
For example, in Unix/Linux, MacAnova objects to names starting with '-'
such as "-savefile" and on a Macintosh, it objects to file names
starting with '.' such as ".savefile", because their use often leads to
difficulties.

The DOS version does not recognize long file names.

                         Use of "" as a file name
In versions with windows, you can always use the 'empty string' "" as
a file name.  This brings up a dialog box in which you can select a
folder and a file.  It also may change the default directory or folder.
See topic 'files'.

                               Path names
If your file name includes directory or folder information, it is a
'path name' and can be relative or absolute.  In the next paragraphs,
for a Macintosh or Windows 95/98/NT, 'directory' should be taken to mean
'folder'.  This information is fairly technical.

Relative paths
  Under DOS/Windows, Unix/Linux, and Mac OS X a file name that specifies
  a relative path might be "../data.dir/mydata" or
  "tseries.dir/tseries.dat" (under Windows or DOS you could replace '/'
  by '\\').

  On Mac OS 9, a relative path starts with ":".  Examples would be
  "::data.dir:mydata" and ":tseries.dir:tseries.dat".

  These examples specify a file 'mydata' in a directory 'data.dir' in
  the "parent" directory of the default directory, and a file
  'tseries.dat' in a sub-directory 'tseries.dir' in the default
  directory.
```

```
Absolute paths
  On Unix/Linux and Mac OS X, "/usr/lib/macanova/survey.dat" is a
  typical absolute path name.  It must start with '/'.

  On Windows and DOS an absolute path name starts with '/' or 'X:/'
  where 'X' is a designator such as 'A' or 'C' of a disk drive.  For
  example, if the current DOS default drive is drive 'C', both
  "C:/data.dir/survey.dat" and "/data.dir/survey.dat" specify the same
  file.

  On Mac OS 9, an absolute path name starts with a disk name and
  contains at least one ":".  Example: "MyFloppy:data.dir:survey.dat".

              Use of '~' in file names and variable HOME
You may be able to use an abbreviated file name like "~/Name" or
"~:Name" (Mac OS 9), where "Name" is the name of a file.  For this to
work, a CHARACTER scalar HOME must exist and contain the complete name
(absolute path) of a directory.  For example, in Windows if HOME is
"C:/SURVEY", the "~/mydata.dat" becomes "C:/SURVEY/mydata.dat". On
Mac OS 9, when HOME is "MyFloppy:Survey", "~:mydata.dat" becomes
"MyFloppy:Survey:mydata.dat".

On Unix/Linux, Mac OS X, and most versions of Windows, HOME is
predefined to be the user's home directory (for example, /hom/gary
or /Users/gary or c:/Documents and Settings/gary).  On DOS, HOME
is pre-defined to be the full name of the directory where the
executable MacAnova program is located.  You can override the default
for HOME using command line option -home.  See topics 'launching'
and 'customize'.

On Unix/Linux computers, a file name of the form ~name/filename, where
name is the log in name of another user, ~name is translated to the name
of that user's home directory, following a convention used in the csh
shell.
```

2.113 files

Keywords: files, input, output, missing values

```
                              Introduction
This topic describes some aspects of the use of files.  Much of it is
fairly technical and not of great interest to the casual user.  It has
sections on default directories or folders and a little information on
file format.  See topics 'file_names', 'data_files', matread_file',
'vecread_file' and 'macro_files' for information on file names and
descriptions of formats for data files and macro files.  See topic
'file_names' for information on specifying file names.

                     Default directories or folders
Generally MacAnova first looks for the named file in the default
```

directory or folder, unless the supplied file name is a "path" name, specifying a directory or folder and a file. If MacAnova does not find the file in the current directory, it then looks in the directories or folders in CHARACTER vector DATAPATHS. See topic 'DATAPATHS'.

If you start MacAnova from the command line, then the initial default directory is the current directory at the command line (on DOS, this might be different if MacAnova is started by a *.BAT file that includes a CD command).

Changing the default directory

On command line versions of MacAnova, you cannot change the default directory after startup.

In the windowed versions of MacAnova, the default directory is changed whenever you use the file navigation dialog box to select a file in a different folder or directory. Thus the first time you read a file from a given folder, you should use "" as file name. If you want, you can then read other files in the same folder by specifying their names.

File formats

On all systems, data, macro, batch, and window files must be plain text (ASCII) files. These files often have a .txt extension. If you create them using a word processor, be sure to specify plain text format when you save them. On any system, MacAnova can correctly read text files from all other systems (i.e., with Windows/DOS, Macintosh or Unix/Linux line separating codes). See topic 'data_files'.

See topics 'vecread_file' and 'matread_file' for information on how data should be organized in files to be read by vecread() and read() or matread(), respectively.

See topic 'macro_files' for information on the format of files to be read by macroread().

See batch() for information on the format of batch files.

Files written by asciisave() are ASCII files but are not meant to be read or edited by humans, and their format is arcane and subject to change. The asciisave() format is the same across computer types, so that, for example, a DOS machine can read a Macintosh asciisave() file and vice versa. asciisave() files can also be emailed as is, without encoding.

Files written by save() are binary files with formats that may differ among computer types, but most can be read on all systems.

Help files

The default help file, usually "Macanova.hlp.txt" in the SharedSupport directory below the MacAnova application, is a text file in a special format. The format is described near the start of the file. If you develop a file of macros, you could use this information to write a special help file. Or a macro file can be its own help file if you put

```
the properly formatted help after a line starting
_E_N_D_O_F_M_A_C_R_O_S_.  See topic 'macro_files'.

The help command will automatically search all the files whose names are
in pre-defined CHARACTER vector HELPFILES.  You can use addhelpfile() to
add a file name to HELPFILES.

Internally, help uses gethelp() and keywords 'file', 'orig' and 'alt'
to switch among files.

See help(), gethelp(), arimahelp(), designhelp(), graphicshelp(),
mathhelp(), mulvarhelp(), regresshelp(), tserhelp() and addhelpfile().
```

2.114 findfile()

Usage:

```
findfile(filename), filename a CHARACTER scalar
```

Keywords: files

```
findfile(filename) returns a path to the requested file, or NULL if the
file cannot be found.  Searching is done based on the paths in
DATAPATHS.  First, findfile() looks for ./filename (the current
directory), then DATAPATHS[1]/filename, DATAPATHS[2]/filename, and
so on.  findfile() returns a path to the first matching file.  Paths
may be relative or absolute depending on DATAPATHS.
```

2.115 floor()

Usage:

```
floor(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                  Usage
floor(x) rounds the elements of the REAL variable x to the next integer
in the negative direction, producing a vector, matrix, or array with the
same shape as x.

Example:
  Cmd> floor(vector(3.1416, -3.1416, 12))
  (1)           3          -4          12

                           Too large arguments
When x > 4503599627370495 or x < -4503599627370495, floor(x) is set to
MISSING because of the impossibility of exact representation of integers
```

```
beyond these limits.  These limits may be different on some computers.

                          Structure argument
If x is a structure, so is floor(x).  If xi is the i-th component of x,
the i-th component of floor(x) is floor(xi).

                           Cross references
See also topics ceiling(), round(), 'structures'.
```

2.116 for

Usage:

```
   for(i,vec)){command1;command2; ...}, vec a REAL vector
   for(i,start,end [, incr]){command1;command2;...}, start, end and incr
     REAL scalars
   for(i,NULL){...}
```

Keywords: syntax, control

```
                                  Usage
for(Index,Range){statement1;statement2;... } where Range is a REAL
vector of length N repeats the statements in {...} N times with Index
successively taking the values Range[1], Range[2], ..., Range[N].

Unless the last statement in {...} is NULL (';;') its value may be
printed on each loop.  The most common form for Range is 'run(n)' which
results in the statements in {...} being repeated n times, with variable
Index taking values 1, 2, ..., n.

for(Index,NULL){...} skips the compound statement {...} entirely.  An
example might be
  Cmd> for(i, run(length(x))[x>10]){print(paste("x[",i,"] > 10"))}
when max(x) < 10.  In this case run(length(x))[x>10] is NULL.  See
topics 'subscripts' and 'NULL'.

for(Index,i1,i2){...} is equivalent to for(Index,run(i1,i2)){...}.

for(Index,i1,i2,incr){...} is equivalent to for(Index,
run(i1,i2,incr)){...}.

                                  Value
A 'for' statement does not have a value.  Hence such constructs as
  yyy <- for(i,run(3)){i+2} or 4 + for(i,run(3)){i+2}
are illegal.

                           Early termination
You can terminate a "for loop" prematurely using syntax elements 'break'
and 'breakall' or pre-defined macro breakif().

You can skip to the end of a "for loop" using syntax element 'next'.
```

```
                                Examples
Example:
  Cmd> @n <- length(a);@s <- 0;for(i,run(@n)){@s <- @s+(a[i]-1)^2;;}; @s
is essentially equivalent to sum((vector(a)-1)^2).  'for(i,run(@n))'
could be replaced by 'for(i,1,@n)' or 'for(i,1,@n,1)'.

The following might be better:
  Cmd> @s <- 0; for(@ai,vector(a)){@s <- @s + (@ai - 1)^2;;}; @s

The opening '{' after 'for(...)' must be on the same line as 'for'.

                            Cross references
See also topics 'while', 'break', 'breakall', breakif(), 'next'.
```

2.117 fprint()

Usage:

```
   fprint(fileName, a, b, ...[,format:Fmt or nsig:m, header:F, labels:F,\
     missing:missStr, height:h, width:w]), Fmt, missStr, fileName CHARACTER
     scalars, m > 0, h >= 12 integer, w >= 30 integer
```

Keywords: output, files

```
                                  Usage
fprint(name,a,b,...) is equivalent to print(a, b, ...,file:name).  The
use of fprint() is discouraged.  Use print(a, b, ..., file:name)
instead.

See print() and write() for details on keyword use.

                            Cross references
See also write(), matprint(), matwrite(), macrowrite().
```

2.118 formatpval()

Usage:

```
   result <- formatpval(pvals [,minpval:minp] [,format:fmt]), non-negative
     REAL vector or scalar, nonnegative REAL scalar minp <= .001, CHARACTER
     scalar fmt
```

Keywords: output

```
                                  Usage
formatpval() is designed to be used in a macro to format printed
P-values similarly to the way the built-in GLM functions such as
regress(), anova() and logistic() do.

result <- formatpval(pvals), where pvals is a REAL scalar or vector with
min(pvals) >= 0, computes the CHARACTER scalar or vector result the same
```

```
length as pvals.

If pvals[i] < minP, where minP is the value of option 'minpval', then
result[i] has the form, for example, "< 1e-8", when minP = 1e-8.  If
pvals[i] >= minP, result[i] is simply paste(pvals[i]).

result <- formatpval(pvals, minpval:minP), where 0 <= minP <= .001 does
the same, except the supplied value of minP is used.

result <- formatpval(pvals, format:fmt [, minpval:minP]), where fmt is a
CHARACTER scalar specifying a legal output format (see subtopic
options:"format"), does the same, except fmt is used to format result,
instead of the default format.

                                  Example
  Cmd> tstats <- vector(4.585, 5.536, 6.576, 7.611, 8.826)

  Cmd> formatpval(twotailt(tstats,23),minpval:1e-7,format:".3g")
  (1) "0.000131"
  (2) "1.25e-05"
  (3) "1.04e-06"
  (4) "< 1e-07"
  (5) "< 1e-07"

                              Cross references
See also print(), write(), options:"minpval", options:"format",
twotailt().
```

2.119 fromclip()

Usage:

```
fromclip([ncol]), ncol > 0 an integer
```

Keywords: character variables, input

```
                                    Usage
y <- fromclip() does a vecread() from CLIPBOARD, creating a REAL vector
y.

y <- fromclip(ncol) is equivalent to matrix(fromclip(),ncol)', where
ncol is an integer.  This creates a REAL matrix with ncol columns and
makes most sense if the contents of CLIPBOARD are arranged in ncol
columns as they would be after CLIPBOARD <- x, where x is a matrix with
ncol columns.

                    Importing data in windowed versions
In windowed versions, fromclip() allows easy importing of data from
other applications such as a spreadsheet. For example, if you have
Copied to the Clipboard a 10 by 5 rectangle of spreadsheet cells
containing numbers, or 10 lines of a 5 column numerical table in a word
```

```
processor or editor,
   x <- fromclip(5)
will create a 10 by 5 REAL matrix consisting of the data copied.

fromclip() is implemented as a pre-defined macro.

                            Cross references
See also topic clipreaddata().
```

2.120 fwrite()

Usage:

```
   fwrite(fileName, a, b, ...[,format:Fmt or nsig:m, header:F, labels:F,\
     missing:missStr, width:w, height:h]), Fmt, missStr, fileName CHARACTER
     scalars, m > 0, w >= 30, h >= 12 integers
```

Keywords: output, files

```
                                  Usage
fwrite(name,a,b,...) is equivalent to write(a, b, ..., file:name).  The
use of fwrite() is discouraged.  Use write() instead.

See write() and print() for details on keyword use.

                            Cross references
See also matprint(), macroprint().
```

2.121 getascii()

Usage:

```
   getascii(charVec1 [, charVec2 ...]), all arguments CHARACTER vectors
```

Keywords: character variables

```
                                  Usage
getascii(c), where c is a CHARACTER scalar or quoted string, returns a
REAL vector with integer elements between 1 and 255 inclusive which are
the ASCII codes corresponding to the characters in c.  If c is the null
string "", getascii(c) returns NULL (see 'NULL').

getascii(c1, c2 [, c3 ...]), where all the arguments are CHARACTER
vectors, is equivalent to getascii(paste(c1,c2,..., sep:"")), collapsing
all the elements of all the arguments into one CHARACTER scalar before
decoding.  See paste().

getascii(c) is almost an inverse function to putascii(x,keep:T) in the
sense that getascii(putascii(x,keep:T)) returns x when x is a REAL
vector with integer elements between 1 and 255, and
```

```
putascii(getascii(c),keep:T) returns c when CHARACTER scalar c != "".

                              Examples
Examples:
  Cmd> getascii("ABCDE") # ascii code of 'A' is 65, etc.
  (1)           65           66           67           68           69

  Cmd> getascii(vector("AB","C"), "DE") # same as preceding
  (1)           65           66           67           68           69

  Cmd> getascii("\001\002\003\004\005") #or getascii("\1\2\3\4\5")
  (1)            1            2            3            4            5

  Cmd> getascii("\x01\x02\x03\x04\x05") # same as preceding
  (1)            1            2            3            4            5

  Cmd> getascii(putascii(run(30,34),keep:T))
  (1)           30           31           32           33           34

  Cmd> putascii(getascii("MacAnova"),keep:T)
  (1) "MacAnova"

                           Cross references
See also putascii().
```

2.122 getdata()

Usage:

```
   y <- getdata(setName), where setName is the unquoted or quoted name of a
     data set in the file whose name is in variable DATAFILE
```

Keywords: files, variables, input

```
                                  Usage
getdata() is a pre-defined macro to retrieve named data sets from a file
whose name is in CHARACTER scalar DATAFILE.  Specifically,
    y <- getdata(DataName)
is equivalent to
    y <- read(DATAFILE,"DataName")

The data set name optionally may be a quoted string, but may not be a
CHARACTER variable.

The file whose name is in DATAFILE must be in the form readable by
read() and matread().  See read(), matread() and topic 'matread_file'.

y <- getdata(DataName,quiet:T) suppresses the printing of any
descriptive comments associated with the data set in the file.

                               Data set name
The data set name dataName need not be a legal MacAnova variable name
```

```
since the name of a data set on a file readable by read() and matread()
can include characters such as '.' that are not legal in MacAnova
variable names.  For example, 'y <- getdata(jw11.5)' is legal and will
attempt to retrieve data set "jw11.5" from DATAFILE.

                            Variable DATAFILE
Variable DATAFILE is normally pre-defined to contain "macanova.dat", the
name of a sample file with several data sets.  On some systems DATAFILE
may be pre-defined to be some standard collection of data.  After

  Cmd> DATAFILE <- "disease.dat" # or other file name

getdata() will retrieve data sets from file disease.dat.

If the value of DATAFILE is a pure file name ("disease.dat" but not,
say, "data/disease.dat"), MacAnova will first search in the current
default directory or folder and then look in the directories or folders
whose names are in CHARACTER vector DATAPATHS.  See topics 'DATAPATHS'
and 'file_names'.

                       Initializing in startup file
If you regularly use a particular data file, say, "mydata.txt", you
might find it convenient to add the line DATAFILE <- "mydata.txt" to
your startup file.  Or, on Unix/Linux, Windows and DOS, you can include
'-d mydata.txt' in environmental variable MACANOVA.  See topics
'customize' and 'launching'.

                            A useful convention
A useful convention is to have the first dataset on the file have name
'info' and 0 lines (first header line 'info 0', with several comment
lines starting with ')' listing the datasets available in the file.
Then getdata(info) or even just getdata() will print this information.
See topic 'matread_file' for an example.
```

2.123 getfilename()

Usage:

```
name <- getfilename([cancelok:T])
name <- getfilename(type:"data" or type:"restore" [,cancelok:T])
name <- getfilename(fileonly:T ...[,cancelok:T])
name <- getfilename(pathonly:T ...[,cancelok:T])
name <- getfilename(help:T [,fileonly:T  or  pathonly:T])
name <- getfilename(last:T [,fileonly:T  or  pathonly:T])
```

Keywords: files, input, output

```
                                    Usage
On windowed versions

  Cmd> fileName <- getfilename()
```

brings up a file navigation dialog box in which you can select a file; the complete path name (file name with directory information) of the file is saved as a CHARACTER scalar in fileName. It is an error if the dialog box is cancelled without selecting a file.

```
Cmd> fileName <- getfilename(cancelok:T)
```

does the same except it is not an error when the dialog box is cancelled without selecting a file. It that case, fileName is set to NULL.

There are two usages that are available on any version

```
Cmd> fileName <- getfilename(help:T)
```

returns the name of the current help file.

```
Cmd> fileName <- getfilename(last:T)
```

returns the name of the last file successfully opened by any MacAnova command, either for reading or for writing.

Keywords

fileName <- getfilename(nameonly:T [,last:T or help:T] [,cancelok:T]) does the same, but only the name of the file, excluding the directory information (path), is returned.

pathName <- getfilename(pathonly:T [,last:T or help:T] [,cancelok:T]) does the same, but only the directory information is returned.

It is an error to use both nameonly:T and pathonly:T as arguments.

On a Mac OS 9, without last:T or help:T, the files displayed in the file navigation dialog box are limited to text files such as data files and workspace files created by save(). To further restrict the files displayed to text files, include keyword phrase type:"text" as an argument. Keyword 'type' can be used with 'nameonly:T' and 'pathonly:T'. It has no effect in non-Macintosh versions.

When to use

One use for getfilename() is as an argument to macro addmacrofile():

```
Cmd> addmacrofile(getfilename())
```

allows you interactively to choose a file to be added to the list of macro files to be searched.

A similar use is as an argument to adddatapath():

```
Cmd> adddatapath(getfilename(pathonly:T))
```

adds to variable DATAPATHS the name of the folder or directory containing the file selected. DATAPATHS contains a list of directories that are searched when a file is not found in the current default

```
directory.

A use for 'help:T' might be in a macro searching several help files to
enable it to restore the current help file when it was done.

A use for 'last:T' might be in a macro which uses vecread() with
'silent:T' and you want to retrieve the name of the file read.

The following macro fragment should do that:
  @filename <- argvalue($1,"file name","string")
  @x <- vecread(@filename, silent:T)
  print(paste("Reading data from file",getfilename(last:T)))
  . . . .

See topics 'macros' and 'macro_syntax' for information on writing
macros.
```

2.124 gethelp()

Usage:

```
gethelp([Topic1,Topic2,...] [,file:FileName or orig:T or alt:T]\
  [,scrollback:T, usage:T, silent:T, printname:T])
gethelp(Topic1:Subtopic1 [,Topic2:Subtopic2,...] [,other keywords]),
  Topic1, Topic2 ...  names of topics, Subtopic1, Subtopic2, ...,
  CHARACTER scalars or vectors
gethelp(Topic, subtopic:Subtopics [,other keywords])
gethelp(Topic, subtopic:"?" [,other keywords])
gethelp(Pattern [keywords]) where Pattern has form "start*",
  "start*end", or "*mid*", "start*mid*", ...
gethelp(key:KeyName [keywords]), where KeyName is a CHARACTER scalar or
  "?"
gethelp(news), gethelp(news:yymmdd1) or gethelp(news:vector(yymmdd1, \
  yymmdd2)), where yymmdd1 and yymmdd2 are integers like 990103,
  19990103, 000717 or 20000727
```

Keywords: general, files

```
                              Introduction
gethelp() retrieves help information from a "help file", a text file
in a special format.  It can be used directly, but is usually accessed
by macros help() or usage() which enable searching several files for
information.

gethelp() searches only the "current" help file.  This is initially
"macanova.hlp" but can be changed by keywords 'file', 'alt' and 'orig'
or by macros help() and usage().

                                  Usage
gethelp() with no argument prints a short message giving some of the
help() and gethelp() options.
```

```
gethelp("*") lists all help topics in the current help file.  There are
over 350 topics in the default help file, macanova.hlp, so this can be a
long list.

gethelp(Topic) prints information about the named topic.  The topic name
may be unquoted (gethelp(break)) or quoted (gethelp("break")).  Only the
current help file is searched.

gethelp(Topic, usage:T) does the same, except it gives only a brief
summary of how a command is used, without lengthy details or
explanation.  This is the same information as is provided by usage() and
getusage().  In fact, gethelp(Topic, usage:T) is equivalent to
getusage(Topic).

gethelp(topic, silent:T [,usage:T]) does the same, except any warning
messages are suppressed.  silent:T does *not* suppress printing of
help or usage information.

gethelp() returns an "invisible" LOGICAL scalar whose value is True only
when at least one topic requested was found.  The value may be assigned
or tested but will not be printed automatically.

  Cmd> if (!gethelp(topic)){print("No help found")}

See topic 'variables:"invisible"'.

                                Subtopics
Most help topics are divided into subtopics that can be accessed
individually if you know what they are.  Common subtopic names are
"usage" and "example".

gethelp(Topic, subtopic:"?") lists all subtopic names, if any,
associated with Topic.  When the topic name is no longer than 10
characters, gethelp(Topic:"?") does the same.

gethelp(Topic, subtopic:Subtopics), where Subtopics is a quoted string
or CHARACTER vector gives help only on the subtopic or subtopics named
in Subtopics.  Upper and lower case is ignored in these names and all
subtopics starting with any name in Subtopics will be listed.  In the
most common case, Subtopic is a quoted string.  No other topics can be
arguments.  When the topic name is no longer than 10 characters,
gethelp(Topic:Subtopics) does the same.

                                Examples
Examples:
  Cmd> gethelp(anova); gethelp(macros); gethelp("break")
  Cmd> gethelp(break) #nows work; previously didn't
  Cmd> gethelp(transformations) #now works; previously didn't
  Cmd> gethelp(regress, usage:T) # same as usage(regress)
  Cmd> gethelp(anova, subtopic:"example") # or gethelp(anova:"example")
  Cmd> gethelp(regress:vector("usage","example"))
  Cmd> gethelp(transformations,subtopic:"?")
```

```
  Cmd> if (!gethelp(foo,silent:T)){print("No help on topic foo")}
```

See macrousage() for a way to get usage information on currently defined macros.

See below for using keywords 'file', 'orig' and 'alt' to control the file from which gethelp() gets information.

Keyword 'scrollback'

gethelp(Topic, scrollback:T) is legal only on a windowed version. It causes the gethelp() output to be automatically scrolled back to its start. You can make this the default behavior by setoptions(scrollback:T). See topics setoptions(), 'options'

On windowed versions, the Help on the Help menu brings up a browser (web/html) based version of the standard MacAnova help. It does NOT search the standard help files. If you select a command name or other topic name in the command window with the mouse, menu item Help gives you help on that topic.

Keyword 'key' to find topic names

gethelp(key:"foo") lists all topic names associated with key "foo". Examples of keys are "Residuals", "Missing Values", "ANOVA", and "Variables". In matching keys, case (upper or lower) is ignored. Moreover, you need type only enough letters to identify a key uniquely. For example, gethelp(key:"resid") gives the same output as gethelp(key:"Residuals").

gethelp(key:"?") lists all recognized keys.

You can also use one of keywords 'file', 'orig', or 'alt' (see below) with 'key'.

Wild card characters

When you aren't sure of the exact topic name, you can use the "wild card" characters '*' and '?' in a quoted string to define a pattern to be matched. '*' will match any 0 or more successive letters in a topic name and '?' matches any single letter. All matching names are printed. If only one name matches, full help will be given on that topic.

Examples of wild card use

Examples:

```
  gethelp("res*")     lists all topic names beginning with "res"
  gethelp("*line*")   lists all topic names containing "line"
  gethelp("*anova")   lists anova,fastanova, manova, wtanova, wtmanova
  gethelp("*plot")    lists names ending in "plot" like chplot(),
                      boxplot()
  gethelp("*pl?t")    lists all topic names whose last 4 characters or
                      "p", "l", any character, and "t" repectively
                      (plot(), lineplot() and split(), for example).
  gethelp("*tat*")    lists cellstats(), rotate(), rotation()
  gethelp("add*lin*") gives help for addlines() (only 1 matching)
```

```
  gethelp("*i*e*t")   lists all topic names containing "i" and "e", in
                      that order, and ending with "t" ('assignment' and
                      lineplot(), for example).

                              Multiple topics
gethelp(Topic1,Topic2,... [,scrollback:T or usage:T] [,silent:T]) prints
information about each of the topics specified by the arguments.  If an
argument is a quoted string containing '*' or '?', it is used as a
pattern as above, but if there is more than one topic, only the first
topic in the help file whose name matches is printed.  The same rules
concerning quotes apply as when there is only one topic.

gethelp(Topic1:Subtopics1, Topic2:Subtopics2 [,...]) prints information
in the designated subtopics of Topic1, Topic2, ... .  You can also mix
in ordinary topic names without subtopics.

You can't name more than one topic when you use keyword 'subtopic'.

                         Examples of multiple topics
  gethelp(help,break) will produce this text and information about
                      syntax element 'break'
  gethelp(anova:vector("usage","examples"),option:"scrollback")
                      will print two subtopics to topic 'anova'
                      and one subtopic of topic 'option'

                              Default help file
The help information is kept in a file in a particular format.  The
default file name is macanova.hlp.  A description of the format is near
the start of the macanova.hlp

The actual file used can be changed using keyword 'file' or restored to
the default by keyword phrase 'orig:T'.  You can retrieve the name of
the actual current help file by getfilename(help:T).

Macros help() and usage() may change the current help file.  By default,
if the current help file contains the wanted topic, or if the topic is
not found in any help file, the current help file is not changed.  If
the topic is found in another file, that file becomes the current help
file.

                                Topic 'news'
gethelp(news [,scrollback:T]) lists in reverse chronological order news
items about MacAnova starting with the most recent entry back for three
months.

gethelp(news:vector(Date1,Date2) [,scrollback:T]), where Date1 and Date2
are numbers of the form yymmdd or yyyymmdd, lists in reverse
chronological order news items about MacAnova development dated between
Date1 and Date2.  For example, gethelp(news:vector(991201,0000131)) and
gethelp(news:vector(19991201,20001230)) list all news items dated in
December, 1999 or January, 2000

gethelp(news:Date) lists all news items on or after Date.  For example,
```

```
gethelp(news:000101) and gethelp(news:200000100) list all news items on
or after January 1, 2000.

gethelp(news:0) lists all available news items.  This will produce many
of lines of output and is not recommended.

                         Using other help files
Currently there are ten files distributed with MacAnova which contain
help.  Some also contain the macros they provide help for.

   File                     Contents
  ------------------------------------------------------------------------
  arima.mac              Help on macros in arima.mac
  design.hlp             Help on the macros in file design.mac
  graphics.mac           Help on macros in graphics.mac
  macanova.hlp           Help on all MacAnova commands, predefined macros
                         and general information (the default help file)
  macanova.nws           Old news items and obsolete help topics removed
                         from macanova.hlp
  math.mac               Help on macros in math.mac
  mulvar.mac             Help on macros in mulvar.mac
  regress.mac            Help on macros in regress.mac
  tser.hlp               Help on the macros in file tser.mac
  userfun.hlp            Help related to programming user functions

On a single call, gethelp() scans only a single file for help.  The
default help file is macanova.hlp.

Predefined macro help() uses gethelp() to scan all files whose names are
in CHARACTER variable HELPFILES until it finds help on a requested
topic.  The default value of HELPFILES contains all the above files
except macanova.nws and userfun.hlp.  help() is used almost identically
to gethelp().

If you know which file the topic is, you can use one of the pre-defined
macros arimahelp(), designhelp(), graphicshelp(), mathhelp(),
mulvarhelp(), regresshelp(), tserhelp() and userfunhelp() for easy
retrieval of help from the other files (except macanova.nws).  These are
used essentially the same way as help().  See topics arimahelp(),
designhelp(), graphicshelp(), mathhelp(), mulvarhelp(), regresshelp(),
tserhelp() and userfunhelp() for details.  The may have an advantage
over help() in that they scan only one file.

Or you can use gethelp() with keyword 'file'.  See below.

                        Examples for help macros
  arimahelp("*")           lists all topics in file arima.mac
  designhelp(aliases2)     prints help information on macro aliases2()
                           in design.mac
  mathhelp(i0,usage:T)     prints usage on macro i0() in math.mac
  regresshelp(key:"anova") lists topics related to key 'anova' in file
                           regress.mac
```

```
These macros make use of keywords 'file', 'orig' and 'alt' that direct
gethelp() to change help files.

                    Keywords 'file', 'orig' and 'alt'
gethelp(file:FileName) where FileName is a quoted string or CHARACTER
variable specifying the name of a file, makes that file an alternate
help file and switches over to using it.  In versions with windows,
when FileName is "", you use a dialog box to select the file.  One or
more topic names can follow file:FileName or file:FileName can be the
last argument.  The alternate help file remains active until another is
specified, or keyword phrase orig:T appears in a gethelp() command.
If the file is not in the correct format for a help file, the results
are unpredictable but not pleasant.

gethelp(orig:T) restores the help file to what it was at startup.  This
will either be the standard help file or the one specified (under
Unix/Linux or DOS) by the -h option on the command line.  See topic
'launching'.  One or more topic names can follow orig:T, or orig:T can
be the last argument.

gethelp(alt:T) restores the help file to the one most recently specified
by file:fileName.  It is an error if no alternative file was previously
set.  This allows you easily to use two help files, the default one and
one alternate.  You switch back and forth between them by
gethelp(orig:T) and gethelp(alt:T).  One or more topic names can follow
alt:T, or alt:T can be the last argument.

                            Historical notes
Prior to Version 4.12, gethelp() was named help().  help() is now a
macro that uses gethelp() to search several help files.  If you want
help() to work identically to gethelp(), you can define a simplified
version by:
  Cmd> help <- macro("gethelp($0)")

Prior to Version 4.11, topics longer than 12 characters or that were
control words ('if', 'while', 'for', 'else', 'elseif', 'break',
'breakall') had to be quoted.  This is no longer the case.


Help() is a synonym for gethelp() and is used identically.
```

2.125 gethistory()

Usage:

```
gethistory(n) where n > 0 is an integer
gethistory()
```

Keywords: general

```
                                  Usage
gethistory(n), where n is a positive integer, returns a CHARACTER vector
```

containing up to n previous commands in the order they were executed.

gethistory(), with no argument, returns a CHARACTER vector containing all available previous commands.

At most nHist - 1 commands are returned, where nHist is the value of option 'history'. See topic 'options'.

If there are no previous commands available, both gethistory(n) and gethistory() return "".

See sethistory() for information on how to replace the current internal list of previous commands.

gethistory() is not implemented in the limited memory DOS version or in any version that does not allow keyboard or menu retrieval of previous commands.

2.126 getkeywords()

Usage:

```
getkeywords(arg1 [,arg2, arg3 ...]), arg1, arg2, ... arbitrary.
```

Keywords: syntax, macros

Usage

result <- getkeywords(arg1 [, arg2, ..., argK]) returns a CHARACTER vector of length K. If argument i is a keyword phrase, result[i] is the keyword name; otherwise result[i] is "".

```
Cmd> getkeywords(cos:3, kappa:PI,a,"b",3)
(1) "cos"
(2) "kappa"
(3) ""
(4) ""
(5) ""
```

Use in macro

The principal use for getkeywords() is in a macro. It is one of the tools for analyzing the argument list to a macro. For example, it is sometimes necessary to identify which arguments are keyword phrases and which are not.

Here is a snippet of code that might be in a macro.

```
@keynames <- getkeywords($0) # check entire argument list
@keynames <- @keynames[!ismissing(@keynames)] # extract keyword names
if (!isnull(@keynames)){
  @legalkeys <- vector("short","long","narrow","wide")
  for (@i,1,length(@keynames)){
```

```
      if (match(@keynames[@i], @legalkeys, 0) == 0) {
        error(paste(@keynames[@i],"is not a legal keyword"))
      }
    }
  }
```

This checks to see that all keywords in the argument list are in a list of permissible keyword names.

Cross references

See also argvalue(), keyvalue(), 'keywords', 'macro_syntax', 'macros', match().

2.127 getlabels()

Usage:

```
getlabels(x [,silent:T,trim:F]) or getlabels(x,dims [,silent:T,trim:F]),
  dims a vector of positive integers
```

Keywords: general, variables

Usage

getlabels(x) returns the coordinate labels assocated with variable x. When x is a structure or ndims(x) = 1, the result is a CHARACTER vector of length ncomps(x) or dim(x)[1] or a CHARACTER scalar; otherwise the result is a structure with ndims(x) components, each of which a CHARACTER vector of length dim(x)[i] or a CHARACTER scalar.

A scalar is returned for a coordinate label only if all the labels for that coordinate are identical and either are "" or start with "@", in which case the first label is returned. Effectively, non-essential elements are trimmed from a vector of labels.

getlabels(x, dims), where dims is a vector of positive integers, returns the labels associated with coordinates dims[1], dims[2], ... of x in the same form as for getlabels(x). If length(dims) = 1, the result is a CHARACTER scalar or vector; otherwise it is a structure of CHARACTER scalars or vectors.

For both usages, when x has no labels, a warning message is printed and NULL is returned. See topic 'NULL'.

Keywords 'trim' and 'silent'

getlabels(x [, dims], trim:F) forces the complete labels for each requested dimension to be returned without trimming of non-essential elements.

getlabels(x [, dims], silent:T) suppresses the warning message if there are no labels.

You can determine whether a variable has labels by

```
  Cmd> if (!isnull(getlabels(x,silent:T))){...do something...}

                              Cross references
See also topics 'labels', haslabels(), addmacrofile(), adddatapath().
```

2.128 getmacros()

Usage:

```
   getmacros(name1 [,name2 ... ] [,quiet:T, silent:T, printname:F]), name1,
     name2 ... quoted or unquoted macro names to be read from one of files
     named in CHARACTER vector MACROFILES
```

Keywords: macros, files, input

```
                                      Usage
getmacros(Macro1,Macro2,...) retrieves macros Macro1, Macro2, ... from
one of the files whose names are in pre-defined CHARACTER vector
MACROFILES.  The macro names may be quoted (getmacros("mymacro")) or
unquoted (getmacros(mymacro)), but may not be CHARACTER variables.

By default, getmacros() prints the name of the file from each macro is
read.

getmacros(Macro1,Macro2,...,quiet:T) retrieves the macros but suppresses
printing the descriptive comments associated with them.

getmacros(Macro1,Macro2,...,printname:F [,quiet:T]) does the same, but
the file name or names are not printed.

If there is more than one copy of any of the named macros in the files
named in MACROFILES, getmacros() retrieves the first one found.  The
files of macros are searched in the order they are in MACROFILES.

                                Default macro files
MACROFILES is predefined with value vector("graphics.mac","regress.mac",
"design.mac","tser.mac","arima.mac","mulvar.mac","math.mac",
"macanova.mac").  Each name in MACROFILES may also include a "path" with
directory or folder information.

You can easily add files to this list using pre-defined macro
addmacrofile() (see topic addmacrofile()) or replace it entirely by,
say, MACROFILES <- vector("mymacrofile1", "mymacrofile2").  If you often
use a particular macro file or files you might find it convenient to
have MACROFILES modified in your startup file.  See topic 'customize'.

                            Automatic search for macros
Use of getmacros() is less necessary than it once was, since the default
behavior of MacAnova is now to search the files in MACROFILES for any
undefined macro you try to use.  For example, even if macro covar() has
not previously been read from file "MacAnova.mac", either by getmacros()
or macroread(),
```

```
  Cmd> cov <- covar(x)

will read covar() and then execute it.  However, is sometimes convenient
to use getmacros() to read in several macros at a time.  Moreover,
getmacros() echos the header lines on macros (unless you use quiet:T);
these often contain details about usage which you otherwise might miss.

                                  Example
Example:  If MACROFILES has its default value
  Cmd> getmacros(covar, spectrum, "confound3")
retrieves macros covar() from file "macanova.mac", spectrum() from
"tser.mac" and confound3() from "design.mac".  The quotes around
confound3() are not needed but do no harm.

Note: Prior to 4/28/96, getmacros() searched only the file specified in
CHARACTER scalar MACROFILE.  For backward compatibility, if vector
MACROFILES does not exist, getmacros() uses MACROFILE.  It is an error
if neither MACROFILES or MACROFILE exist.
```

2.129 getnotes()

Usage:

```
   getnotes(x [,silent:T]), x REAL, LOGICAL, CHARACTER or a structure,
     macro or GRAPH variable
```

Keywords: general, variables

```
                                   Usage
getnotes(x) returns the "notes" attached to variable x, if any, as a
CHARACTER scalar or vector.  If x has no notes, the result is NULL and a
warning message is printed.

getnotes(x,silent:T) does the same, except when x does not have any
attached notes, the warning message is suppressed.

x can be any type of variable, including structure, macro and GRAPH.

                              Cross references
See also topics 'notes', attachnotes(), appendnotes(), and hasnotes().
```

2.130 getoptions()

Usage:

```
   getoptions(option1:T [,option2:T ... ] [,badoptok:T]), option1, option2,
     ...  option names.
   getoptions() or getoptions(all:T) gets all option values as a structure
   Type 'usage(options)' for a succinct list of all options and their
      permissible values.
```

Keywords: control

Usage

getoptions(option1:T, option2:T, ...), where option1, option2, ... are option names, returns the values of the specified options. If more than one option is specified, the result is a structure with appropriately named components. For example, getoptions(format:T) returns the default format used in printing, getoptions(seeds:T) is equivalent to getseeds(quiet:T), and getoptions(height:T,width:T) returns a structure with components 'height' and 'width'.

getoptions() or getoptions(all:T) returns the values for all options.

getoptions(all:T, option1:F, option2:F,...) returns values for all options except those specified.

Legal option names

Legal option names are 'angles', 'batchecho', 'dumbplot', 'errors', 'findmacros', 'format', 'fstats', 'height', 'history', 'inline', 'labelabove', 'labelstyle', 'keyboard', 'maxlinelen', 'maxwhile', 'minpvalue', 'missing', 'nsig', 'pvals', 'prompt', 'restoredel', 'savehistry', 'seeds', 'traceback' 'update', 'warnings', 'wformat', and 'width'.

See topic 'options' for details on these options. Type 'usage(options)' for a list of options with legal values and defaults.

Option name 'lines' is recognized as a synonym for 'height' for compatibility with earlier versions.

On windowed versions, option 'scrollback' is also legal.

In the Mac OS 9 version, options 'font' and 'fontsize' are also legal.

Options 'format' and 'wformat'

The value returned for 'format' or 'wformat' always has the type specifier ('f' or 'g') at the end ("12.5g"), even if it was set with a string starting with 'f' or 'g' ("g12.5").

Keyword 'badoptok'

getoptions(option1:T[, option2:T, ...] ,badoptok:T) does the same, except it is not an error if the options requested are legel. If no legal options are specified, getoptions() returns NULL.

This feature is intended to allow macros using new options to be written in such a way that they are backward compatible with MacAnova versions without the new options.

Cross references

See also restore(), save().

2.131 getseeds()

Usage:

```
getseeds([quiet:T])
```

Keywords: random numbers

```
                                  Usage
getseeds() prints the current seeds and returns the current seeds in the
random number generator used by runi(), rnorm(), rbin() and rpoi() as an
"invisible" REAL vector of length 2.  See topic 'variables:"invisible"'.

getseeds(quiet:T) returns the seeds as a regular (not invisible) REAL
vector of length 2, but does not print them.

You can use getseeds() together with setseeds() to restart the random
number generators from the same point more than once.  If you retrieve
the seeds by 'seeds <- getseeds(quiet:T)', you can later reset the
random number generators to the same place by 'setseeds(seeds)'.

The seeds are saved by save() or asciisave() (unless you specify
'options:F') and are restored by restore().

                             Cross references
See also setseeds(), runi(), rnorm(), rbin(), rpoi().
```

2.132 gettime()

Usage:

```
gettime(), gettime(quiet:T), or gettime(keep:T [,quiet:F])
gettime(interval:T), gettime(interval:T,quiet:T), or
   gettime(interval:T,keep:T [,quiet:F])
```

Keywords: general

```
                                  Usage
gettime() prints the time in seconds since the start of the run.

gettime(interval:T) prints the time in seconds since the last time
gettime() was used (since the start if this is the first usage).

gettime(quiet:T) and gettime(interval:T, quiet:T) do nothing but save
the current time for the next time gettime(interval:T) is used.

gettime(keep:T [, quiet:F]) returns the time since start as a REAL
scalar.  It prints nothing unless quiet:F is an argument.

gettime(interval:T, keep:T [, quiet:F]) returns the time since last use
as a REAL scalar.  It prints nothing unless quiet:F is an argument.
```

```
                    Printing elapsed time of command
You can create a macro that will print the elapsed time of an action by
  Cmd> timeit <- macro("gettime(quiet:T);{$0};gettime(interval:T)")
Then, for example,
  Cmd> timeit(x <- rnorm(1000);stuff <- describe(x))
will print time spent generating x and computing descriptive statistics.
See also topics macro(), 'macros'.

On most computers, gettime() returns the actual time elapsed as might be
measured with a stop watch.  On a few computers, the time is the amount
of central processor time used.  This will generally be less, often much
less than the actual elapsed time.

                                 Examples
Examples:
  Cmd> gettime()
  Time since start is 377.65 seconds.

  Cmd> gettime(quiet:T);mymacro(x,y);gettime(interval:T)
  Elapsed time is 3.6718 seconds

  Cmd> gettime(quiet:T);mymacro(x,y);gettime(interval:T,keep:T)
  (1)        3.699

  Cmd> gettime(quiet:T);mymacro(x,y);gettime(interval:T,keep:T,quiet:F)
  Elapsed time is 3.6523 seconds
  (1)       3.6523
```

2.133 getusage()

Usage:

```
   getusage([Topic1,Topic2,...] [,file:FileName or orig:T or \
     alt:T, silent:T])
   getusage(Pattern) where Pattern has form "part*", "*part", or "*part*".
   getusage(key:KeyNames), where KeyNames is a CHARACTER vector or "?"
```

Keywords: general

```
                               Introduction
getusage() is used by macro usage() to retrieve usage information from a
"help file", a text file in a special format.  It can be used directly,
but has the disadvantage that it searches only one file.

getusage() searches only the "current" help file.  This is initially
"macanova.hlp" but can be changed by keywords 'file', 'alt' and 'orig'.

                                   Usage
getusage() works identically to gethelp(), except it gives a only a
brief summary of the usage of commands, functions, and macros, instead
of full details.  On some general information topics such as 'options'
and 'graph_keys' it lists available items; on other general information
```

```
topics the usage is just a suggestion to read the full help.

getusage(...) is equivalent to gethelp(..., usage:T).

getusage(..., usage:F) is equivalent to gethelp(...).

getusage() returns an "invisible" LOGICAL scalar whose value is True
only when at least one topic requested was found.  The value may be
assigned or tested but will not be printed automatically.  See topic
'variables:"invisible"'.

getusage() scans only one help file.  See gethelp() for details.
Pre-defined macro usage(), used almost identically, scans all the files
named in CHARACTER vector HELPFILES.  See topic usage().

For many commands and macros, you can get more complete usage
information by help(topic, subtopic:"usage").

                            Historical notes
Prior to Version 4.12, getusage() was named usage().  usage() is now a
macro that uses getusage() to search several help files.  If you want
usage() to work identically to getusage(), you can define a simplified
version by:
  Cmd> usage <- macro("getusage($0)")

Prior to Version 4.11, topics longer than 12 characters
('transformations', for example) or that were control words ('if',
'while', 'for', 'else', 'elseif', 'break', 'breakall') had to be quoted.
This is no longer the case.

                            Cross references
See gethelp() for full details.
```

2.134 glm

Keywords: glm, anova, categorical data, multivariate analysis, regression

```
                              Introduction
The commands for analyzing linear and generalized linear models are as
follows:
   anova(), fastanova()             Analysis of Variance
   glmfit()                         Generalized linear model analysis
   ipf(), logistic()                Logistic Regression
   manova(),                        Multivariate Analysis of Variance
   poisson()                        Log linear models
   probit()                         Probit analysis
   regress()                        Linear Regression
   robust()                         Robust Regression
   screen()                         Best subset linear regression
```

These are generally referred to as GLM commands in help topics. See their individual help entries for details. Type help(key:"glm") for a list of help entries related to analyzing linear and generalized linear models.

In addition, wtanova(), wtmanova() and wtregress() do weighted ANOVA, MANOVA and regression. Since the same computations are done when weights are specified using keyword 'weights' or 'wts' (see below), these are not further mentioned here.

glmfit() is a general function that can, with appropriate keyword arguments, be used instead of anova(), logistic(), poisson(), and probit(). In the future, additional options will allow analyses not possible at present.

All GLM commands have certain elements in common.

Model
  The first argument of a GLM command specifies a model as a quoted string or CHARACTER variable. Examples are regress("y=x1+x2+x3") and anova("x=a + a.b"). If the model is absent (for example, anova() or logistic(,n)) the most recent GLM model is assumed or the model in CHARACTER variable STRMODEL is used. Type help(models) for information on how to specify a model.

Treatment of MISSING values
  When there are MISSING values in any of the variables in a GLM model, any case with any MISSING values is omitted entirely. The maximum level of any factor is taken to be the maximum level on any of the complete data cases.

Side effect variables
  All GLM commands but screen() create certain side-effect variables. The most important are the following (not all may be produced by every command).

   STRMODEL, a CHARACTER scalar containing the model used.

   TERMNAMES, a CHARACTER vector containing the names of the terms in the model including the error terms. When the GLM command does an iterative fit without keyword phrase 'inc:T' (see topic 'glm_keys'), the value of TERMNAMES still has the same number of elements but has the form vector("","",...,"Overall model","ERROR1"), reflecting the fact that only model and error deviances are computed.

   DEPVNAME, a CHARACTER scalar containing the name of the response variable in the model.

   SS, a REAL vector of sums of squares or deviances, one for each term in the model. For manova() this is an array of SSCP matrices, with the first subscript indexing the term. Except when 'marginal:T' is an argument to anova(), manova() or robust(), these are computed sequentially and measure the importance of a term after fitting

previous terms, and ignoring later terms. The first dimension of SS has labels identical to TERMNAMES. After manova(), dimensions 2 and 3 are labeled with the column labels of the response variable if it has labels or by vector("(1)","(2)", ...) otherwise. After a GLM command that does an iterative fit without keyword phrase 'inc:T', the value of SS is vector(0,0,...,ModelDeviance,ErrorDeviance).

DF, a REAL vector containing the degrees of freedom associated with each term in the model. After a GLM command that does an iterative fit without keyword phrase 'inc:T', the value of DF is vector(0,0,...,ModelDF,ErrorDF).

RESIDUALS, a REAL vector or matrix of residuals from the fitted model. For any case with MISSING values in the data, RESIDUALS is MISSING.

WTDRESIDUALS, a REAL vector or matrix of weighted residuals from the fitted model. For analyses using iteratively re-weighted least squares sucu as logistic(), probit(), or poisson(), the weights are those used on the last iteration. For any case with MISSING values in the data, WTDRESIDUALS is MISSING. WTDRESIDUALS is not created by anova(), regress() or manova() unless weights are provided.

XTXINV (regress()), the inverse or generalized inverse of X'X or X'WX, where X is the n by k matrix of predictors, including the constant vector if it is in the model, and W is the diagonal matrix of weights, if any.

HII, the REAL vector of leverages, the diagonal elements of X(XTXINV)X' or W X(XTXINV)X', where W is the diagonal matrix of weights, if any.

COEF (regress() only), the model coefficients.

It is an error if any GLM command finds that any side effect variable is locked (see lockvars(), unlockvars(), 'variables:"locked_variables"').

Private information

Besides creating side effect variables, most GLM commands save "private" information about the analysis. This is used by commands such as regpred(), contrast(), coefs() and secoefs(). It can be retrieved by command modelinfo(). This information is not preserved by save() and asciisave() unless keyword phrase 'all:T' is used. It is discarded when you assign a value to STRMODEL or delete STRMODEL.

Cross references

See topic 'glm_keys' for a list of keyword phrases recognized by more than one GLM command.

2.135 glm_keys

Keywords: glm, anova, regression, multivariate analysis, categorical data

```
Here is a list of keyword phrases recognized by more than one GLM
command:

   Keyword phrases  Commands recognizing
   ---------------  ----------------------------------------
   print:F          All GLM commands
    Directs that most of the output to the screen is suppressed,
    although side effect variables are created.

   silent:T         All GLM commands but screen()
    Directs that all output except error messages is suppressed; side
    effect variables are computed when there are no errors,

   coefs:F          All GLM commands but screen(), regress(),
                    fastanova(), ipf(), robust();
    Suppresses the computation of coefficients or a generalized inverse
    to X'X (X'WX when there are weights).  Except in the case of
    balanced ANOVA, coefs() and secoefs() cannot be used to retrieve
    coefficients later.  In addition, some of modelinfo() options are
    effectively disabled after using 'coefs:F' on a GLM command.
    'coefs:F' is not legal with 'marginal:T'.

   fstats:T         regress(), anova(), manova(), robust()
    Directs that F-statistics and P values are computed and printed.
    The denominator is the mean square for the next following term whose
    name is of the form "ERROR1", "ERROR2", ... .  For manova(),
    statistics are given separately for each variable and printing of
    the SS/SP matrices is suppressed, although they are created as side
    effect variables.

   pvals:T          All GLM commands except screen()
    Directs that F or Chi-Squared P values are computed and printed for
    F-statistics, t-statistics, and deviances.  pvals:F suppresses
    P values when they might otherwise be printed.

   inc:T         poisson(), ipf(), logistic(), glmfit()
    Specifies that an incremental analysis of deviance table is to be
    computed and printed.  No longer legal on robust().

   marginal:T    anova(), manova(), robust()
    Specifies that SS (SS/SP matrices for manova()) are computed
    marginally.  When there are no empty cells, and sometimes when there
    are, the computed SS or SS/SP are equivalent to SAS Type III
    quantities.  'marginal:T' is not legal with 'coefs:F'.

   maxiter:n     fastanova(), poisson(), ipf(), logistic(), robust(),
                 glmfit()
    Specifies the maximum number of iterations allowed in fitting
```

```
   eps:smallVal  fastanova(), poisson(), ipf(), logistic(), robust(),
                 glmfit()
    Specifies the a threshhold in relative change of objective function
    for determining when convergence has been reached

   problimit:smallVal glmfit() with 'dist:"binomial"', logistic(),
                 probit()
    Restricts iteration so that fitted probabilities are between
    smallVal and 1 - smallVal, where 1e-15 <= smallVal < .0001.

   wts:vec       anova(), manova(), regress()
   weights:vec
    Specifies a REAL vector to be used as weights.  'wts' and 'weights'
    are equivalent.

   offsets:vec   poisson(), logistic(), probit(), glmfit()
    Causes the model to be fit to link to be 1*vec + Model, where vec is
    a REAL vector the same length as response y.  vec must be in the
    same units as the link function.  Thus for poisson() and ipf(), vec
    should be units of log(E[response]); for logistic(), vec should be
    in units of log(p/(1-p)) and for probit() vec should be in units of
    invnor(p).

    When vec is a linear combination of X-variables in the model, say
    b01*x1 + b02*x2 + b03*x3, the coefficients computed for x1, x3 and
    x5 will b1-b01, b2-b02 and b3-b03, where b1, b2 and b3 are the
    values that would be computed when offsets:vec is not an argument.
    The residual deviance is not affected.  If inc:T is an argument, the
    deviance associated with x1, x2, and x3 reflects the departure of b1
    from b01, b2 from b02, and b3 from b3.  This makes it possible to
    test a hypothesis that one or several coefficients to have specified
    values.  See logistic(), poisson() and probit() for examples.

                          Printing F and P value
If neither fstats:T nor fstats:F is an argument, for anova() and
fastanova() (but not manova()), the printing of F-statistics is
controlled by option 'fstats'.  See topic 'options'.

If neither pvals:T nor pvals:F is an argument, for all GLM commands
except manova(), robust() and screen(), the printing of P values is
controlled by option 'pvals', except that if options 'pvals' has value
False, P values will be printed if F-statistics are.

Sums of squares or deviances are normally computed sequentially.  For
anova(), manova(), and robust() these are SAS Type I quantities.  Thus
in the unbalanced case, several analyses may be necessary to compute all
the sums of squares or deviances needed.

                          Effect of 'marginal:T'
Keyword phrase, 'marginal:T', when it can be used, causes SS or SS/SP to
be computed differently.  When there are no empty cells in the design
and no aliased variates, and sometimes when there are, the SS or SS/SP
computed are SAS Type III quantities.  In every case, they are numerator
```

```
SS or SS/SP for a test that all the coeffients of non-aliased
X-variables in a term are 0, where aliasing is determined by the
original order of the sequential fit.  If there is aliasing, the
quantities computed may depend on the order in which the terms are fit.
```

2.136 glmfit()

Usage:

```
glmfit([Model] [,dist:distName,link:linkName, n:denom, incr:T,\
  print:F or silent:T, maxiter:m, epsilon:eps, coefs:F, offsets:OffVec,\
  scale:sigma]), distName and linkName CHARACTER scalars, denom > 0
  REAL scalar or vector, integer m > 0, REAL eps > 0, REAL vector OffVec
```

Keywords: glm, anova, regression, categorical data

```
                                  Usage
glmfit(Model,dist:DistName ,link:LinkName,...) does a generalized linear
model analysis with assumed response distribution DistName and link
function LinkName, somewhat in the manner of program GLIM.  The response
variable y must be a vector (isvector(y) is True).

See topic 'models' for information on and examples of quoted string or
CHARACTER scalar Model.

Current legal values for DistName are "binomial", "poisson", and
"normal" (or "gaussian").  If DistName is "binomial" or "poisson", you
must have y[i] >= 0.

Current legal values for LinkName are "logit", "probit", "log", and
"identity".

If dist:DistName is omitted, the default DistName is "normal".

If link:LinkName is omitted the default LinkName depends on DistName --
"logit" for "binomial", "log" for "poisson", and "identity" for
"normal".

Because of these defaults, glmfit(Model), with no distribution or link
specified, is equivalent to anova(Model, unbalanced:T).

                       Binomial response variable
If DistName is "binomial" you must specify the number of trials using
keyword 'n' as for logistic() or probit().  The value Denom for 'n' must
either be a REAL scalar >= max(y) or a REAL vector of the same length as
y with Denom[i] >= y[i].

                          Algorithm and output
Except when DistName is "normal" and LinkName is "identity", an
iterative algorithm is used to model link(E[y]) or link(E[y/Denom]) as a
linear function of X-variables associated with the right hand side of
Model.  Normally a two line Analysis of Deviance table is printed.  Line
```

1 is the difference 2*L(1) - 2*L(0), where L(0) is the log likelihood for a model with all coefficients 0 and L(1) is the maximized log likelihood for the model fit. Line 2 is 2*L(2) - 2*L(1) where L(2) is the maximized log likelihood under a model fitting one parameter for every y[i]. Under certain conditions, the latter can be used to test the goodness of fit of the model using a chi-squared test. When DistName is "normal" and LinkName is "identity", an Analysis of Variance table is printed including all terms.

Side effect variables created

glmfit() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS, DF, HII, DEPVNAME, TERMNAMES, and STRMODEL. See topic 'glm'. With DistName is "normal" and LinkName is "identity", SS contains the ANOVA sums of squares; otherwise SS contains deviances. After an iterative fit without keyword phrase 'inc:T' (see below), TERMNAMES has value vector("","", ...,"Overall model","ERROR1"), DF has value vector(0,0, ...,ModelDF,ErrorDF) and SS has value vector(0,0,...,ModelDeviance, ErrorDeviance).

Keyword 'inc'

glmfit(Model,dist:DistName,link:LinkName,inc:T,...) computes the full fitted model and all partial models -- only a constant term, the constant and the first term, and so on. It prints an Analysis of Deviance table, with one line for each term, representing a difference 2*L(i) - 2*L(i-1) where L(i) is the maximumized log likely for a model including terms 1 through i, plus the deviance of the complete model labeled as "ERROR1". Each line except the last can be used in a chi-squared test to test the significance of the term on the assumption that the true model includes no later terms. The value of 'inc' is ignored when DistName is "normal" and LinkName is "identity".

Relationship to other functions

The use of glmfit() provides an alternative method to specify a logistic or probit analysis of binomial responses, or a log linear analysis of Poisson responses.

Function	DistName	LinkName
logistic()	"binomial"	"logit"
probit()	"binomial"	"probit"
poisson()	"poisson"	"log"
anova()	"normal"	"identity"

In the future additional distributions such as "gamma" will be implemented, as well as additional links such as "sqrt", "recip", or "power". If you specify an unimplemented combination of LinkName and DistName, an informative error message is printed.

Problimit error message

When fitting a model with a binomial dependent variable, a warning message similar to the following

WARNING: problimit = 1e-08 was hit by glmfit() at least once

usually indicates either the presence of an extreme outlier or a best fitting model in which many of the probabilities are almost exactly 0 or

1. The latter case may not represent any problem, since the fitted probabilities at these points will be 1e-8 or 1 - e-8. You can try reducing the threshold using keyword 'problimit' (see below), but you will probably just get the message again.

Other Keyword Phrases

Keyword phrase	Default	Meaning
		Keyword 'maxiter'
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting
		Keyword 'epsilon'
epsilon:eps	1e-6	Small positive REAL specifying relative error in objective function (2*log likelihood) required to end iteration
		Keyword 'problimit'
problimit:small	1e-8	With dist:"binomial", iteration is restricted so that no fitted probabilities are < small or > 1 - small. Value of small must be between 1e-15 and 0.0001.
		Keyword 'offsets'
offsets:OffVec	none	Causes model to be fit to link to be 1*Offvec + Model, where OffVec is a REAL vector the same length as response y. OffVec must be in the same units as the link function, say, logits, logs, or probits. See topic 'glm_keys' for more information and poisson(), logistic() and probit() for examples.
		Keyword 'scale'
scale:sigma	1	sigma must be a positive REAL scalar or ? (MISSING). Its value will replace a default multiplier used by secoefs() and contrast() to compute standard errors. If the value is MISSING, sigma will be computed as sqrt(SS[m]/DF[m]), where m = length(SS). The default is 1 unless dist is "normal" when it is sqrt(SS[m]/DF[m]). In secoefs(), scale multiplies the square roots of the diagonal values of the inverse of X'WX, where X is the matrix of X-variables, and W is a diagonal matrix of weights computed using the converged fit.

Cross references

See topic 'glm_keys' for details on keyword phrases print:F, silent:T, coefs:F.

See also topics logistic(), poisson(), probit(), 'glm'.

2.137 glmpred()

Usage:

```
glmpred(variates,factors [, estimate:F, seest:F, sepred:T, n:N,
  silent:T]), variates and factors REAL vectors or matrices or NULL, N a
  positive scalar or REAL vector with positive elements.
```

Keywords: glm, regression, anova, categorical data

```
                                Usage
glmpred(Variates, Factors) computes estimates of the expected value of
the response variable y for specified values of any variates and levels
of any factors in the latest GLM model.  It returns a structure with
REAL components (vectors, except after manova()) "estimate" and "SEest".

If there are no variates in the model, Variates should be NULL (see
topic 'NULL').  Otherwise, Variates should be REAL.  If there are Nvar
variates in the model, Variates should either be a vector of length Nvar
containing values for each of the variates, or a matrix with Nvar
columns, with each row containing values for each variate.

If there are no factors in the model, Factors should be omitted or
explicitly NULL.  Otherwise, Factors should be REAL.  If there are Nfac
variates in the model, Factors should either be a vector of length Nfac
containing levels for each of the factors, or a matrix with Nfac
columns, with each row containing levels for each factor.

If either Variates or Factors contains data for only one case, it is
used for all cases.  Otherwise, you must have nrows(Variates) =
nrows(Factors).

Caution: After anova(), manova() and regress(), standard errors are
computed using the final error mean square in the model.  This may not
be appropriate with mixed models, including split plot designs.

                           Keyword 'silent'
glmpred(Variates, Factors, silent:T) does the same except that certain
advisory messages are suppressed.  'silent:T' can be used with any other
keywords.  The default value of 'silent' is False unless the value of
option' 'warnings' is False.

                Keywords 'sepred', 'seest' and 'estimate'
glmpred(Variates, Factors, sepred:T) adds component SEpred to the output
structure containing a vector or matrix of prediction standard errors.
This is only permissible after regress(), anova() or manova() and their
weighted alternatives.

glmpred(Variates, Factors, seest:F) suppresses the computation of
standard errors.

glmpred(Variates, Factors, estimate:F) suppresses the computation of
expected values.  This option is legal only after anova(), manova(),
regress() and their weighted alternatives.
```

```
You cannot use glmpred() after fastanova() or ipf() or when coefs:F was
used on the preceding GLM command.

                        After binomial response GLM
After GLM functions involving a Binomial response variable (logistic(),
probit(), glmfit(...,dist:"binomial")), the values computed are the
estimated probabilities p of "success" associated with each case (set of
values).  In this case, you can also use keyword phrase n:N, where N is
a REAL variable, to specify the number of trials for each case.  N can
be a scalar or a vector whose length matches the number of cases.  The
resulting estimated values are N*phat, where phat are the estimated
probabilities.

                        After other nonlinear GLMs
After GLM functions such as poisson(), logistic(), or probit(), where
the expectation of the response is a non-linear function of a linear
combination of the predictors, the standard error is computed from the
expectation and standard error in the linear scale using the
delta-method.  When the response is binomial and you also use n:N, the
standard errors are those of N*p.

                                Assumption
Comment: Standard errors are computed on the assumption that all effects
are fixed and not random.  When this is not appropriate, the standard
errors will usually indicate more precision than is warrented.

                                 Example
Examples:
  After regress(), glmpred(x,sepred:T) is equivalent to regpred().
  After anova("y=x+a+b"), x a variate, a and b factors,
     glmpred(x, hconcat(a,b)) computes fitted values and their standard
     errors for each case.
  glmpred(modelvars(variates:T), modelvars(factors:T)) computes fitted
  values and their standard errors for each case, regardless of the
     model.

                            Cross references
See also glmtable(), glmpred(), regpred(), modelinfo(), popmodel(),
pushmodel().
```

2.138 glmtable()

Usage:

```
   glmtable([wtdmeans:T or x:vals, estimate:F, seest:F, sepred:T, n:N]\
     [,silent:T]) or
   glmtable(Term,[wtdmeans:T or x:vals, estimate:F, seest:F, sepred:T,\
     n:N] [,silent:T]) where vals is REAL vector and TERM is CHARACTER
     scalar of form "A.B. ...", where A, B are factors in current GLM
     model, N is a positive REAL scalar or vector or array of positive
     numbers.
```

Keywords: glm, anova

```
                                  Usage
glmtable() computes tables of fitted values (estimated cell expected
values) and their standard errors based on the computations of the most
recent GLM (generalized linear or linear model) command such as anova()
or poisson().  It returns a structure with components "estimate" and
"SEest" containing the tables, each of which has a dimension for each
factor in the model, in the order the variables appear in the model.  If
there are variates in the model, the fitted values are computed with
each variate set to its unweighted mean value and thus are what are
sometimes called the covariate adjusted cell means.

If only one array of values is computed, glmtable() returns that array,
not a structure.

                               Keyword 'silent'
glmtable(silent:T) does the same except that certain advisory messages
are suppressed.  'silent:T' can be used with any other keywords.  The
default value of 'silent' is False unless the value of option'
'warnings' is False.

                     Sepred seest and estimate keywords
glmtable(sepred:T) adds component SEpred to the output structure
containing a table of prediction standard errors.  This is only
permissible after regress(), anova() or manova() and their weighted
alternatives.

glmtable(seest:F) suppresses the computation of standard errors.

glmtable(estimate:F) suppresses the computation of expected values.
This option is legal only after anova(), manova(), regress() and their
weighted alternatives.

Caution: After anova(), manova() and regress(), standard errors are
computed using the final error mean square in the model.  This may not
be appropriate with mixed models, including split plot designs.

                              Keyword 'wtdmeans'
glmtable(wtdmeans:T [,...]) does the same except it adjusts cell fitted
values to the weighted means of the variates.  You can use wtdmeans:T
only when there are variates and when the previous GLM command used
unweighted OLS (anova() or manova() with no weights supplied).  This
option would be probably appropriate when the weights were proportional
to sample sizes.

                                  Keyword 'x'
glmtable(x:Vals [,...]), where Vals is a REAL vector with length = the
number of variates (non-factors) in the model, does the same
computation, except it uses the elements of Vals instead of unweighted
or weighted variate means.  This option allows you to estimate cell
means that are adjusted to any level of the covariates.  Use of x:Vals
is an error if there are no variates in the current GLM model.
```

Marginal table

glmtable(Term [,...]) returns an estimated marginal table for the factors specified by Term. Term is a quoted string or CHARACTER scalar of the form "Name1.Name2.Name3....", where Name1, Name2, ... are names of factors in the current GLM model. If there are k factor names in Term, the value will be an array with k dimensions (vector if k = 1, matrix if k = 2), with the dimensions ordered in the same order as in the model, not the order in Term if that is different. You cannot use glmtable(term [,...]) after anova() with a balanced design unless Term includes all the factors in the model.

```
  Example:
  Cmd> glmtable("a.b", x:17) # same as glmtable("b.a",x:17)
```

glmtable(term:k [,...]) is equivalent to glmtable(TERMNAMES[k] [,...]), computing the marginal table matching term k in the model.

Example:

```
  Cmd> glmtable(term:3, x:17).
```

You can use sepred:T when estimating a marginal table.

Comment: When the marginal table for any term in the model contains empty cells, especially when a factor is nested in another with different numbers of levels, the estimated means may not be what you want.

After binomial GLM

For GLM functions involving a binomial response variable (logistic(), probit(), glmfit() with dist:"binomial"), the values computed are the estimated probabilities p of "success" associated with each cell. In this case, you can also use keyword phrase n:N, where N is a REAL variable, to specify the number of trials for each cell. N can be a scalar, a vector whose length matches the size of the table, or a matrix or array whose dimensions match those of the table. The resulting table is a table of N*p.

After other nonlinear GLMs

After GLM functions such as poisson(), logistic(), or probit(), where the expectation of the response is a non-linear function of a linear combination of the predictors, the standard error is computed from the expectation and standard error in the linear scale using the delta-method. When the response is binomial and you also use n:N, the standard errors are those of N*p. You cannot use seest:T or sepred:T after fastanova() or ipf().

Assumption

Comment: Standard errors are computed on the assumption that all effects are fixed and not random. When this is not appropriate, the standard errors will usually indicate more precision than is warrented. In particular, this is would be the case when one factor indexes replicates in a randomized block design and you use glmtable(Term,seest:T) to estimate treatment means, where Term contains all the factors except blocks.

```
                          After nonlinear GLMs
After fitting a non-linear model by logistic(), probit(), poisson(), or
glmfit(), when Term doesn't contain all the factors in the model,
glmtable(Term) first computes the estimated marginal table in the linear
scale (logit, probit, or log) and then transforms it back into the scale
of the response.  This means that the computed marginal table is not the
marginal means of the fitted table.  For example, if b is a factor with
3 levels, after logistic("y=a*b", n:40), sum(glmtable("a.b"))/3 is not
the same as glmtable("b").

                               Limitation
When keyword phrase coefs:F was an argument on the most recent GLM
command, glmtable() is not available.

                            Cross references
See also topics anova(), anovapred(), glmpred(), regpred(), modelinfo(),
popmodel(), pushmodel(), 'glm'.
```

2.139 goodfactors()

Usage:

```
goodfactors(n), n an integer scalar or vector
```

Keywords: general

```
                                  Usage
goodfactors(n), where n is a vector of positive integers < 2^52 =
4503599627370496, returns an integer vector the same length as n whose
i-th element is the smallest integer >= n[i] having no prime factors
larger than 29.  It is designed to be used to choose legal lengths of
vectors to be Fourier transformed using rft(), hft() or cft().

goodfactors(n,m), where n is the same and m is a positive integer
scalar, does the same, except the i-th element of the result is the
smallest integer >= n[i] having no prime factors larger than m.

NOTE: Because the product of "unpaired" prime factors can't be too
large, it is not guaranteed that a value returned by goodfactors() is
actually a legal length for the Fourier transform functions.  For
example, goodfactors(255255) = 255255, but 255255 = 3*5*7*11*13*17 is
not a legal length.

Example:
  Cmd> goodfactors(vector(217,1409))
  (1)          220         1421

                            Cross references
See also primefactors(), cft(), hft(), rft().
```

2.140 grade()

Usage:

```
grade(x [ ,down:T]), x REAL or CHARACTER or a structure with all REAL
  or all CHARACTER components.
```

Keywords: ordering

```
                                  Usage
grade(a) is similar to rank(a), producing a vector, matrix, or array of
the same shape as a, but with the indices of the minimum, second
smallest, ..., maximum values in each column in place of the ranks.
For example, if x is vector(3.2,1.4,5.6,2.1), grade(x) computes the
vector(2,4,1,3) since x[2], x[4], x[1], x[3] are the values of x in
increasing order.  The basic property is that, if x is a vector,
x[grade(x)] is the same as sort(x).

Argument a can be either REAL or CHARACTER.  When a is CHARACTER,
ordering is based on the ASCII collating sequence.  See sort() for the
complete ordering of characters.

grade(a,down:T) or simply grade(a,T) does the same except the underlying
sort is in decreasing order so that grade(a,down:T)[1,] computes the
case (row) numbers of the maximum of each column.

Two uses for grade() are x[grade(x[,i]),] which reorders the rows of x
so that column i is ordered, and grade(x)[1,] or grade(x,T)[1,] which
compute the indices of the minimum and maximum of each column of x.

                            Structure argument
It is also acceptable for x to be a structure, whose non-structure
components are all REAL or all CHARACTER.  In that case, grade() returns
a structure of the same form, each of whose non-structure components is
the result of applying grade() to the corresponding component of x.

                        Treatment of MISSING values
If there are k MISSING values of a column, the last k elements of the
result are the indices of the MISSING values.  For example,
grade(vector(3,1,?,0,?)) computes vector(4,2,1,3,5).  Consequently, if x
is a vector, x[grade(x)] and x[grade(x,down:T)] compute the same vector
as sort(x) and sort(x,down:T), even when there are MISSING values.

                            Treatment of ties
When there are ties, the values computed for the tied elements are
unpredictable but still satisfy that x[grade(x),] is the same as
sort(x).

                                 Examples
Examples:
  Cmd> grade(vector(27,22,25,26,22,21,?,24))
yields vector(6,2,5,8,3, 4,1,7), since the minimum (21) is in position
6, the 2nd and 3rd smallest (22) are in positions 2 and 5, . . ., the
largest (27) is in position 1, and the only MISSING value is in position
7.
```

```
  Cmd> grade(vector(27,22,25,26,22,21,?,24),down:T)
yields vector(1,4,3,8,2,5,6,7).

                            Cross references
See also sort(), rank().
```

2.141 graphicshelp()

Usage:

```
graphicshelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
graphicshelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
  Subtopics
graphicshelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
graphicshelp(key:Key), CHARACTER scalar Key
graphicshelp(index:T [,scrollback:T])
```

Keywords: general, plotting

```
                                  Usage
graphicshelp(Topic1 [, Topic2, ...]) prints help on topics Topic1,
Topic2, ... related to macros in file graphics.mac.  The help is taken
from file graphics.mac.

graphicshelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these macros.

graphicshelp(index:T) or simply graphicshelp() prints an index of the
topics available using graphicshelp().  Alternatively,
help(index:"graphics") does the same thing.

graphicshelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER
scalar or vector, prints subtopics of topic Topic.  With subtopic:"?", a
list of subtopics is printed.

graphicshelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where
Suptopics1 and Subtopics2 are CHARACTER scalars or vectors, prints the
specified subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to graphicshelp().  In windowed
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

                               Keyword 'key'
graphicshelp(key:key) where key is a quoted string or CHARACTER scalar
lists all topics cross referenced under Key.  graphicshelp(key:"?")
prints a list of available cross reference keys for topics in the file.

graphicshelp() is implemented as a predefined macro.
```

```
                          Cross references
See help() for information on direct use of help() to retrieve
information from graphics.mac.
```

2.142 graphs

Keywords: plotting

```
                        Basic plotting commands
The basic plotting commands are as follows:
  plot(x,y)                   Plot of columns of y against x
  lineplot(x,y)               Connected line plot of columns of y
                              against x
  chplot(x,y,symbols:ch)      Plot of columns of y against x using
                              symbols specified by ch
  stringplot(x,y,strings:s)   Draw labeling information in s at
                              positions defined by x and y
  boxplot(x1,x2,...,xk)       Box plots of vectors x1, ..., xk
  boxplot(structure(x1,...xk))
  addpoints(x,y)              Add data to an existing graph
  addlines(x,y)               Add line connected data to an existing
                              graph
  addchars(x,y,symbols:ch)    Add data to an existing graph using
                              symbols specified by ch
  addstrings(x,y,strings:s)   Add labeling information in s at coord-
                              inates in x and y in an existing graph
  showplot()                  Redisplay previously displayed graph

                  Arguments of basic plotting commands
Arguments x and y can be replaced by a structure with at least two
components which are interpreted as x and y.  Any additional components
are ignored.  For example, plot(x,y) and plot(structure(x,y,z)) are
equivalent.

It is not an error when x or y is NULL; a warning message is printed and
no plotting occurs.

Except for stringplot() and addstrings(), any points or lines outside
the border of the plot are omitted.  The exception for stringplot() and
addstrings() allows you to place custom labels outside the frame of the
graph.

                       Assignment to GRAPHWINDOWS
You can also draw a graph by a command like
  Cmd> GRAPHWINDOWS[1] <- structure(x:height,y:weight [,...])
where any additional arguments are graphics keyword phrases.  See topics
'graph_assign' and 'GRAPHWINDOWS' for details.

                              Keyword use
See topic 'graph_keys' for information on optional graphics keyword
phrases.  You can use some of these to specify axis labels and a title
```

or plotting limits. Examples are xmin:0, xmax:10,ymin:-1, ymax:1, xlab:"X axis label", ylab:"Y axis label", and title:"Title above graph". You can use others to specify whether the graph is to be displayed, written to a file, or be saved in the form of a GRAPH variable. Using keyword 'keys', you can specify such keyword information as components of a structure.

Other references

See topic 'graph_files' for information on how to save a plot in a file using keywords 'file', 'new, 'ps', 'screendump', and 'epsf'.

See topic 'graph_ticks' for information on using keywords 'ticks', 'xticks', 'yticks', 'xticklen', 'yticklen', 'xticklabs' and 'yticklabs' to modify default tick mark placement and labeling.

See topic 'graph_border' for information on using keyword 'borders' to control which borders will be drawn

Use of mouse

In windowed versions you can use the mouse in a graphics window to specify the positions of points, lines or rectangles to be drawn into the graph. You can do the same, with some restrictions, in the Unix/Linux version using Tektronix emulation with an emulator that implements graphical input mode. See Mouse().

Logarithmic scaling of axes

Keyword phrases 'logx:T' and/or 'logy:T' specify that logarithmic scaling is to be used for the corresponding axis. When plotted, the values are transformed to logarithms, but tick marks are still in the original units. For example, plot(x,y,logy:T) creates a semi-log plot, with a linear x-axis and a logarithmic y-axis and plot(x,y,logx:T, logy:T) creates a log-log plot.

With logx:T, any data points with x <= 0 are treated as if they had MISSING values as are data points with y <= 0 when you use logy:T. A warning message is given.

Low resolution ("dumb") plots

By default, all plotting commands produce high resolution graphs. Keyword phrase 'dumb:T' on any plotting command directs that the graph should be "dumb", that is a low resolution plot using characters that can be printed on any printer. If you prefer to have dumb plots as the default, type setoptions(dumbplot:T); 'dumb:F' will then be necessary to get high resolution plots. The default size of a dumb plot, including labels, is M lines by N - 1 character positions, where M and N are the values of options 'height' and 'width'. You can override these defaults by graphics keywords 'height' and 'width' whose values define M and/or N. See topics setoptions(), 'options' and 'graph_keys'.

Specification of data to be plotted

Commands plot(), chplot(), lineplot(), stringplot(), addpoints(), addlines(), addchars() and addstrings() all require arguments x and y which specify plotting positions. x is a REAL vector and y is a REAL

vector or matrix. If y has more than 1 column, each column is plotted against x. For addstrings(), y must be a vector of the same length as x.

Alternatively, arguments x and y can be replaced by a structure with at least two REAL components. For example, plot(structure(x,y)) is equivalent to plot(x,y). If there are more than two components, the additional ones are ignored. For example, plot(structure(x,y, info:"Test Data")) is also equivalent to plot(x,y).

Except for stringplot() and addstrings(), x can be a scalar or a vector of length 2 which implicitly specifies ny equally spaced values where ny = nrows(y). When x = x0 is a scalar x0, the implied vector is vector(x0,x0+1,x0+2,...). If x is vector(x0,dx), the implied vector is vector(x0,x0+dx,x0+2*dx, ...). Otherwise, x and y must have the same number of rows.

For plot(), chplot(), and lineplot(), if x or y are specified as keyword phrases, as in plot(Time:tm,Level:y), the keywords are used as axis labels; however, keywords 'xlab' or 'ylab' (see below) will override the name:x or name:y forms. See topic 'graph_keys'.

Examples

plot(1,y) is short for plot(run(nrows(y)),y)
plot(vector(1979,1/12),y) is short for
plot(run(0,nrows(y)-1)/12+1979,y)

The second example might be used to plot monthly data starting January 1979 against time.

GRAPH variable LASTPLOT

As a "side effect", all plotting commands create a GRAPH variable with name LASTPLOT which encapulates all the information used to create the plot. The information is saved in a resolution independent form. You can assign LASTPLOT to another variable (for example, plot1 <- LASTPLOT) or redisplay it, possibly with changed limits or labeling information, using showplot(). You can print it (as a "dumb" plot) by print(LASTPLOT) or write(LASTPLOT) or simply by typing LASTPLOT.

You can suppress the creation of LASTPLOT by keyword phrase 'keep:F'. This might be useful if you were running out of memory.

Vaariable GRAPHWINDOWS

In addition, MacAnova maintains GRAPHWINDOWS, a special structure variable, with one component for each possible graphics window (1 component in non-windowed versions). Briefly, GRAPHWINDOWS[I] (component I of GRAPHWINDOWS) is either a GRAPH variable encapsulating the plot in graphics window I, or is NULL when there is no plot in graphics window I. See topic 'GRAPHWINDOWS'.

Adding information to a plot

Commands addpoints(), addchars(), addlines(), and addstrings() allow you to display GRAPH variables with added information. Alternatively and

equivalently, you can use the keyword phrase 'add:T' as an argument to plot(), chplot(), lineplot() or stringplot(). If the first argument is a GRAPH variable (for example, addlines(graph,x,y)), the plot combines the information in the GRAPH variable with the new information provided. Otherwise, the information in LASTPLOT is used. In no case is any GRAPH variable other than LASTPLOT changed.

If graph is a GRAPH variable, plot(graph,x,y), chplot(graph,x,y, symbols:c) and lineplot(graph,x,y) are equivalent to addpoints(graph, x,y), addchars(graph,x,y,symbols:c) and addlines(graph,x,y), respectively.

To force recomputation of any of xmin, xmax, ymin or ymax to include all data use keyword phrases xmin:?, xmax:?, ymin:?, or ymax:?.

To suppress immediate display of the graph, as when you are building a complex graph in stages, use 'show:F' as an argument to each plotting command. When you are done, simply type showplot(). It is an error to use both 'show:F' and 'keep:F'.

Examples

```
  Cmd> plot(x,y,show:F); graphVar <- LASTPLOT
  Cmd> addpoints(3,4)  # or plot(3,4,add:T) or plot(LASTPLOT,3,4)
  Cmd> addpoints(graphVar,10,20,keep:F)#or plot(graphVar,10,20,keep:F)
  Cmd> showplot(xmin:0,xmax:0,ymin:0,ymax:0)
```

produces three plots. The first and third are plots of y vs x with the addition of a single point at x=3 and y=4, and the second is a plot of y vs x with the addition of a point at x = 10 and y = 20. MacAnova also recomputes the extremes displayed for the third plot. Because of 'keep:F' LASTPLOT is not updated after the second addpoints() command.

Graph Windows

On versions with windows, up to 24 windows are available for use by plotting commands. In addition, "Panel of Graphs" windows, containing miniature replicas of up to four plots in their four corners are also created.

The graph in the currently displayed window, including the panel windows, may be saved to the Clipboard by selecting Copy from the Edit menu, saved to a file by selecting Save Graph As... on the File menu, or printed by selecting Print... on the File menu.

On any plotting command you can specify which window to draw in by keyword phrase 'window:n', where 1 <= n <= 8. Keyword phrase 'window:0' means you want to reuse the most recently drawn window. This is the default on any plotting command adding information to a previous plot. This is useful for displaying a sequence of related plots that differ in the value of a parameter. If 'window:n' is not used and a plot is not being added to, the first unused window is selected, or if all windows are in use, a message is printed.

Pausing between plots

Keyword phrase 'pause:T' on any plotting command results in MacAnova

pausing after the plot is drawn. This is particularly useful when drawing repeated graphs in a 'for' or 'while' loop. Keyword phrase 'pause:F' suppresses any such pause. The default on windowed versions is pause:F while under DOS or Unix/Linux it's pause:T. On all machines, the pause can be terminated by hitting RETURN. You can use the File and Edit menus to print the graph or to Copy the graph to the Clipboard. Together with the window keyword, this permits you to display an unlimited number of graphs successively in the same window, pausing after each one to examine it and possibly print it or copy it.

Plotting on Unix/Linux

The nonwindowed version on Unix/Linux produces Tektronix-compatible plotting sequences. If MacAnova is running in a Xterm pseudo VT100 window, a pseudo Tektronix 4014 graphics window is opened and drawn to, switching back to the VT100 window when done.

The CHARACTER strings which make up the value of option 'tekset' (see subtopic 'options:"tekset"') are sent to the terminal to switch into and out of Tektronix emulation mode. MacAnova initializes the value of 'tekset' for Xterm, when that is appropriate. Otherwise, the value of 'tekset' is intialized with strings that are recognized by Versaterm, a Macintosh terminal emulator program. When you are using some other terminal emulator (such as DOS kermit or Macintosh NCSA Telnet) that supports Tektronix 4014 plotting, you will need to set option 'tekset' appropriately. Some suggested values are given in topic 'options'. See also topics vt() and tek().

2.143 GRAPHWINDOWS

Keywords: plotting, syntax

Introduction

A special variable GRAPHWINDOWS is always defined in MacAnova. It is a structure (see topic 'structures') with a component for each possible graphics window. In the DOS version and non-windowed Unix/Linux versions, GRAPHWINDOWS has only one component.

Each component of GRAPHWINDOWS is either a GRAPH variable or is NULL. When it is a GRAPH variable, it encapsulates everything used to draw the plot in the corresponding window (see topic 'variables').

In windowed versions, when you close a graphics window, the corresponding component of GRAPHWINDOWS is set to NULL.

Names of components

The name of GRAPHWINDOWS[I] (component I of GRAPHWINDOWS), where I is a positive integer, is one of the following:

Name	Meaning
Empty_I	GRAPHWINDOWS[I] is NULL
Graph_I	GRAPHWINDOWS[I] is a GRAPH variable
LASTPLOT	GRAPHWINDOWS[I] is a GRAPH variable that is identical to

GRAPH variable LASTPLOT which is normally created as a
"side effect" every time a plot is drawn.

Thus
  Cmd> compnames(GRAPHWINDOWS)
summarizes the status of all the windows. See compnames().

If the name of GRAPHWINDOWS[I] is LASTPLOT and you plot to window J != I, the name of GRAPHWINDOWS[I] is changed to Graph_I and GRAPHWINDOWS[J] is given name LASTPLOT.

After a plotting command with keyword phrase 'keep:F', GRAPHWINDOWS[I] becomes a NULL with name Empty_I. If LASTPLOT exists, it doesn't change.

When the name of GRAPHWINDOWS[I] is LASTPLOT, neither LASTPLOT <- var or delete(LASTPLOT) affect the contents of GRAPHWINDOWS[I] but its name automatically becomes Graph_I since it can no longer be guaranteed to be the same as LASTPLOT.

Deleting GRAPHWINDOWS

delete(GRAPHWINDOWS) does not remove GRAPHWINDOWS but replaces all of its components by NULL, without changing what is actually displayed in any graphics window. See delete().

Assignment to GRAPHWINDOWS

You can also assign to components of GRAPHWINDOWS. For details see topic 'graph_assign'. Briefly, the behavior is as follows.

GRAPHWINDOWS[I] <- NULL makes component I of GRAPHWINDOWS NULL without changing what is displayed or affecting LASTPLOT.

GRAPHWINDOWS[I] <- graphVar, where graphVar is a GRAPH variable, makes GRAPHWINDOWS[I] identical to graphVar and displays its new contents in graphics window I. GRAPH variable LASTPLOT is set identical to graphVar and GRAPHWINDOWS[I] is renamed LASTPLOT.

GRAPHWINDOWS[I] <- structure(graphics key words) modifies or replaces component GRAPHWINDOWS[I] and displays it in graphics window I. GRAPH variable LASTPLOT is set identical to GRAPHWINDOWS[i] which is renamed LASTPLOT.

NULL components

GRAPHWINDOWS[I] can be NULL for one of three reasons: (a) Nothing has been drawn in the window or it has been closed; (b) keyword phrase 'keep:F' was used when the graph in the window was drawn; or (c) the graph variable has been cleared, either by delete(GRAPHWINDOWS) or by GRAPHWINDOWS[I] <- NULL.

Effect of save() and restore()

By default, save() and asciisave() save GRAPHWINDOWS as part of the saved workspace, although this can be suppressed by keyword phrase 'graphwind:F'. When the saved workspace is restored, GRAPHWINDOWS is

```
restored.  In windowed versions, the restored graphs are redrawn,
replacing any existing graphs (see restore()).  The names of the
restored components are all either Empty_I or Graph_I, even if one of
the original components of GRAPHWINDOWS was named LASTPLOT.

                            Cross references
See topic 'graphs' and 'graph_keys' for general information about
plotting in MacAnova.
```

2.144 graph_assign

Usage:

```
   GRAPHWINDOWS[I] <- gVar, positive integer I, GRAPH variable gVar
   GRAPHWINDOWS[I] <- structure(x:xvalues,y:yvalues [keyword phrases]),
     xvalues REAL vector, yvalues REAL vector or matrix
   GRAPHWINDOWS[I] <- structure(x:xvalues,y:yvalues, symbols:ch\
     [, keyword phrases])
   GRAPHWINDOWS[I] <- structure(x:xvalues,y:yvalues, lines:T\
     [, keyword phrases])
   GRAPHWINDOWS[I] <- structure(x:xvalues,y:yvalues, strings:s\
     [, keyword phrases]), yvalues a REAL vector
   GRAPHWINDOWS[I] <- structure(graphics keyword phrases without 'x', 'y')
   GRAPHWINDOWS[I] <- NULL

   Type help(GRAPHWINDOWS) for information on special variable GRAPHWINDOWS
```

Keywords: plotting, syntax

```
                               Intoduction
This topic describes the effect of GRAPHWINDOWS[I] <- var, where
GRAPHWINDOWS is a special structure variable described in topic
'GRAPHWINDOWS'.

GRAPHWINDOWS[I], component I of GRAPHWINDOWS, is either a GRAPH variable
encapsulating all the information used to draw the plot in the I-th
graphics window or NULL.  You can draw, modify or label a plot in a
graphics window by GRAPHWINDOWS[I] <- var, where var is a GRAPH
variable, a structure or NULL.  This facility supplements the use of
plotting commands such as plot(), lineplot(), chplot(), stringplot() and
showplot().

In the following, I is an integer.  Currently in the windowed versions,
I must satisfy 1 <= I <= 8 and in the non-windowed versions I = 1.

                            Assignment of NULL
  Cmd> GRAPHWINDOWS[I] <- NULL
sets GRAPHWINDOWS[I] to NULL and renames the component "Empty_I".  If
GRAPHWINDOWS[I] contained a GRAPH variable, it is discarded.  This has
no effect on what, if anything, is displayed in graphics window I or on
the value of variable LASTPLOT if it exists.
```

```
                    Assignment of graph variable
  Cmd> GRAPHWINDOWS[I] <- graphVar
graphVar a GRAPH variable, displays the plot in graphics window I and
then makes both GRAPHWINDOWS[I] and LASTPLOT identical to graphVar.
That is, it does the same as showplot(graphVar, window:I) (simply
showplot(graphVar) in non-windowed versions).  Moreover, GRAPHWINDOWS[I]
is renamed LASTPLOT.

                       Assignment of structure
  Cmd> GRAPHWINDOWS[I] <- Str
where Str is a structure, can duplicate the behavior of plot(),
lineplot(..., window:I), chplot(..., window:I), stringplot(...,
window:I) and showplot(..., window:I).  Str must contain at least one
component whose name is the same as a graphics keyword (see topic
'graph_keys') and such keyword phrases determine what happens.  Any
components with graphics keyword names must have the same types of
values as the corresponding keyword.  For example, a component named
'lines' must be a LOGICAL scalar, T or F and a component 'x' must be a
REAL vector.

Str may always include components names 'x', 'y', 'xlab', 'ylab' and
'title', as well 'ticks', 'border', 'xticks', 'yticks', 'xticklen',
'yticklen', 'xticklabs' and 'yticklabs'.  When 'x' and 'y' are
components, Str may also have component 'add'.  Components 'keep' and
'show' are allowed but must have value True.  If Str has component
'window', its value must be I.  You may not use 'file' or other keywords
related to files as component names.

If Str does not have components 'x' and 'y', the assignment behaves like
showplot() and Str should not have components 'lines', 'linetype',
'thickness', 'impulses', or 'add'.

If Str has components 'x', 'y' and 'symbols', the assignment behaves
like chplot().  With 'add:T', it behaves like addchars().  Str can also
have components 'lines', 'linetype', 'thickness' and 'impulses'.

If Str has components 'x', 'y' and 'strings', the assignment behaves
like stringplot().  With 'add:T', it behaves like addstrings().  Str
must not have components 'lines', 'linetype', 'thickness' or 'impulses'.

If Str has components 'x', 'y' and 'lines' but not 'symbols', the
assignment behaves like lineplot().  With 'add:T', it behaves like
addlines().

If Str has components 'x' and 'y', but not 'symbols', 'strings' or
'lines', the assignment behaves like plot().  With 'add:T', it behaves
like addpoints().  Str can also have component 'impulses' but not
'linetype' or 'thickness'.

                              Use of mouse
In some versions, you can use Mouse() to create Str with coordinates
taken interactively from a graphics window.  See Mouse() for details.
```

```
                                Examples
Examples:
  Cmd> GRAPHWINDOWS[3] <- structure(x:temp, y:percent, lines:T,\
    symbols:"\1", xlab:"Temperature", ylab:"Percent")

does the same as

  Cmd> lineplot(temp, percent, lines:T, symbols:"\1",window:3,\
    xlab:"Temperature", ylab:"Percent")

Both also create GRAPH variable LASTPLOT which is identical to
GRAPHWINDOWS[3].

  Cmd> GRAPHWINDOWS[4] <- graphVar # GRAPH variable graphVar

does the same as

  Cmd> showplot(graphVar, window:4)

LASTPLOT is created identical to graphVar and GRAPHWINDOWS[4].

  Cmd> GRAPHWINDOWS[1] <- structure(x:17.3, y:25, add:T,\
    strings:"Observations made July 3, 1998", justify:"l")

does the same as

  Cmd> stringplot(GRAPHWINDOWS[1], 17.3, 25, add:T, window:1,\
    strings:"Observations made July 3, 1998", justify:"l")

In some versions
  Cmd> stuff <- Mouse(lines:T,n:10);GRAPHWINDOWS[stuff$window] <- stuff

plots a segmented line with 10 segments with the end and join points
specified by clicking in a graphics window with the mouse.

                            Cross references
See also topics 'graphs', 'graph_keys', plot(), lineplot(), chplot(),
stringplot(), showplot(), addpoints(), addchars(), and addstrings().
```

2.145 graph_border

Usage:

```
   Graphics keyword phrase 'borders:word' controls on which sides of a
     graph borders should be drawn. Possible values for 'word' are "all",
     "none" or some combination of "B", "L", "T", and "R" or their lower
     case counterparts.
```

Keywords: plotting, files, output

```
                            Specifying borders
By default, borders are drawn on all four sides of a graph, providing a
```

```
rectangular frame.  You can modify this behavior using graphics keyword
phrase 'borders:word', where word is a quoted string or character
scalar.  If word isn't "all" (the default) or "none", it must contain
only the characters 'B' (bottom), 'L' (left), 'T' (top) or 'R' (right),
or their lower case counterparts 'b', 'l', 't' and 'r'.  Value "" is
equivalent to "none".

Keyword phrase 'borders:word' also sets the edges where tick marks will
be drawn, unless 'ticks:word' is also an argument.  See topic
'graph_ticks'.

                                Examples
Examples:
  Cmd> plot(x,y,borders:"LB") # or plot(x,y,borders:"lb")
This produces a plot with border and ticks drawn only at left and
bottom.

  Cmd> plot(x,y,borders:"none") # or plot(x,y,borders:"")
This produces a plot with no border or ticks drawn.
```

2.146 graph_files

Keywords: plotting, files, output

```
                              Introduction
This topic summarizes those plotting options allowing you to save a plot
in a file.  See also topics 'graphs', 'graph_keys', 'files'.

                             Keyword 'file'
When keyword phrase file:FileName is an argument to a plotting command,
no plot is displayed.  Instead, PostScript commands for the new plot are
written to file FileName, which must be a CHARACTER scalar or string.
If keyword phrase landscape:T is also present, the plot will be rotated
to fill an 8.5" by 11" page.  If option 'dumbplot' has been set to True
(see subtopic 'options:"dumbplot"'), you will need to put dumb:F to get
a PostScript file.

With file:fileName, if keyword phrase dumb:T or ps:F is also an
argument, a low resolution "dumb" plot is written rather than
PostScript.  This consists only of characters that can be printed on any
printer.  On some systems, if ps:F appears and dumb:T does not, the plot
is written in a binary form appropriate to the platform (Tektronix
commands on Unix/Linux; PICT format on Mac OS 9).

If the keyword phrase 'new:T' is an argument, the information in the
file is destroyed before the plotting commands are written.  Otherwise,
information is added at the end of the file.  It formerly was the case
tha a PostScript "prolog" was written only with new:T but now a prolog
is always written before PostScript commands.

                          Encapsulated postscript
```

On Mac OS 9 you can write the plot as an encapsulated PostScript file by using 'epsf:T, file:fileName'. The file can be imported into some word-processors and graphics editing programs. epsf:T is illegal with new:F or ps:F.

Binary formatted files

On Mac OS 9 and in the DOS extended memory version you can write a binary version of the plot to a file using keyword phrase screendump:fileName. The format used is one appropriate to the computer. The Mac OS 9 version writes a so called PICT file and the extended memory DOS version writes a PCX file. 'screendump:fileName' is not legal together with dumb:T or file:FileName.

On Mac OS 9, item Save Graph As on the File menu writes the graph window currently being displayed as a PICT file.

Cross references

See topics 'data_files', 'matread_file', 'vecread_file' and 'macro_files' for information on files containing sets and macros.

2.147 graph_keys

Usage:

List of keywords which can be used on some or all graphing commands or to name a component of the value of graphics keyword 'keys'.

Keyword	Value
add	T or F
borders	"all", "none" or combination of "B","L","T","R"
dumb	T or F
epsf	T or F
file	CHARACTER scalar file name
height	Integer >= 12
impulse	T or F
keep	F or T
keys	structure with graphics keyword component names
landscape	T or F
lines	T or F
linetype	Integer between -99 and 99
logx	T or F
logy	T or F
new	T or F
notes	CHARACTER vector
pause	T or F
ps	F or T
screendump	CHARACTER scalar file name
show	F or T
silent	F or T
strings	CHARACTER scalar or vector
symbols	CHARACTER or integer scalar, vector or matrix
thickness	REAL scalar between .1 and 10
ticks	"all", "none" or combination of "B","L","T","R"
title	CHARACTER scalar (<= 75 characters)
width	Integer >= 30
window	Integer between 0 and 8
x	REAL vector or NULL (only in 'keys')
xaxis	F or T
xlab	CHARACTER scalar (<= 50 characters)
xmax	REAL scalar or ?
xmin	REAL scalar or ?
xticklabs	CHARACTER vector
xticklen:	REAL scalar >= -1
xticks	REAL vector, ? or NULL
y	REAL vector or matrix or NULL (only in 'keys')
yaxis	F or T
ylab	CHARACTER scalar (<= 20 characters)
ymax	REAL scalar or ?
ymin	REAL scalar or ?
yticklabs	CHARACTER vector
yticklen	REAL scalar >= -1
yticks	REAL vector, ? or NULL

Topic 'graphs' has general information on making graphs.

Keywords: plotting

Here is a summary of the optional keyword phrase arguments recognized by most plotting commands. See also topics 'graphs', 'graph_files', 'keywords', 'graph_assign'. You can get a list of all the keywords with their permissible values by typing 'usage(graph_keys)'.

After the list of keywords there are specific descriptions of the use of 'add:T', 'keys:Str' and GRAPHWINDOWS[i] <- structure(...) with structure components named using graphics keywords.

Catalog of graphics keywords

Keyword Phrase	Description
x:xdata *	xdata a REAL vector specifying data for x-axis or NULL
y:ydata *	ydata a REAL vector or matrix specifying data for y-axis or NULL
symbols:symVar	symVar a CHARACTER or integer scalar, vector or matrix specifying plotting symbols; symVar = "###" means use row or column number as plotting symbol
strings:charVar	charVar a CHARACTER scalar or vector specifying labelling information to be plotted at positions specified by x and y.
keys:Str	Str a structure with names matching other graphics keywords; see below.
add:T	Combine the information in the current plotting command with information in LASTPLOT (or a GRAPH variable argument) to create a new graph; see below.
title:"Plot title of your choice"	(up to 75 characters)
xlab:"X-axis label"	(up to 50 characters)
ylab:"Y-axis label"	(up to 20 characters)
xmin:xMinVal or xmin:?	Minimum and maximum values for
xmax:xMaxVal or xmax:?	x-axis and y-axis. Value ?
ymin:yMinVal or ymin:?	means compute from new data and
ymax:yMaxVal or ymax:?	any data in graph being modified.
logx:T	Use log scale for x-axis
logy:T	Use log scale for y-axis
xaxis:F	Do not draw x axis (line y = 0).
yaxis:F	Do not draw y axis (line x = 0).
borders:Word	Controls which sides of the graph borders will be drawn. Word can be "all", "none", "", or a combination of one or more of "B", "b", "L", "l", "T", "t", "R", "r". See topic 'graph_border'.
ticks:Word	Controls which sides of the graph will have tick marks. Word has same form as for 'borders'.

	See topic 'graph_ticks'.
xticks:RealVec yticks:RealVec	Locations for x- or y-axis tick marks and labels. xticks:? and yticks:? mean compute from data. xticks:NULL and yticks:NULL mean no tick marks or labels. See topic 'graph_ticks'
xticklabs:CharVec yticklabs:CharVec	Labels for x-axis or y-axis tick locations. Permissible only with custom tick positions and when length(CharVec) = number of ticks. See topic 'graph_ticks'
xticklen:length yticklen:length	Length of x- or y-axis ticks. Value length >= -1. length < 0 means outside frame; length > 2 means full gridline. Defaults are .5
lines:T	On all plotting commands except boxplot(), stringplot() and addstrings(), connect successive points with lines.
lines:F	Suppress drawing lines on lineplot() and addlines().
linetype:n	On lineplot() and other commands with lines:T sets the line type to n, default is 1 (solid line). n must be an integer -100 < n < 100. n < 0 is the same as abs(n) except no warning is printed when lines are not being drawn; n = 0 is the same as n = -1
thickness:W	On lineplot() and other commands with lines:T sets the line thickness to W times normal thickness, default = 1. W must be between .1 and 10. Has no effect when not feasible as with dumb:T.
impulse:T	On all plotting commands except boxplot(), stringplot() and addstrings(), draw line from x-axis (y=0 line) to points.
show:F	Do not display plot, only save as LASTPLOT but not in GRAPHWINDOWS
keep:F	Do not save plot as LASTPLOT or in GRAPHWINDOWS; only display. The corresponding component of GRAPHWINDOWS is cleared unless show:F is an argument.
dumb:T	Make low resolution plot using printable characters only, suitable for printing on a line printer. Default is taken from option 'dumbplot'; F means high

	resolution.
height:n	Number of lines to be used in "dumb" plot; n >= 12 is required; ignored without dumb:T.
width:n	Width in character positions to be used in "dumb" plot; n >= 30 is required; ignored without dumb:T.
window:n	Draw plot in window n (1 <= n <= 8). (versions with windows only) If n = 0, use window most recently used.
pause:T (versions with windows) pause:F (other versions)	Forces (T) or suppresses (F) a pause after the graph is drawn.
file:FileName	Write PostScript to file Filename
new:T	Clear file FileName before writing
landscape:T	PostScript plot will be rotated so as to fill 8.5" by 11" page.
ps:F	Suppresses PostScript when writing a plot to a file. On Unix/Linux, Tektronix plotting commands are written. On Mac OS 9, a PICT file is written. On other systems, a 'dumb' plot is written.
epsf:T (Mac OS 9 only)	Produces an encapsulated PostScript file. Legal only with file:fileName.
silent:T	Suppress warning messages
screendump:FileName (Mac OS 9 and DOS extended memory versions only)	Save a copy of screen in file FileName in a form other applications may be able to import.
notes:charVec	Adds notes to GRAPH symbol LASTPLOT ; charVec must be a CHARACTER vector or scalar. See topic 'notes'.

* 'x' and 'y' have no special meaning as keywords but have meaning as component names of the value of keyword 'keys' when it is an argument. See below.

Use of keyword phrase 'add:T'

When keyword phrase 'add:T' is an argument to plot(), chplot(), lineplot() or stringplot(), it makes them behave like addpoints(), addchars(), addlines() and addstrings(), respectively. 'add' may not be used without new data, so it is illegal with showplot(). Conversely, use of 'add:F' on addpoints(), addchars(), addlines() or addstrings() makes them behave like plot(), chplot(), lineplot() or stringplot().

Use of keyword phrase 'keys:Str'

The various graphics keyword phrases are usually used as separate arguments to plotting commands. Instead you can specify them by argument keys:Str, where Str is a structure with components with names matching graphics keywords. With this usage, no graphics keyword

```
phrases except keys:Str can be arguments.  In addition to the regular
graphics keyword names such as 'xlab' and 'ticks', you can use names 'x'
and 'y' to specify components of Str which provide data to plot.

This is probably best illustrated by example.  The following lines are
equivalent.

  Cmd> plot(time,d,xmin:0,ymin:0,xlab:"Time",ylab:"Distance")

  Cmd> plot(time,d,keys:structure(xmin:0,ymin:0,xlab:"Time",\
       ylab:"Distance"))

  Cmd> plot(keys:structure(x:time,y:d,xmin:0,ymin:0,xlab:"Time",\
       ylab:"Distance"))

The following is illegal because it mixes use of 'keys' with graphics
keywords 'xlab' and 'ylab' outside the structure.

  Cmd> plot(time,d,keys:structure(xmin:0,ymin:0), xlab:"Time",\
       ylab:"Distance") # xlab & ylab are not in the structure

          Use of keywords with GRAPHWINDOWS[I] <- structure(...)

Many of the graphics keywords can be used to define the components of a
structure being assigned to GRAPHWINDOWS[I].  For example,

  Cmd> GRAPHWINDOWS[3] <- structure(x:time,y:d,lines:T,xlab:"Time",\
       ylab:"Distance")

has the same effect as

  Cmd> lineplot(time,d,window:3,xlab:"Time", ylab:"Distance").

The structure being assigned to GRAPHWINDOWS[I] must not have components
named 'file', 'new', 'landscape', 'ps', 'epsf' or 'screendump'.  If
there are components named 'keep' and/or 'show' they must have value
True.  In windowed versions, the structure should not have components
'dumb', 'height' or 'width'.  If there is a component named 'windows',
its value must be the same as I.  If either component 'x' or 'y' is
NULL, GRAPHWINDOWS[I] is unchanged.

See topics 'GRAPHWINDOWS' and 'graph_assign' for more details
```

2.148 graph_ticks

Usage:

```
Graphics keywords 'xticks', 'yticks', 'xticklen', 'yticklen', 'ticks',
  'xticklabs', 'yticklabs' are used to modify default tick marks in
  graphs, provide labels for them and specify which sides of a plot they
  are drawn.
Values for 'xticks' and 'yticks' are REAL vectors or NULL.
Values for 'xticklen' and 'ytick' length are REAL scalars >= -1.
Values for 'ticks' must be "all", "none" or a combination of letters 'B"
  (bottom), 'L' (left), 'T' (top) and 'R' (right) or their lower case
  counterparts.
Values for 'xticklabs' and 'yticklabs' must be CHARACTER vectors, ?, or
  NULL
```

Keywords: plotting

```
                              Introduction
This topic summarizes how you can customize tick positions, appearance
and labelling on graphs.  See topic 'graphs' for general information on
plotting commands and topic 'graph_keys' for information about keyword
phrases.

By default, plotting commands draw tick marks at automatically computed
"neat" positions.  The default tick length is about half the width of a
character.  All tick marks are labeled in the left or bottom margin,
with the default labels being their locations.

                          Setting tick positions
You can customize the horizontal and vertical positions of tick marks by
keywords 'xticks' and 'yticks' whose values must be REAL vectors or
NULL.  xticks:NULL and yticks:NULL suppress ticks and their labels
entirely.  xticks:? or yticks:? cause the default tick mark positions to
be used.

                           Setting tick lengths
You can control the lengths of the tick marks by keywords 'xticklen' and
'yticklen' whose values should be REAL scalars >= -1.  The default
length correspondes to xticklen:.5 and yticklen:.5.  Values < 0 give
ticks outside the border and 0 values suppress drawing ticks but not
their labels.  Values > 2 cause full gridlines from one side of the plot
to the other to be drawn.

                        Setting borders with ticks
By default, tick marks are drawn on any edge of the graph where a border
line is drawn, with full gridlines drawn if a border is drawn at either
end of the line.  You can modify this behavior using 'ticks:word', where
'word' is a quoted string or character scalar.  If 'word' isn't "all" or
"none", it must contain only the characters 'B' (bottom), 'L' (left),
'T' (top) or 'R' (right), or their lower case counterparts 'b', 'l', 't'
and 'r'.  Value "" is equivalent to "none".

Using 'borders:word' without 'ticks' automatically sets the sides where
ticks are drawn to match where borders are drawn.  See topic
```

```
'graph_border'.

Regardless of the value of 'ticks', tick labels will always be put at
the bottom and left.

                      Setting character tick labels
You can replace numerical labels for ticks with arbitrary labels using
keyword phrases 'xticklabs:charvec' and/or 'yticklabs:charvec', where
charvec is a CHARACTER vector.  This is permissible only in the
following situations (described for the x-axis).

 Keyword phrase 'xticks:realvec' is an argument and length(charvec) =
 length(charvec) = length(realvec).

 You are displaying or adding to a existing plot for which x-axis tick
 positions were supplied and length(charvec) = number of such positions.

 You are labelling the "box number" axis on a box plot and
 length(realvec) = number of boxes.

In particular, 'xticklabs' is ignored if default tick positions are
being used, even when charvec is of the right length.

                                Examples
Examples:
  Cmd> plot(x,y,xticks:vector(1,2,4),yticks:NULL,xticklen:1.5,\
   ticks:"B", xticklabs:vector("Fair", "OK", "Super"))

gives x-axis ticks 1.5 times the width of a character (3 times normal)
at x = 1, 2 and 4 on the bottom border only, giving them special labels,
and suppresses all y-axis ticks and their labels.

  Cmd> plot(x,y,xticklen:3,yticklen:-.5, ticks:"BL")
draws full gridlines perpendicular to the x-axis and default length
ticks along the outside of the left edge of the frame.  With ticks:"L",
ticks:"R", ticks:"LR" or ticks:"none", the gridlines would not be drawn
at all.

  Cmd> showplot(xticks:?,yticks:?)
sets the default tick positions, without altering their length.  If
custom tick labels were set, they are discarded.  Use of 'xticklabs' or
'yticklabs' produces a warning message but has no other effect.

                            Cross references
See also topics 'graphs' and 'graph_keys'.
```

2.149 halfnorm()

Usage:

```
halfnorm(x [,ties:"ignore" or "average" or "minimum"]), x REAL or a
  structure with REAL components.
halfnorm(n:N), integer N > 0.
```

Keywords: transformations, descriptive statistics, ordering

```
                                Usage
halfnorm(x) computes the vector of approximate half normal scores for
the data in the REAL vector x.  This probably makes sense only when the
elements of x are all non-negative, although that is not required.

halfnorm(n:N), N a positive integer, is equivalent to halfnorm(run(N)).

What is computed is equivalent to
   invnor(.5 + .5*(rank(abs(x),ties:"ignore") - .375)/(n + .25))
where n is the number of non-MISSING values.  The value corresponding to
a missing value is MISSING.

The most important use of halfnorm() is probably plot(halfnorm(ss),
sqrt(ss)), where ss is a vector of 1 degree of freedom sums of squares.
This produces a half normal plot of sqrt(ss).

halfnorm(x [keywords]) has the same labels as x, if any.

                            Keyword 'ties'
halfnorm(x,ties:Method), where Method is "ignore", "average", or
"minimum" (or "i", "a", "m") computes invnor(.5 +
.5*(rank(abs(x),ties:Method) - .375)/(n + .25)).  See rank() for a
detailed discussion of the three methods.  It is hard to think of a
situation when you would want to use "minimum" with halfnorm().

                  Matrix or array structure argument
If x is a matrix, the result is a matrix each of whose columns contains
the half normal scores for the corresponding column of x.

If x is an array, halfnorm(x) is an array of the same size and shape
with all the elements with fixed values of subscripts 2, 3, ... defining
a "column" whose half normal scores are computed.  An array with
dimension > 2 is always treated as an array and not as a matrix, even if
there are at most two dimensions greater than 1.

It is also acceptable for x to be a structure, whose non-structure
components are all REAL.  In that case, halfnorm(x) returns a structure
of the same form, each of whose non-structure components is the result
of applying halfnorm() to the corresponding component of x.

                               Example
Example:
  Cmd> x <- vector(.59,8.82,9.46,3.34,3.49) # ranks are 1,4,5,2,3

  Cmd> halfnorm(x)
```

```
  (1)     0.14976      1.0162      1.5588     0.39821     0.67449

                            Cross references
See also rankits().
```

2.150 haslabels()

Usage:

```
   haslabels(x), x a variable that is not NULL.
```

Keywords: general, variables

```
                           Usage with example
haslabels(x) is True if and only if variable x has coordinate labels.

Example;
  Cmd> x <- yourMacro(y)
  Cmd> if (haslabels(y)){
        x <- matrix(x,labels:structure("",getlabels(y,2)))}

                            Cross references
See topics 'labels', getlabels().

haslabels() is implemented as a pre-defined macro.
```

2.151 hasnotes()

Usage:

```
   hasnotes(x), x a variable that is not NULL.
```

Keywords: general, variables

```
                           Usage with example
hasnotes(x) is True if and only if variable x has attached notes.

Example:
  Cmd> x <- yourMacro(y)

  Cmd> if (hasnotes(y)){attachnotes(x, getnotes(y))}

                            Cross references
See topics 'notes', getnotes(), attachnotes(), appendnotes().

hasnotes() is implemented as a pre-defined macro.
```

2.152 hconcat()

Usage:

```
hconcat(a,b,c,... [,labels:structure(rowLabs,colLabs), silent:T]) where
  a, b, c, ... matrices and rowLabs and colLabs are CHARACTER scalars or
  vectors
```

Keywords: combining variables, variables, null variables

```
                                  Usage
hconcat(a,b,c,...) combines matrices a, b, c ... side to side by
concatenating their rows.

All arguments must be of the same type, REAL, LOGICAL, or CHARACTER, and
have the same number of rows m.  The result is a matrix of that type
with m rows and na+nb+nc+... columns, where na, nb, nc, ... are the
number of columns of a, b, c, ... .


An argument that is a vector of length m is considered to be a m by 1
matrix.  In particular, if a is a vector of length m, hconcat(a) is a m
by 1 matrix.

An argument that is an array with only two dimensions not equal to 1 is
considered to be a matrix (see 'matrices').  For example,
  Cmd> hconcat(array(run(6),1,2,3),array(2*run(8),2,1,4))
is equivalent to
  Cmd> hconcat(matrix(run(6),2),matrix(2*run(8),2))

If a is a vector of length n, hconcat(a) is a matrix with n rows and 1
column.

Any argument of type NULL is ignored.  If all arguments are NULL, so is
the result.

                             Keyword 'labels'
hconcat(a,b,...,labels:structure(rowLabs,colLabs) [,silent:T]) uses
CHARACTER scalars or vectors rowLabs and colLabs as row and column
labels for the result.  With silent:T, no warning is printed if labels
are the wrong size.  See topic 'labels' for details.

                             Cross references
See also topics vconcat(), vector(), 'matrices', 'NULL', 'vectors'.
```

2.153 hconj()

Usage:

```
hconj(hx), hx a REAL matrix representing complex data with Hermitian
  symmetry
```

Keywords: time series, complex arithmetic

Usage

hconj(hx) returns the complex conjugate (in packed Hermitian form) of the columns of the vector or matrix hx, considered as complex series with Hermitian symmetry in packed Hermitian form.

Cross references

See also cconj(), hreal(), himag(), creal(), cimag().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for working with complex matrices.

2.154 hdivh()

Usage:

hdivh(hx1 [, hx2]), hx1 and hx2 REAL matrices representing complex data with Hermitian symmetry

Keywords: time series, complex arithmetic

Usage

hdivh(hx1, hx2) computes the element-wise complex ratio of elements in the columns of REAL matrices or vectors hx1 and hx2, considered as complex matrices in packed Hermitian form. The result is also a complex matrix in packed Hermitian form.

Any ratio of the form (0 + 0i)/(0 + 0i) is returned as 0 + 0i. The ratio of a non-zero element of cx1 and 0 + 0i is MISSING + MISSING*i.

When hx1 represents a single complex series (has 1 column), that series is divided by all the series in hx2. Similarly when hx2 represents a single complex series, all the series in hx1 are divided by hx2.

For example, if hx1 is m by 1 and hx2 is m by 3, hdivh(hx1,hx2) is equivalent to hdivh(hconcat(hx1,hx1,hx1),hx2).

hdivh(hx) is equivalent to hdivh(hx,hx) and yields a result with all real parts = 1 and all imaginary parts = 0 (except (0 + 0i)/(0 + 0i) elements).

Cross references

See also hdivhj(), cdivc(), cdivcj(), hprdh(), hprdhj(), cprdc(), cprdcj().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for working with complex matrices.

2.155 hdivhj()

Usage:

```
hdivhj(hx1 [, hx2]), hx1 and hx2 REAL matrices representing complex data
  with Hermitian symmetry
```

Keywords: time series, complex arithmetic

```
                                  Usage
hdivhj(hx1, hx2) computes the element-wise complex ratio of elements
in the columns of hx1 and hconj(hx2), considered as complex matrices in
packed Hermitian form.  The result is also a complex matrix in packed
Hermitian form.

Any ratio of the form (0 + 0i)/(0 + 0i) is returned as 0 + 0i.  The ratio
of a non-zero element of cx1 and 0 + 0i is MISSING + MISSING*i.

When hx1 represents a single complex series (has 1 column), that series
is divided by all the complex conjugates of the series in hx2.
Similarly, when hx2 represents a single complex series, all the series in
hx1 are divided by hconj(hx1).

For example, if hx1 is m by 1 and hx2 is m by 3, hdivhj(hx1,hx2) is
equivalent to hdivhj(hconcat(hx1,hx1,hx1),hx2).

hdivhj(hx) is equivalent to hdivhj(hx,hx).

                            Cross references
See also hdivh(), cdivc(), cdivcj(), hprdh(), hprdhj(), cprdc(),
cprdcj(), hconj().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.156 help()

Usage:

```
help(topic [, allfiles:T])
help(topic1, topic2, ... [, allfiles:F])
help(Topic:Subtopics), Topic the name of a topic, Subtopics a CHARACTER
  scalar or vector naming subtopics of Topic
help(Topic, subtopic:Subtopics)
help(Topic, subtopic:"?")
help(index:helpfilename), CHARACTER scalar helpfilename
help(Pattern) where Pattern has form "start*", "start*end", or "*mid*",
  "start*mid*", ...
help(news [,allfiles:T]), help(news:yymmdd1 [,allfiles:T]) or
  help(news:vector(yymmdd1, yymmdd2) [,allfiles:T]), where yymmdd1 and
  yymmdd2 are integers like 990103, 19990103, 000717 or 20000727
help(key:KeyName [keywords]), where KeyName is a CHARACTER scalar or
  "?" (scans only current help file)
help(topic [,subtopic:Subtopics] , file:fileName or orig:T or alt:T),
  CHARACTER scalar fileName (scans only one help file)
```

Keywords: general

```
                                  Usage
help() with no argument will print a short message giving some of the
help() and gethelp() options.

help(topicname) and help("topicname") lists help information on the
named topic.

help(topicname, subtopic:"subtopic_name") does the same, but lists only
the information in the named subtopic.  If topicname has no more than 10
characters, help(topicname:"subtopic_name") does the same.

You can print several subtopics by, for example, help(topicname,
subtopic:vector("usage", "examples")).

help() returns an "invisible" LOGICAL scalar whose value is True only
when at least one topic requested was found.  The value may be assigned
or tested but will not be printed automatically.  See topic
'variables:"invisible"'.

Files named in pre-defined CHARACTER vector HELPFILES are searched until
the topic is found.  This is done by repeated calls to gethelp(), which
scans only one file at a time.

By default, when help is requested on only one topic, the search stops
the first time the topic is found in a file.  To force scanning all
files, use keyword phrase 'allfiles:T'.

help(topic1, topic2, ...) requests help on several topics at once.  You
cannot use keyword 'subtopic' when there is more than one topic, but any
argument can be of the form topicname:"subtopic_name".

In this case, by default, all the help files are scanned, which can take
```

a noticable time. If you know all the topics are in a single file, you can save a little waiting time by using 'allfiles:F' as an argument.

Index topics

help(index:helpfilename), where helpfilename is a quoted string or CHARACTER scalar identifying one of the help files named in HELPFILES prints the index topic in that file. In particular,

```
  Cmd> help(index:HELPFILES[3])
```

prints the index of the file whose name is in the third element of HELPFILES.

If helpfilename doesn't exactly match any name in HELPFILES (ignoring case), various matches are attempted using the wild card characters '*' and '?'. Thus help(index:"design.hlp") and help(index:"design") both print the annotated index of design.hlp.

No other arguments or keywords can be used with 'index'.

Examples

Examples:

```
  Cmd> help(anova); help(macros); help("break")
  Cmd> help(break); help(transformations) #now work; previously didn't
  Cmd> help(regress, usage:T) # same as usage(regress)
  Cmd> help(stepsetup, subtopic:"example")# or help(stepsetup:"example")
  Cmd> help(tsplot:vector("usage","example"))
  Cmd> if (!help(foo,silent:T)){print("No help on topic foo")}
  Cmd> help(index:"math")
```

Resetting current help file

You can reset the current help file to any particular file in HELPFILES by

```
  Cmd> help(file:HELPFILES[I]) # 1 <= I <= length(HELPFILES)
```

On other usages of help(), the current help file is either not changed or is changed to the last file searched.

When no help is found or when there are several topics without all:F, the current help file is not changed.

When there are several topics with all:F or a single topic, and at least one topic is found, the file containing the found topic becomes the current help file.

Regardless of the number of topics, when reset:T is an argument, the current help file is not changed.

Topic 'news'

help(news) lists in reverse chronological order news items about MacAnova starting with the most recent entry back for three months.

help(news:vector(Date1,Date2) [,scrollback:T] [,all:T]), where Date1 and Date2 are numbers of the form yymmdd or yyyymmdd, lists in reverse chronological order news items about MacAnova development dated between

```
Date1 and Date2.  For example, help(news:vector(991201,0000131)) and
help(news:vector(19991201,20001230)) list all news items dated in
December, 1999 or January, 2000

help(news:Date) lists all news items on or after Date.  For example,
help(news:000101) and help(news:200000100) list all news items on or
after January 1, 2000.

help(news:0) lists all available news items.  This will produce many of
lines of output and is not recommended.

You can use 'scrollback:T' with all these ways to read news topics.

By default, when looking for news topics, only file HELPFILES[1]
(usually "macanova.hlp") is scanned.  If you include 'all:T' as an
argument, all files in HELPFILES are searched for news items.

                          Variable HELPFILES
HELPFILES is a CHARACTER vector initialized when MacAnova starts up to
contain all the standard help files except "userfun.hlp".

help() uses HELPFILES as a "circular" list.  If the current help file is
in the list, than the search starts with that file and wraps back to
HELPFILES[1] if necessary.  If the current help file is not in that
list, then the current help file is scanned, followed by the files named
in HELPFILES.

                          Adding help files
You can use macro addhelpfile() to add a file name to HELPFILES.  For
example, addhelpfile("mymacros.hlp") and addhelpfile("mymacros.hlp",T)
add "mymacros.hlp" at the start and end of HELPFILES, respectively.

                        File related keywords
You can use gethelp() keywords 'file', 'orig', 'alt' and 'key', but in
that case only the current help file is scanned, not the files in
HELPFILES.

                            History note
help() is implemented as a macro which invokes gethelp().

NOTE: Prior to Version 4.12, gethelp() was named help() and there was no
macro named help().

                          Cross references
See gethelp(), getusage() and usage().
```

2.157 hft()

Usage:

```
hft(hx [,divbyT:T]), hx a REAL matrix considered as complex in Hermitian
  form
```

Keywords: time series, complex arithmetic

```
                                  Usage
hft(hx) where hx is a REAL vector or matrix, computes the real discrete
Fourier transform of each column of hx, considered as a complex series
with Hermitian symmetry in packed Hermitian form.

Any MISSING values in hx are replaced by 0 in computing the result and a
warning message is printed.

hft(hx,divbyt:T) does the same, except the result is divided by the
number of rows of hx.

                              Inverse transform
hconj(rft(rx,divbyt:T)) is the inverse of hft() in the sense that hx and
hconj(rft(hft(hx),divbyt:T)) are equal except for rounding error.

                            Limitation on length
The largest prime factor of nrows(hx) must not exceed 29.  You can use
primefactors() to find the maximum factor of nrows(hx) and goodfactors()
to find a length >= nrows(hx) which has no prime factors > 29.  In
addition, the product of all the "unpaired" prime factors can't be too
large.  For example N = 3*5*7*11*13*17*M^2 = 255255*M^2, where M is an
integer, breaks the algorithm and hence is not allowed.

                              Cross references
See also cft(), rft(), hconj(), primefactors(), goodfactors().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.158 himag()

Usage:

```
himag(hx), hx a REAL matrix representing complex data with Hermitian
  symmetry
```

Keywords: time series, complex arithmetic

```
                                  Usage
himag(hx) computes the imaginary part of the packed Hermitian matrix hx.
For examplem, himag(vector(1,2,3,4,5)) is vector(0,5,4,-4,-5) and
himag(vector(1,2,3,4,5,6)) is vector(0,6,5,0,-5,-6).

See also hconj(), cconj(), hreal(), creal(), cimag().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
```

working with complex matrices.

2.159 hist()

Usage:

```
hist(x [, nbars] [,keyword phrases]), REAL vector x, integer nbars >= 2
hist(x, vector(anchor,width) [,keyword phrases]), anchor REAL scalar,
  width > 0 scalar
hist(x, edges [,keyword phrases]), edges REAL vector with increasing
  elements
hist(x .... , keys:structure(keyword phrases))
Keyword phrases are relfreq:T, freq:T, leftendin:T, outsideok:T, draw:T,
  save:T plus most graphics keywords
```

Keywords: plotting, descriptive statistics

```
                                  Usage
hist(x, nbars) draws a histogram of the data in REAL vector x using with
nbars equal width bars which include all data.  The bar edges are not
"neat".  For example, 1,1.5,2,2.5,3.0, ... are "neat", 2.71,3.82, 4.93,
6.04, ... are not "neat".

Bar heights are in the so called "density scale" with height = (M/N)/W,
where M is the number of values in a bar with width W and N is the
number of non-MISSING values in x.  This choice makes the total area of
the bars = 1.

A value x is included in bar i when L[i] < x <= R[i], where L and R are
vectors of the left and right edges of the bars.

                          Keyword 'leftendin'
hist(x, nbars, leftendin:T) does the same, except a value x is included
in bar i when L[i] <= x < R[i].  'leftendin:T' can be used with any
variant of hist() arguments and any other keywords.

                      keywords 'freq' and 'relfreq'
hist(x, nbars, freq:T) and hist(x, nbars, relfreq:T) do the same except
that bar heights are frequencies (M) or relative frequencies (M/N) with
no adjustment for bar width.  'freq:T' and 'relfreq:T' can be used with
any variant of hist() arguments.

                        Default number of classes
hist(x [,keyword phrases]) does the same using floor(log2(N)) + 1 bars.

                          Anchored boundaries
hist(x, vector(anchor, width) [,keyword phrases]) does the same, except
the edges of the bars are of the form anchor + j*width, with anchor +
min(j)*width < min(x) and anchor + min(j) > max(x), with the lowest and
highest bar edges chosen to include all the data.

                        Specified boundaries
```

```
hist(x, Edges [,keyword phrases]) draws a histogram with bar boundaries
from REAL vector Edges with length(Edges) > 2 and satisfying Edges[i] <
Edges[i+1].  The number of bars is nbars = length(Edges) - 1.  A warning
message is printed when bar widths are not all equal and 'relfreq:T' or
'freq:T' is an argument.

                                Outsideok
hist(x, Edges, outsideok:T) does the same, except it is not an error
when some extreme values are outside the bars defined by Edges.  When
values are outside, a warning message is printed.  Without outsideok:T
this is an error.  A value y is outside the bars when y < Edges[1] or y
> Edges[nbars+1].  Without 'leftendin:T', Edges[1] is outside; with
'leftendin:T', Edges[nbars+1] is outside.

All of the usual plotting related keywords, including 'dumb', 'xlab',
'ylab', and 'title', may be used with hist().  See also topics 'graphs',
'graph_keys', 'graph_borders' and 'graph_files'.

                       Keywords 'save' and 'draw'
result <- hist(x ... , save:T [, draw:T] [,graphics keywords]) returns
structure(x:xvals, y:yvals [,graphics keywords], lines:T, yaxis:F) as
value.  REAL vectors xvals and yvals are such that lineplot(xvals,yvals,
yaxis:F [,graphics keywords]) or lineplot(keys:results) draws the
histogram.  Nothing is drawn unless 'draw:T' is also an argument.

                             Keyword 'keys'
An alternate way to specify keyword values is to create a structure
keyValues of keyword values and use 'keys:keyValues' as the only keyword
phrase argument.  For example

  Cmd> keyValues <- structure(xlab:"Bone length",relfreq:T,\
        title:"Bone histogram", ylab:"Relative frequency",save:T)

  Cmd> stuff <- hist(bones,vector(0,.25),keys:keyValues)

does the same as
  Cmd> stuff <- hist(bones,vector(0,.25),xlab:"Bone length",relfreq:T,\
        title:"Bone histogram", ylab:"Relative frequency",save:T)

                            Cross references
See also topic panelhist().
```

2.160 hpolar()

Usage:

```
   hpolar(hx [,unwind:F or crit:val]), hx a REAL matrix representing
     complex data with Hermitian symmetry, val a REAL scalar,
     0.5 < val <= 1
```

Keywords: time series, complex arithmetic

```
                                  Usage
hpolar(hx) computes the polar form of the packed Hermitian matrix hx,
storing it in pseudo packed Hermitian form, with the amplitude or
absolute value as the real part and the phase as imaginary part.  Thus
hreal(hpolar(hx)) and himag(hpolar(hx)) return REAL matrices whose
columns are the amplitudes and phases of the complex series represented
by the columns of hx.

                               Angle units
The value of the computed phase is in radians, degrees or cycles
depending on the value of option 'angles'.  See subtopic
'options:"angles"'.  By default the phase is "unwound" so as to minimize
discontinuities arising from wrap-around.

                      Keywords 'crit' and 'unwind'
hpolar(hx,crit:Val), where .5 <= Val < 1 changes the criterion
controlling "unwinding".  The default is .75.  See unwind() for details.

hpolar(hx,unwind:F) suppresses the unwinding.

                            Cross references
See also cpolar(), crect(), hrect().

See topic 'complex' for information on complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.161 hprdh()

Usage:

```
hprdh(hx1 [, hx2]), hx1 and hx2 REAL matrices representing complex data
  with Hermitian symmetry
```

Keywords: time series, complex arithmetic

```
                                  Usage
hprdh(hx1, hx2) computes the element-wise complex ratios of elements in
the columns of REAL matrices or vectors hx1 and hx2, considered as
complex matrices in packed Hermitian form.  The result is also a complex
matrix in packed Hermitian form.

Any ratio of the form (0 + 0i)/(0 + 0i) is computed to be 0.

If hx1 or hx2 represents a single complex series (has 1 column), that
series is multiplied by all the series in the other arguments.  For
example, if hx1 is m by 1 and hx2 is m by 3, hprdh(hx1,hx2) is
equivalent to hprdh(hconcat(hx1,hx1,hx1),hx2).

hprdh(hx) is equivalent to hprdh(hx,hx).
```

Cross references
See also hprdhj(), cprdc(), cprdcj(), hdivh(), hdivhj(), cdivc(), cdivcj().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for working with complex matrices.

2.162 hprdhj()

Usage:

```
hprdhj(hx1 [, hx2]), hx1 and hx2 REAL matrices representing complex data
  with Hermitian symmetry
```

Keywords: time series, complex arithmetic

Usage
hprdhj(hx1, hx2) computes the element-wise complex product of elements in the columns of hx1 and hconj(hx2), considered as complex matrices in packed Hermitian form. The result is also a complex matrix in packed Hermitian form.

If hx1 or hx2 represents a single complex series (has 1 column), that series is multiplied by all the series in the other arguments. For example, if hx1 is m by 1 and hx2 is m by 3, hprdhj(hx1,hx2) is equivalent to hprdhj(hconcat(hx1,hx1,hx1),hx2).

hprdhj(hx) is equivalent to hprdhj(hx,hx) and produces an packed Hermitian output matrix with the squared moduli of the elements of hx, considered as complex numbers, in the real part of the result, with zeros in the imaginary part.

Cross references
See also hprdh(), cprdc(), cprdcj(), hdivh(), hdivhj(), cdivc(),

cdivcj(), hconj().
See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for working with complex matrices.

2.163 hreal()

Usage:

```
hreal(hx), hx a REAL matrix representing complex data with Hermitian
  symmetry
```

Keywords: time series, complex arithmetic

```
                                Usage
hreal(hx) computes the real part of the packed Hermitian matrix hx.  For
example, hreal(vector(1,2,3,4,5)) returns vector(1,2,3,3,2) and
hreal(vector(1,2,3,4,5,6)) returns vector(1,2,3 4,3,2).

                            Cross references
See also hconj(), cconj(), himag(), creal(), creal().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.164 hrect()

Usage:

```
hrect(hx), hx a REAL matrix representing complex data with Hermitian
  symmetry
```

Keywords: time series, complex arithmetic

```
                                Usage
hrect(hx) is the inverse operation to hpolar().  Matrix hx is assumed to
contain the polar form of a packed Hermitian series, with amplitudes or
absolute values in the real part and phases in the imaginary part.  The
result contains the real and imaginary parts of that series in packed
Hermitian form.  See topic 'complex' for discussion of complex matrices
in MacAnova.

The phases are assumed to be in units of radians, degrees or cycles
depending on the value of option 'angles'.  See subtopic
'options:"angles"'.

                            Cross references
See also cpolar(), crect(), hpolar().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.165 htoc()

Usage:

```
htoc(hx), hx a REAL matrix representing complex data with Hermitian
  symmetry
```

Keywords: time series, complex arithmetic

Usage

htoc(hx) returns the fully complex equivalent of the matrix hx, considering its columns as complex series with Hermitian symmetry. If hx is m by n, htoc(hx) is m by 2*n, the real and imaginary parts of column i of hx in columns 2*i-1 and 2*i of the result.

Cross references

See also ctoh(), cconj(), hconj(), hreal(), himag(), creal(), cimag().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for working with complex matrices.

2.166 hypot()

Usage:

hypot(x,y), x and y REAL of the same size and shape, or structures with matching REAL components

Keywords: transformations

Usage

hypot(x,y) is mathematically equivalent to sqrt(x^2 + y^2) but can provide answer when either (i) x^2 + y^2 is too big to be represented in the computer; or (ii) x^2 + y^2 is smaller than the smallest non-zero value representable in the computer and thus evaluates to 0. x and y must be REAL vectors, matrices, or arrays with the same dimensions.

Structure arguments

hypot(x,y) is also defined when x and y are structures of the same shape. The result is a structure whose i-th component is hypot(xi,yi), where xi and yi are the i-th components of x and y.

Character arguments

hypot(x,y) can be used when both x and y are CHARACTER variables with matching dimensions. In that case the result is a CHARACTER variable describing the transformation of the arguments. For example, hypot(vector("X1","X2"),vector("Y1","Y2")) returns vector("hypot(X1,Y1)", "hypot(X2,Y2)"). This feature may be useful in creating new labels for a transformed variable.

Cross references

See also topics atan(), 'transformations', 'structures', 'labels'.

2.167 if

Usage:

```
if (Logical){cmd;...}
if (Logical){cmd1;...} else {cmd2;...}
if (Logical1){cmd1;...) elseif (Logical2){cmd2;...} [else {cmd3;...}]
```

Keywords: syntax, control

```
                              Usage of 'if'
if(Logical){Statement} allows conditional execution of Statement, an
arbitrarily complex statement or compound statement.  Statement will be
executed if and only if Logical has the value True.  Logical should be a
scalar LOGICAL variable or expression.  A simple example would be
  Cmd> if(min(x) > 0){logx <- log(x);;}

                         Usage of 'if ... else'
if(Logical){Statement1} else {Statement2} results in Statement1 being
executed when Logical is True, and Statement2 being executed when
Logical is False.  An example would be
  Cmd> if(min(x) > 0){logx <- log(x);;}else{
         error("Illegal non-positive values")}

                     Usage of 'if... elseif... else'
if(Logical1){Statement1} elseif(Logical2){Statement2} else {Statement3}
executes Statement1 when Logical1 is True, executes Statement2 when
Logical1 is False and Logical2 is True, and executes Statement3 when
both Logical1 and Logical2 are False.  An example would be
  Cmd> if(min(x) > 0){logx < -log(x);;}elseif(max(x) < 0){
            logx <- -log(-x);;}else{
            error("Not all positive and not all negative")}

There can be additional elseif(Logical){Statement} constructs before the
concluding 'else{...}', and the concluding 'else{...}' can be omitted.
The first Statement for which the corresponding Logical is True is
executed.

                                  Value
The value of any of these constructs starting with 'if' is the value of
whichever '{Statement}' is actually executed.  If no value is wanted, it
is good practice to terminate each Statement with ';;' so the value will
be NULL, as in the examples above.  If all the Logicals are False and
there is no concluding 'else{...}', the value is NULL.  For example,
if(2 > 3){ ...}elseif(5 < 4){...} has value NULL.  See below for
examples of how you can the fact that a conditional statement has a
value.  See also topic 'NULL'.

Once a Logical has been found to be True, any subsequent Logicals are
not evaluated and are not even checked for syntactical correctness.

                                 Examples
  Cmd> if(ismatrix(x)){xinv <- solve(x);;}else {error("not matrix");;}
  Cmd> a <- if(x < 3){1} else {2}# a <- 1 if x < 3 and a <- 2 otherwise
  Cmd> b <- if(x > 0){1} elseif(x < 0){-1} else{0} # b is 1, -1, or 0
```

```
                          Syntactic restriction
Note: Each '{' following 'if(...)', 'elseif(...)' or 'else' must be on
the same line as the preceding 'if', 'elseif' or 'else'; 'elseif' and
'else' must be on the same line as the preceding '}'.  As usual,
however, you can terminate the line with '\', and continue on the next
line as in the following:
  Cmd> if(x<0)\
         {y <- 1;;}\
       else\
         {y <- 2;;}
```

2.168 inforead()

Usage:

```
   inforead(fileName,Name [,quiet:F, echo:T or F, silent:T, notfoundok:T,\
     nofileok:T,badkeyok:T]), fileName and Name CHARACTER scalars; FileName
     can also be CONSOLE or be string:charVal where charVal is a CHARACTER
     scalar or vector.
```

Keywords: input, files, character variables

```
                                    Usage
inforead(FileName,Name) returns the comments (lines starting with ')')
following the header line of data set or macro Name on file FileName.
They are returned, with the leading ')' stripped off, as a CHARACTER
scalar which can be printed.

The contents of the data set or macro are ignored and there is no
checking as to whether the header line is in correct format.  The header
lines are not echoed unless 'quiet:F' is an argument; see below.

It is an error if the file cannot be read or if the named data set or
macro is not found, but see keywords 'notfoundok' and 'nofileok' below.

Name must be a quoted string or CHARACTER scalar and FileName normally
has a similar form, but see keyword 'string' below.

                                File name ""
In versions with windows, if FileName is the null string "", you will
be able to select the file using a dialog box.

inforead(FileName) does the same for the first data set or macro on the
file, assuming that the first non-empty line is the header line of a
data set or macro.

                               Keyword 'file'
inforead(file:FileName [,Name]) is equivalent to inforead(FileName
[,Name]).

See below for the usage inforead(string:charVec [,Name]).
```

Example

Example:

```
  Cmd> haldinfo <- inforead("macanova.dat","halddata"); print(haldinfo)
```

reads the comments describing data set halddata on file macanova.dat into variable haldinfo and then prints haldinfo.

Commands save() and asciisave() would save haldinfo in a file along with the rest of your workspace. When you later use restore() to recover the workspace, print(haldinfo) displays information about the data without having to refer to the original data file.

Keywords

Keywords 'quiet', 'silent', 'notfoundok', 'nofileok' and 'badkeyok' modify the behavior of inforead(). 'echo' is recognized but ignored.

Keyword phrase	Meaning
quiet:T	Header and descriptive comments will not be printed (the default for inforead())
quiet:F	Header and descriptive comments will be printed in addition being returned as value.
silent:T	Only error messages will be printed; incompatible with quiet:F or echo:T
notfoundok:T	Failure to find the macro or data set is not considered an error so NULL is returned and no error message is printed
nofileok:T	Failure to open the file is not considered an error so NULL is returned and no error message is printed
badkeyok:T	Unrecognized or duplicate keywords are silently ignored.

Keywords notfoundok and nofileok are designed to be helpful in a macro. You can check the returned value using isnull() and take special action if the isnull() returns True. See isnull().

Keyword 'string'

inforead(string:CharVar) where CharVec is a CHARACTER scalar or vector, does not read from a file. Instead, it "reads" CharVar as if each element were a line (or several lines if there are embedded end-of-line characters) read from a file.

The first element or line of CharVar must be a header line for a data set or a macro. In particular, info <- inforead(string:CLIPBOARD) would read the header information of the first variable on a replica of a data file in the special variable CLIPBOARD. In windowed versions, this would be taken from the Clipboard. See topic 'CLIPBOARD'.

Keywords 'string' and 'file'

If either keyword 'file' or 'string' is used, they can appear in any position in the argument list, as can setName which must be the only non-keyword argument. For example, inforead(quiet:T,"mymacro", file:"myfile.dat") is equivalent to inforead("myfile.dat","mymacro",

```
quiet:T).

                            Cross references
See topics 'matread_file' and 'macro_files' for information on the
format of files readable by inforead().

See also matread(), macroread(), read(), save(), asciisave().
```

2.169 interrupt

Keywords: general, syntax, control

```
                                Description
To stop the execution of a command after it has been initiated by Return
or Enter, press the interrupt key, defined for specific systems as
follows:
   System        Interrupt Key
   ------        -------------
   Carapace      Interrupt on the File menu or the Interrupt button
   DOS           Ctrl+C
   Unix/Linux    Ctrl+C

You can also press the interrupt key to stop a large amount of output
being written to the screen or window.

In the windowed versions, you may have to wait a few seconds for the
interruption to happen.
```

2.170 invbeta()

Usage:

```
   invbeta(P, alpha, beta [,upper:T or lower:F]), P, alpha and beta REAL,
     elements of P between 0 and 1, those of alpha and beta > 0
```

Keywords: probabilities, random numbers

```
                                   Usage
invbeta(p,a,b) computes the pth quantile (100*p percent point) of the
beta distribution with parameters a and b.

The elements of p must be between 0 and 1, inclusive, and the elements
of a and b must be positive REAL numbers.

If p, a, and b are not all scalars (single numbers), all non-scalar
arguments must have must have the same size and shape and any scalar
arguments are used to compute all the elements of the result.

invbeta(p,a,b,upper:T) and invbeta(p,a,b,lower:F) compute an upper tail
quantile mathematically equivalent to invbeta(1 - p,a,b), but more
```

```
accurate when p is very close to 1.

invbeta() is the inverse of cumbeta().

                    Generating beta random variables
invbeta(runi(n),a,b) will generate a random sample of size n from a beta
distribution .

                      Binomial confidence interval
You can use invbeta() to compute an "exact" confidence for a probability
p based on an observed value x_obs of a binomial random variable with n
trials and P(success) = p.

  Cmd> n <- 19; x_obs <- 11; alpha <- .05 # 95% confidence

  Cmd> p_l <- invbeta(alpha/2,x_obs,n - x_obs + 1)

  Cmd> p_u <- invbeta(alpha/2,x_obs + 1,n - x_obs,upper:T)

  Cmd> vector(p_l,p_u) # exact confidence limits
  (1)       0.335     0.79748

  Cmd> vector(cumbin(x_obs,n,p_u),cumbin(x_obs,n,p_l,upper:T)) #check
  (1)       0.025       0.025

                                 Example
  Cmd> invbeta(.975,3,4) # P(x <= .77722) = .975
  (1)      0.77722

  Cmd> invbeta(.025,3,4,upper:T) # P(x >= .77722) = .025
  (1)      0.77722

                            Cross references
See also cumbeta(), cumbin(), runi().
```

2.171 invchi()

Usage:

```
   invchi(P, df [,noncen, epsilon:eps] [,upper:T or lower:F]), P, df and
     noncen REAL, elements of P between 0 and 1, elements df > 0, elements
     of noncen >= 0, eps > 0 small.
```

Keywords: probabilities, random numbers, confidence intervals

```
                                  Usage
invchi(p,df) computes the pth quantile (100*p percent point, critical
value) of the chi squared distribution with df degrees of freedom.

invchi(p,df,Noncen [, epsilon:eps]) computes the pth quantile of
non-central chi-squared with non-centrality parameter Noncen.  The
accuracy of the inverse is controled by eps which has default value
```

1e-10.

The elements of p must be between 0 and 1 and the elements of df must be positive but need not be integers. If present, the elements of Noncen must be non-negative.

Any of p, df or Noncen that are not scalars (single numbers) must be the same size and shape. Any argument that is a scalar is used to compute all elements of the result.

invchi(p,df [,Noncen], upper:T) and invchi(p,df [,Noncen], lower:F) compute an upper tail quantile mathematically equivalent to invchi(1 - p, df [,Noncen]). It may be more accurate for p very close to 1.

invchi() is the inverse of cumchi().

Use in tests and confidence limits

If alpha is small, invchi(alpha,df,upper:T) or invchi(1-alpha,df) is the critical value for a chi-squared test of significance level alpha.

If Ssq is the sample variance from a normally distributed random sample of size n, then (n-1)*Ssq/invchi(vector(1-alpha/2, alpha/2),n-1) is a 1-alpha confidence interval for the population variance.

Generating chi squared random variables

invchi(runi(n),df [,Noncen]) will generate a random sample of size n from a possibly non-central chi-squared distribution .

Cross references

See also cumchi(), runi().

2.172 invdunnett()

Usage:

```
invdunnett(P, ngroup, errorDf [,groupSizes][,onesided:T][,epsilon:eps]
  [,upper:T or lower:F]) P REAL with elements between 0 and 1, elements
  of ngroup integers >= 2, elements of errorDf >= 1, elements of
  groupsizes >= 0, eps > 0, default = .00001.
```

Keywords: probabilities, comparisons

Usage

invdunnett(P, K, Df) computes Pth quantile (probability point, critical value) of Tmax, where Tmax is the maximum of abs(t21), abs(t31), ..., abs(tK1), where t21, t31, ..., tK1 are K-1 t-statistics of the form tI1 = (xbarI-xbar1)/ stderr(xbarI-xbar1), I = 2,...,K.

xbar1, xbar2, ..., xbarK are the means of independent normal random samples of the same size with identical population means and variances, and the standard errors are computed using an independent estimate of error variance with Df degrees of freedom. When K = 2 the value is the

same as invstu((1+P)/2, Df). See invstu().

See below for computing quantiles when the sample sizes differ.

P, K and Df must be REAL. The elements of P must be between 0 and 1. The elements of K must be integers >= 2, and the elements of Df must be >= 1, not necessarily integers.

Any of the arguments P, K or Df that are not scalars must all be vectors, matrices or arrays of the same size and shape; the value has the same size and shape.

invdunnett(P, K, Df, upper:T) and invdunnett(P, K, Df, lower:F) compute the upper tail quantile invdunnett(1-P, K, Df).

Keyword 'onesided'

invdunnett(P, K, Df, onesided:T [,upper:T]) computes the quantiles for Tmax, where Tmax is now the maximum of t21, t31, ..., tK1, not of their absolute values. When K = 2 the value is the same as invstu(P,Df [,upper:T]).

Keyword 'epsilon'

invdunnett(P, K, Df [, onesided:T], epsilon:eps), where eps is a small positive number (default .00001) which controls the accuracy to which the quantile is computed. Specifically the logit of the probability corresponding to the computed quantile should be no farther than eps from the true logit of P (logit(P) = log(P) - log(1-P). Since invdunnett() uses the algorithm underlying cumdunnett() configured so as to compute probabilities to within .00001, eps should not be smaller than the default.

Use in multiple comparisons

invdunnett() is primarily used to compute critical values for a multiple comparisons procedure due to C. W. Dunnett wherein a control group (group 1) is compared to K-1 other treatment groups using K-1 t-tests.

See cumdunnett() for computing P values for the Dunnett test.

Example:

```
  Cmd> invdunnett(.05, 5, 5*8 - 5,upper:T)
```

computes the two-sided critical value for the Dunnett test with significance level alpha = 0.05 for a completely randomized design with 5 groups, all with sample size 8.

Unequal group sizes

invdunnett(x, K, Df, groupSizes [,onesided:T] [,upper:T]) computes quantiles for Tmax, with REAL argument groupSizes specifying the sample sizes.

In the simplest usage, groupSizes is a vector (ndims(groupSizes) = 1), with elements >= 0. If groupSizes is a matrix or array (ndims(groupSizes) > 1), it is treated as if it were a vector, matrix or array, with one less dimension, each of whose elements is a vector with

length = last dimension of groupSizes. The first ndims(groupSizes) - 1 dimensions of groupSizes must match the dimensions of any of x, K, or DF which is not a scalar. In particular, a m by 1 matrix, which is treated as a vector of length m by most MacAnova functions, is interpreted by invdunnett() as a set of m vectors of length 1.

In computing an element of the result based on a vector of group sizes (either all of groupSizes when it is a vector, or a row or "slice" of groupSizes when ndims(groupSizes) > 1), invdunnett() uses up to k of the non-zero leading values in the vector, where k is the corresponding element of K. If there are fewer than k non-zero values, the last one is replicated as many times as needed. It is an error to have a zero value followed by a nonzero value or to have all values zero.

If there is only 1 non-zero value in a row or "slice" of groupSize, the replication of this element means the group sizes are assumed to be equal. In particular, this is the interpretation when groupSizes is a scalar or a m by 1 matrix.

Examples

```
Cmd> invdunnett(.05, 4, 12 - 4, vector(6,2,2,2), upper:T)
```

computes the 5% critical value for a completely randomized design with 4 groups and sample sizes 6, 2, 2 and 2.

```
Cmd> invdunnett(.05, vector(3,4), vector(12 - 3, 12 - 4),\
      matrix(vector(6,3,3,0, 6,2,2,2),4)', upper:T)
```

computes 5% critical values for two completely randomized designs, one with 3 groups and sample sizes 6, 3, and 3, the other with 4 groups with sample sizes 6, 2, 2, and 2. Because trailing values in the rows of groupSizes are replicated, matrix(vector(6,3, 6,2),2)' would be an equivalent way to specify the group sizes.

```
Cmd> invdunnett(.01, 4, 12 - 4, 3, upper:T)
```

computes the same result as invdunnett(.01, 4, 12 - 4, upper:T), because groupSizes is a scalar.

Ratios of group sizes

Only the ratios of non-zero elements of groupSizes are relevant. For example, for 5 groups (K=5), the following groupSizes are equivalent: vector(5,4,3,3,3), vector(5,4,3), vector(5,4,3,0,0), vector(10,8,6). vector(5,4,3,0,3) and vector(5,4,3,0,0,1) would be errors because a non-zero value follows a zero.

Caution: invdunnett() is very computation intensive. If you do not have a fairly fast computer, it may be unacceptably slow. On one Macintosh 68000 computer with no math coprocessor, a single value took over 2300 seconds to compute. Until you know how long it will take on your computer, don't compute more than one value at a time. Using a somewhat larger value for epsilon, for example, epsilon:.0001 or epsilon:.0005, may speed up the calculation at the cost of some loss of accuracy.

```
                          Cross references
See also cumdunnett(), invstudrng().
```

2.173 invF()

Usage:

```
   invF(P, df1, df2 [,upper:T or lower:F]), P, df1 and df2 REAL, elements
     of P between 0 and 1, those of df1 and df2 > 0
```

Keywords: probabilities, random numbers, confidence intervals

```
                                  Usage
invF(p,df1,df2) computes the pth quantile (probability point, critical
value) of the F distribution with df1 and df2 degrees of freedom.

The elements of p must be between 0 and 1 and the elements of df1 and
df2 must be positive REAL numbers (not necessarily integers).

If p, df1, and df2 are not all scalars (single numbers), all non-scalar
arguments must have the same size and shape.  Any scalar arguments are
used to compute all elements of the result.

invF(p,df1,df2,upper:T) and invF(p,df1,df2,lower:F) compute the pth
upper tail quantile.  The result is mathematically equivalent to invF(1
- p, df1,df2) but may be more accurate for small p.

invF() is the inverse of cumF().

                       Use in confidence limits
If S1sq and S2sq are sample variances from independent normal random
samples of sizes n1 and n2,, you can compute a 1 - alpha confidence
interval for the variance Var1/Var2 as
  (S1sq/S2sq)/invF(vector(1-alpha/2, alpha/2), n1-1, n2-1).

                     Generating F random variables
invF(runi(n), df1, df2) will generate a random sample of size n from a F
distribution.

                          Cross references
See also cumF(), runi().
```

2.174 invgamma()

Usage:

```
   invgamma(P, alpha [,upper:T or lower:F]), P and alpha REAL, elements of
     P between 0 and 1, those of alpha > 0
```

Keywords: probabilities, random numbers

```
                                  Usage
invgamma(p,alpha) computes the pth quantile (100*p percent point) of the
gamma distribution with shape parameter alpha.  Its principal use is to
compute critical values for test statistics with a gamma distribution
but may also be used to compute exact confidence intervals for a Poisson
mean.

The elements of p must be between 0 and 1; the elements of alpha must be
positive but need not be integers.

If neither p nor alpha is a scalar (single number), they must be the
same size and shape.  If just one argument is a scalar, it is used to
compute all the elements of the result.

invgamma(p,alpha,upper:T) and invgamma(p,alpha,lower:F) compute the pth
upper tail quantile.  The result is mathematically equivalent to
invgamma(1 - p, alpha) but may be more accurate for small p.

invgamma() is the inverse of cumgamma().

2*invgamma(p,df/2 [,upper:T]) is equivalent to invchi(p,df [,upper:T]).

                    Generating gamma random variables
mu*invgamma(runi(n),alpha)/alpha will generate a random sample of size n
from a gamma distribution with mean mu and shape parameter alpha.

                       Poisson confidence interval
You can use invgamma() to compute an "exact" confidence interval for mu
based on an observed value x_obs of a Poisson random variable with mean
mu.
  Cmd> x_obs <- 11 # observed value of x

  Cmd> mu_l <- invgamma(.025,x_obs) # lower 95% limit

  Cmd> mu_u <- invgamma(.025,x_obs+1,upper:T) # upper 95% limit

  Cmd> vector(mu_l,mu_u) # "exact" 95% confidence interval for mu
  (1)      5.4912      19.682

  Cmd> vector(cumpoi(x_obs,mu_u),cumpoi(x_obs,mu_l,upper:T)) # check
  (1)       0.025       0.025

                            Cross references
See also cumgamma(), cumchi(), invchi(), cumpoi(), runi().
```

2.175 invnor()

Usage:

```
   invnor(P [,upper:T or lower:F]), P REAL with elements of P between 0 and
     1, df > 0
```

Keywords: probabilities, confidence intervals

```
                                Usage
invnor(P) computes the quantiles (probability points, critical values)
of the normal distribution corresponding to each element of P. P must be
a REAL vector or array with elements between 0 and 1.  The result has
the same size and shape as P.

invnor(P,upper:T) and invnor(P,lower:F) compute upper tail quantiles of
the standard normal distribution.  The result is mathematically
equivalent to invnor(1 - P) but may be more accurate for small P.

A critical value for a two-tail Z-test with significance level alpha or
for a 1-alpha confidence interval may be computed as invnor(alpha/2,
upper:T) or invnor(1-alpha/2).

Critical values for a one-tail Z-test with significance level alpha are
computed as invnor(alpha) (lower tail test) and invnor(alpha,upper:T) or
invnor(1-alpha) (upper tail test).

invnor() is the inverse of cumnor() in the sense that invnor(cumnor(z))
should be the same as z within rounding error and cumnor(invnor(P))
should be the same as P within rounding error.

                            Cross references
See also cumnor(), rnorm()
```

2.176 invstu()

Usage:

```
invstu(P, df [,upper:T or lower:F]), P and df REAL, elements of P
  between 0 and 1, those of df > 0
```

Keywords: probabilities, random numbers, comparisons, confidence intervals

```
                                Usage
invstu(P,df) computes the quantiles (probability points, critical
values) of Student's t- distribution with df degrees of freedom
corresponding to each element of P.  P must be a REAL vector or array
with elements between 0 and 1 and df must be a REAL vector or array with
positive but not necessarily integral elements.

If df is a scalar the result has the same size and shape as P and df is
used to compute all the values.

If P is a scalar, the result has the same size and shape as df and
consists of P-th probability points for the different values of df.

If neither P nor df are scalars, they must be the same size and shape
and corresponding elements of P and df are used to compute elements of
the result.
```

```
invstu(P,df,upper:T) and invstu(P,df,lower:F) compute upper tail
quantiles.  The result is mathematically equivalent to invstu(1 - P, df)
but may be more accurate for small P.

invstu() is the inverse of cumstu() in the sense that, within rounding
error, invstu(cumstu(x,df),df) should be the same as x and
cumstu(invstu(P,df),df) should be the same as P.

                  Use in tests and confidence intervals
A critical value for a two-tail t-test on df degrees of freedom with
significance level alpha or for a 1-alpha confidence interval may be
computed as invstu(alpha/2,df, upper:T) or invstu(1-alpha/2,df).

Critical values for a one-tail t-test on df degrees of freedom with
significance level alpha are computed as invstu(alpha,df) (lower tail
test) and invstu(alpha,df,upper:T) or invstu(1-alpha,df) (upper tail
test).

Bonferronized critical values for K simultaneous two-tail t-tests with
significance level alpha or K simultaneous 1 - alpha confidence
intervals are computed as invstu(.5*alpha/K,df, upper:T).

                 Generating students t random variables
invstu(runi(n),df) will generate a random sample of size n from a
Student's t-distribution .

                             Cross references
See also cumstu(), runi().
```

2.177 invstudrng()

Usage:

```
   invstudrng(P, ngroup, errorDf [,epsilon:eps] [,upper:T or lower:F]),
     elements of P between 0 and 1, elements of ngroup integers >= 2,
     elements of errorDf >= 1, eps > 0 small
```

Keywords: probabilities, comparisons, confidence intervals

```
                                  Usage
invstudrng(P, K, Df) computes the quantiles (probability points,
critical values) of the Studentized range based on K normal variates and
an independent estimate of variance with Df degrees of freedom.  All
three arguments must be REAL.  The elements of P must be between 0 and
1. K must consist of integers >= 2, and the elements of Df must be >= 1,
not necessarily integers.

Any of the arguments P, K or Df that are not scalars must be vectors,
matrices or arrays all of the same size and shape.

invstudrng(P,2,Df) should be the same as sqrt(2)*invstu((1+P)/2,Df)
```

```
except for computational error.

invstudrng(P, K, Df, upper:T) and invstudrng(P, K, Df, lower:F) compute
upper tail quantiles.  The result is mathematically equivalent to
invstudrng(1 - P, K, Df).

                        Use in multiple comparisons
Many so-called multiple comparison methods are based on these quantiles,
among them the Tukey HSD (Honestly Significant Difference) and the SNK
(Student-Newman-Keuls) methods.  For example, if you have K independent
normal samples of size n, all with the same variance, and Ssq is the
pooled estimate of the variance, you can compute the 5% HSD as

  Cmd> q05 <- invstudrng(.05,K,K*(n-1),upper:T);hsd <- q05*sqrt(Ssq/n)

                Use in overall test of equality of means
In the same situation, you can test the null hypothesis that all means
are equal by the studentized range statistic computed as Q <-
(max(xbars) - min(xbars))/sqrt(Ssq/n).  This is an alternative to the
ANOVA F-statistic.  You can compute the alpha-level critical value for Q
as invstudrng(alpha, K,K*(n-1), upper:T).  Here xbars is a vector
containing the K sample means and Ssq is the pooled estimate of
variance.  See cumstudrng() for computing P values for Q.

                             Keyword 'epsilon'
invstudrng(P, K, Df, epsilon:eps [,upper:T]), where eps is a small
positive scalar, does the same computation with accuracy influenced by
eps.  The smaller the value of eps, the more accurate the result should
be, but the longer it will take to compute it.  The default value of eps
is 0.00001.

                             Cross references
See also cumstudrng(), invstu().
```

2.178 ipf()

Usage:

```
ipf([Model] [, print:F or silent:T, incr:T, pvals:T, maxiter:m,\
  epsilon:eps]), vec a REAL vector, m an integer > 0, eps REAL > 0
```

Keywords: glm, categorical data

```
                                   Usage
ipf(Model) uses iterative proportional fitting to compute a Poisson
regression (log linear) fit of the model specified in the CHARACTER
variable Model.  The default output is the deviance from the full model.

See topic 'models' for information on specifying Model.

                                Keyword 'inc'
ipf(Model,inc:T) fits the same model except a sequential analysis of
```

deviance is computed. The sequential analysis of deviance has a line for each term in the model giving the term name, degrees of freedom, and the change of deviance obtained by including the term in the given order. Because each of the submodels must be fit iteratively, with a complicated models or a large data set ipf(Model,inc:T) can take many times longer to execute than ipf(Model).

Keyword 'pvals'

ipf(Model [,...], pvals:T) prints chi-squared P values with each deviance.

If option 'pvals' has value True, P values will be printed unless pvals:F is an argument.

Model omitted

ipf([,keywords]) or ipf(,inc [,keywords]) fits the last model used by any of the GLM commands such as regress() or poisson(). See topic 'glm'.

Defaulting to poisson

If there are any non-factors in the model ipf() defaults to poisson().

ipf() also defaults to poisson(), if it does not identify the model as balanced. The only forms of balance it recognizes are complete balance (equal number of cases in every cell) and balanced main effect models (no interactions and all two-way marginals have equal cell sizes, for example a Latin square design)

Side effect variables created

ipf() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS, DF, HII, DEPVNAME, TERMNAMES, and STRMODEL. See topic 'glm'. All except HII should be the same as computed by poisson(Model,inc) used. Since HII cannot be computed easily, it is set to a constant vector with values m/n where m = (Model degrees of freedom) and n is the number of values in the dependent variable vector. Thus sum(HII) = m as it should. Without keyword phrase 'inc:T' (see below), TERMNAMES has value vector("","", ...,"Overall model","ERROR1"), DF has value vector(0,0, ..., ModelDF,ErrorDF) and SS has value vector(0,0,...,ModelDeviance, ErrorDeviance).

Keywords 'maxiter' and 'epsilon'

ipf(Model,maxiter:m,epsilon:eps), where m is a positive integer and eps is positive, is the same as ipf(Model) except up to m iterations may take place (the default is 25) and eps is the convergence criterion (default 1e-6). You need not specify either or both.

Keywords 'print' and 'silent'

ipf(Model [,...], print:F) is the same as ipf(Model [,...]) except that most printing is suppressed and the only result is to set the side effect variables.

ipf(Model [,...], silent:T) does computations, creating side effect variables, but prints nothing except actual error messages.

```
                              Limitations
Keyword phrase 'coefs:F' cannot be used with ipf().

Coefficients may be retrieved by coefs(); standard errors are not
available.  You must use poisson() if you require standard errors.
```

2.179 isarray()

Usage:

```
isarray(arg1 [,arg2, ...] [,real:T, logic:T, char:T, integer:T,\
  positive:T, negative:T, nonneg:T])
```

Keywords: macros, general, variables

```
                                  Usage
isarray(arg) returns True if arg is an array of any type, REAL, LOGICAL,
CHARACTER or LONG, and False otherwise.  If arg is undefined, isarray()
returns False.

isarray(arg,real:T) returns True if and only if arg is a REAL array.
Similarly isarray(arg,char:T) and isarray(arg,logic:T) return True
only if arg is a array of the specified type.  You can specify more
than one acceptable type; for example, isarray(arg,real:T,logic:T)
returns True only if arg is a REAL or LOGICAL array.

isarray(arg, integer:T), isarray(arg, positive:T), isarray(arg,
negative:T) and isarray(arg, nonneg:T) are similar, testing that arg is
a REAL array whose value has the specified property.  You can use
'integer:T' with any of 'positive:T', 'negative:T' and 'nonneg:T'.  You
cannot use 'char:T' or 'logic:T' with these keywords.

                           Multiple arguments
isarray(arg1, arg2, ..., argk [,keywords]) returns a LOGICAL vector,
each element of which is True or False depending on whether or not the
corresponding argument is a array with the properties, if any, specified
by keyword phrases.

                                 Purpose
The principal use of isarray() is in checking the arguments of a macro
for appropriateness.  See argvalue() for another way to check for the
properties of macro arguments.

                                Examples
Examples:
  Cmd> isarray(vector(x), matrix(x,4), array(x,2,2,2), structure(x))
      has value vector(T,T,T,F) when x has 8 elements.

  In a macro

   if (!isarray($1,real:T,logic:T)){
```

```
    error("$1 is not a REAL or LOGICAL")
  }

  ensures argument 1 is REAL or LOGICAL before proceeding.

                          Cross references
See also array(), error(), ischar(), isdefined(), isfactor(),
isfunction(), isgraph(), islogic(), ismacro(), ismatrix(), isname(),
isnumber(), isnull(), isreal(), isscalar(), isstruc(), isvector().
```

2.180 ischar()

Usage:

```
ischar(arg1 [, arg2, ...])
```

Keywords: macros, general, variables, character variables

```
                                Usage
ischar(arg) returns True if arg is a CHARACTER variable or quoted string
and False otherwise.  If arg is undefined, ischar() returns False.

ischar(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of
which is True or False depending on whether or not the corresponding
argument is of type CHARACTER.

The principal use of ischar() is in checking the arguments of a macro
for appropriateness.  See argvalue() for another way to check for the
properties of macro arguments.

                                Example
Example:
  Cmd> ischar("hello",3,T) # returns vector(T,F,F).

                          Cross references
See also topics 'macros', isarray(), isdefined(), isfactor(),
isfunction(), isgraph(), islogic(), ismacro(), ismatrix(), isname(),
isnull(), isnumber(), isreal(), isscalar(), isstruc(), isvector().
```

2.181 isdefined()

Usage:

```
isdefined(arg1 [, arg2, ...])
```

Keywords: macros, general, variables

```
                                Usage
isdefined(arg) returns True if arg exists in the MacAnova workspace and
```

```
False otherwise.  If arg is a built-in function, isdefined() returns
True.

isdefined(arg1, arg2, ..., argk) returns a LOGICAL vector, each element
of which is True or False depending on whether or not the corresponding
argument actually exists.

In a macro, isdefined() is useful for checking errors in a macro.  For
instance, a macro to compute mean square errors might have the line
 if(!isdefined(SS) || !isdefined(DF)){
    error("SS or DF not defined")}
before attempting to use SS or DF.

                            Cross references
See topics 'macros', isarray(), ischar(), isfactor(), isfunction(),
isgraph(), islogic(), ismacro(), ismatrix(), isname(), isnull(),
isnumber(), isreal(), isscalar(), isstruc(), isvector().
```

2.182 isfactor()

Usage:

```
isfactor(arg1 [, arg2, ...])
```

Keywords: macros, general, glm, variables

```
                                  Usage
isfactor(arg) returns True if arg is a factor created by factor() and
false otherwise.  If arg is underfined, isfactor() returns False.

isfactor(arg1, arg2, ..., argk) returns a LOGICAL vector, each element
of which is True or False depending on whether or not the corresponding
argument is a factor.

The principal use of isfactor() is in checking the arguments of a macro
for appropriateness.

                            Cross references
See also topics factor(), 'models', 'glm', anova(), isarray(), ischar(),
isdefined(), isfunction(), isgraph(), islogic(), ismacro(), ismatrix(),
isname(), isnumber(), isnull(), isreal(), isscalar(), isstruc(),
isvector().
```

2.183 isfunction()

Usage:

```
isfunction(arg1 [, arg2, ...])
```

Keywords: macros, general, variables

Usage

isfunction(arg) returns True if arg is a MacAnova function such as describe(), sum() or regress() and False otherwise. If arg is undefined, isfunction() returns False.

isfunction(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a MacAnova function.

The principal use of isfunction() is in checking the arguments of a macro for appropriateness. See argvalue() for another way to check for the properties of macro arguments.

Example

Example:

```
  Cmd> isfunction(PI, cos, boxcox, T) # returns vector(F,T,F,F)
```

Cross references

See also topics isarray(), ischar(), isdefined(), isfactor(), isgraph(), islogic(), ismacro(), ismatrix(), isname(), isnull(), isnumber(), isreal(), isscalar(), isstruc(), isvector().

2.184 isgraph()

Usage:

```
isgraph(arg1 [, arg2, ...])
```

Keywords: macros, general, variables

Usage

isgraph(arg) returns True if arg is a GRAPH variable and False otherwise. If arg is undefined, isgraph() returns False.

isgraph(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a GRAPH variable.

The principal use of isgraph() is in checking the arguments of a macro for appropriateness. See argvalue() for another way to check for the properties of macro arguments.

Cross references

See also topics 'graphs', 'macros', isarray(), ischar(), isdefined(), isfactor(), isfunction(), islogic(), ismacro(), ismatrix(), isname(), isnumber(), isnull(), isreal(), isscalar(), isstruc(), isvector().

2.185 islocked()

Usage:
```
islocked(arg1 [, arg2, ...])
```

Keywords: general, variables

```
                                  Usage
islocked(arg) returns True if arg is a locked variable and False
otherwise.  If arg is undefined, islocked() returns False.

islocked(arg1, arg2, ..., argk) returns a LOGICAL vector, each element
of which is True or False depending on whether or not the corresponding
argument is locked.

                              Cross references
See also lockvars(), unlockvars(), 'variables:"locked_variables"'.
```

2.186 islogic()

Usage:
```
islogic(arg1 [, arg2, ...])
```

Keywords: macros, general, variables, logical variables

```
                                  Usage
islogic(arg) returns True if arg is a LOGICAL variable and False
otherwise.  If arg is undefined, islogic() returns False.

islogic(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of
which is True or False depending on whether or not the corresponding
argument is of type LOGICAL.

The principal use of islogic() is in checking the arguments of a macro
for appropriateness.  See argvalue() for another way to check for the
properties of macro arguments.

                                 Example
Example:
  Cmd> islogic("hello",3,T) # returns vector(F,F,T).

                              Cross references
See also topics 'logic', 'macros', isarray(), ischar(), isdefined(),
isfactor(), isfunction(), isgraph(), ismacro(), ismatrix(), isname(),
isnumber(), isnull(), isreal(), isscalar(), isstruc(), isvector().
```

2.187 ismacro()

Usage:

```
ismacro(arg1 [, arg2, ...])
```

Keywords: macros, general, variables

Usage

ismacro(arg) returns True if arg is a macro and False otherwise. If arg is undefined, ismacro() returns False.

ismacro(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a macro.

The principal use of ismacro() is in checking the arguments of a macro for appropriateness. See argvalue() for another way to check for the properties of macro arguments.

Cross references

See also topics 'macros', isarray(), ischar(), isdefined(), isfactor(), isfunction(), isgraph(), islogic(), ismatrix(), isname(), isnull(), isnumber(), isreal(), isscalar(), isstruc(), isvector().

2.188 ismatrix()

Usage:

```
ismatrix(arg1 [,arg2, ...] [,real:T, logic:T, char:T, integer:T,\
  positive:T, negative:T, nonneg:T])
```

Keywords: macros, general, variables

Usage

ismatrix(arg) returns True if arg is a matrix of any type, REAL, LOGICAL, or CHARACTER, and False otherwise. For arg to be considered a matrix it is not necessary that ndims(args) be 2, just that no more than two dimensions have length greater than 1. In particular, a scalar or a vector is considered to be a matrix by ismatrix(). If arg is undefined, ismatrix() returns False.

ismatrix(arg,real:T) returns True if and only if arg is a REAL matrix. Similarly ismatrix(arg,char:T) and ismatrix(arg,logic:T) return True only if arg is a matrix of the specified type. You can specify more than one acceptable type; for example, ismatrix(arg,real:T,logic:T) returns True only if arg is a REAL or LOGICAL matrix.

ismatrix(arg, integer:T), ismatrix(arg, positive:T), ismatrix(arg, negative:T) and ismatrix(arg, nonneg:T) are similar, testing that arg is a REAL matrix whose value has the specified property. You can use 'integer:T' with any of 'positive:T', 'negative:T' and 'nonneg:T'. You cannot use 'char:T' or 'logic:T' with these keywords.

```
                          Multiple arguments
ismatrix(arg1, arg2, ..., argk [,keywords]) returns a LOGICAL matrix,
each element of which is True or False depending on whether or not the
corresponding argument is a matrix with the properties, if any,
specified by keyword phrases.

                                Purpose
The principal use of ismatrix() is in checking the arguments of a macro
for appropriateness.  See also argvalue() for another way to check for
the properties of macro arguments.

                                Examples
Examples:
  Cmd> ismatrix(vector(x), matrix(x,4), array(x,4,1,2), array(x,2,2,2))
      has value vector(T,T,T,F) when x has 8 elements.

   if (!ismatrix($1,real:T)){error("$1 is not a REAL matrix")}
     in a macro would check argument 1 is a REAL matrix.

                            Cross references
See also topics 'matrices', error(), isarray(), ischar(), isdefined(),
isfactor(), isfunction(), isgraph(), islogic(), ismacro(), isname(),
isnull(), isnumber(), isreal(), isscalar(), isstruc(), isvector().
```

2.189 ismissing()

Usage:

```
   ismissing(x) where x is REAL, LOGICAL or CHARACTER or a structure all of
     whose components are REAL, LOGICAL, or CHARACTER
```

Keywords: macros, general, missing values, variables, null variables

```
                                Usage
ismissing(x) returns a LOGICAL variable (with the same shape as x) which
is True where x is MISSING and False where x is not MISSING.  x must be
a vector, matrix, or array.  If x is a CHARACTER variable, an empty
string ("") is considered to be a missing value.

If x is a NULL variable, ismissing(x) is NULL.  See topic 'NULL'.

It is also acceptable for x to be a structure, whose non-structure
components are vectors, matrices or arrays.  In that case, ismissing()
returns a structure of the same form, each of whose non-structure
components is the result of applying ismissing() to the corresponding
component of x.

ismissing(x) has the same labels as x, if any.

                                Examples
```

```
Examples:
   ismissing(vector(1, 3, ?, 7)) and ismissing(vector("A","B","","Z"))
     both return vector(F, F, T, F)
   x[vector(ismissing(x))] <- -1 replaces all MISSING values in x by -1.
   sum(vector(ismissing(x))) returns the number of MISSING values in x,
     whether x is a vector, matrix, array, or structure.

                            Cross references
See also anymissing(), sum().
```

2.190 isname()

Usage:

```
   isname(arg1 [, arg2, ... ] [file:T  or  path:T]), arg1, ... CHARACTER
     scalars
```

Keywords: macros, general, variables

```
                                    Usage
isname(arg) returns T if the value of arg is a legal MacAnova name.
Argument arg must be a CHARACTER scalar or be missing.  See topic
'variables' for information on what is a legal name.

isname(arg1, ..., argk), where the arguments are either CHARACTER
scalars or missing returns a LOGICAL vector of length k with element j
being True if and only if argument k is a CHARACTER scalar whose value
is a legal MacAnova name.

isname(arg1, ..., path:T) does the same except the arguments are
checked as to whether they are appropriate file names or folder
(directory) names on the computer being used.

isname(arg1, ..., file:T), does the same as with 'path:T' except that
False is returned for any argument that can be only a directory, that is
the name ends in a path name separator ('/', or '\' on Windows or DOS,
or':' on Mac OS 9).

The checking done with 'path:T' and 'file:T' may not exactly match the
formal definition of what is a legal file name.  For instance, on
Unix/Linux, isname("-myfile.txt") is False, although '-myfile' is a
legal Unix/Linux file name, because names starting with '-' can lead to
difficulties.

It is legal for an argument to be missing.  For example, a <- isname()
sets a to F and a <- isname("cos",,"sin") sets a to vector(T,F,T).

The principal use of isname() is in checking the arguments of a macro
for appropriateness.

                                  Examples
Examples:
```

```
 isname("PI") returns True
 isname("123+4") returns False
 isname("x","@y",,"T","too_long_a_name") returns vector(T,T,F,F,F)
 isname("macanova.hlp","macros/",file:T) returns vector(T,F) on
   Unix/Linux, Windows and DOS
 isname("macanova.hlp","macros/",path:T) returns vector(T,T) on
   Unix/Linux, Windows and DOS
 isname("macanova.hlp",":macros:",file:T) returns vector(T,F) on a
   Mac OS 9
 isname("macanova.hlp",":macros:",path:T) returns vector(T,T) on a
   Macintosh

                            Cross references
See also topics 'macros', isarray(), ischar(), isdefined(), isfactor(),
isfunction(), isgraph(), islogic(), ismacro(), ismatrix(), isnull(),
isnumber(), isreal(), isscalar(), isstruc(), isvector(), nameof().
```

2.191 isnull()

Usage:

```
isnull(arg1 [, arg2, ...])
```

Keywords: macros, general, variables, null variables

```
                                   Usage
isnull(arg) returns T if arg has type NULL and false otherwise.

isnull(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of
which is True or False depending on whether or not the corresponding
argument is NULL.

See topic 'NULL' for information on NULL variables.

The principal use of isnull() is in checking the arguments of a macro
for appropriateness.

                                  Example
Example:
  Cmd> isnull(NULL, sqrt(2)) # returns vector(T, F)

                            Cross references
See also topics 'macros', isarray(), ischar(), isdefined(), isfactor(),
isfunction(), isgraph(), islogic(), ismacro(), ismatrix(), isname(),
isnumber(), isreal(), isscalar(), isstruc(), isvector().
```

2.192 isnumber()

Usage:

```
isnumber(arg1 [, arg2, ... ]), arg1, ... CHARACTER scalars
```

Keywords: macros, general, variables

```
                                  Usage
isnumber(arg) returns T if arg is a CHARACTER scalar or quoted string
such as "-3.1416" which represents a non-MISSING number.  It is an error
if arg is not a CHARACTER scalar.  It returns F when arg does not
represent a non-MISSING number, for example "MacAnova" or "?".  See
topic 'number' for information on what are legal numbers.

isnumber() (with no argument) is legal and returns F.

isnumber(arg1, ..., argk), where the arguments are either CHARACTER
scalars or empty returns a LOGICAL vector of length k with element j
being True if and only if argument k is a CHARACTER scalar which
represents a non-MISSING number.

It is legal for an argument to be missing.  For example, a <- isnumber()
sets a to F and a <- isnumber("3.14",,"henry") sets a to vector(T,F,F).

The principal use of isnumber() is in checking the arguments of a macro
for appropriateness.

                                 Examples
Examples:

  Cmd> isnumber("3.14")
  (1) T

  Cmd> isnumber(3.14) # not quoted
  ERROR: argument 1 to isnumber() is not CHARACTER scalar

  Cmd> isnumber("1e1000") # too large to be represented
  (1) F

  Cmd> isnumber("3.14",,paste(PI),"PI","?","T","3d10")
  (1) T       F       T       F       F       F       T

                            Cross references
See also topics 'macros', isarray(), ischar(), isdefined(), isfactor(),
isfunction(), isgraph(), islogic(), ismacro(), ismatrix(), isname(),
isnull(), isreal(), isscalar(), isstruc(), isvector(), nameof().
```

2.193 isreal()

Usage:

```
isreal(arg1 [, arg2, ...] [,positive:T or negative:T or nonneg:T]\
  [,integer:T])
```

Keywords: macros, general, variables

```
                                Usage
isreal(arg) returns True if arg is a REAL variable and False otherwise.
If arg is undefined, isreal() returns False.

isreal(arg, integer:T) does the same, except that True is returned only
if arg is REAL and all the values are integers.

isreal(arg, positive:T), isreal(arg, negative:T) and isreal(arg,
nonneg:T) do the same, except that True is returned only if arg is REAL
and all its elements have the indicated properties.  You can use
'integer:T' here as well.

                          Multiple arguments
isreal(arg1, arg2, ..., argk [keywords]) returns a LOGICAL vector, each
element of which is True or False depending on whether or not the
corresponding argument is REAL and satisfies the properties specified by
any keywords.

                                Purpose
The principal use of isreal() is in checking the arguments of a macro
for appropriateness.  See argvalue() for another way to check for the
properties of macro arguments.

                                Example
Example:
  Cmd> isreal("hello",3,T)  # returns vector(F,T,F).

                          Cross references
See also topics 'macros', isarray(), ischar(), isdefined(), isfactor(),
isfunction(), isgraph(), islogic(), ismacro(), ismatrix(), isname(),
isnull(), isnumber(), isscalar(), isstruc(), isvector().
```

2.194 isscalar()

Usage:

```
isscalar(arg1 [,arg2, ...] [,real:T, logic:T, char:T, integer:T,\
  positive:T, negative:T, nonneg:T])
```

Keywords: macros, general, variables

```
                                Usage
isscalar(arg) returns True or False, depending on whether arg is a
scalar, that is a REAL, LOGICAL, or CHARACTER variable all of whose
dimensions are 1.  If arg is undefined, isscalar() returns False.
```

isscalar(arg,real:T) returns True if and only if arg is a REAL scalar. Similarly isscalar(arg,char:T) and isscalar(arg,logic:T) return True only if arg is a scalar of the specified type. You can specify more than one acceptable type; for example, isscalar(arg,real:T,logic:T) returns True only if arg is a REAL or LOGICAL scalar.

isscalar(arg, integer:T), isscalar(arg, positive:T), isscalar(arg, negative:T) and isscalar(arg, nonneg:T) are similar, testing that arg is a REAL scalar whose value has the specified property. You can use 'integer:T' with any of 'positive:T', 'negative:T' and 'nonneg:T'. You cannot use 'char:T' or 'logic:T' with these keywords.

Multiple arguments

isscalar(arg1, arg2, ..., argk [,keywords]) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a scalar with the properties, if any, specified by keyword phrases.

Purpose

The principal use of isscalar() is in checking the arguments of a macro for appropriateness. See argvalue() for another way to check for the properties of macro arguments.

Examples

Examples:

```
  Cmd> isscalar(1,matrix(PI,1), run(5),"hello",F)
```

has value vector(T,T,F,T,T)

In a macro

```
  if (!isscalar($1,logic:T)){error("$1 not T or F")}
```

would check that argument 1 is a LOGICAL scalar.

Cross references

See topics 'macros', isarray(), ischar(), isdefined(), isfactor(), isfunction(), isgraph(), islogic(), ismacro(), ismatrix(), isname(), isnull(), isnumber(), isreal(), isstruc(), isvector().

2.195 isstruc()

Usage:

```
   isstruc(arg1 [, arg2, ... ])
```

Keywords: macros, general, structures

Usage

isstruc(arg) returns True or False, depending on whether arg is a structure. If arg is undefined, isstruc() returns False.

isstruc(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of

which is True or False depending on whether or not the corresponding argument is a structure.

The principal use of isstruc() is in checking the arguments of a macro for appropriateness. See argvalue() for another way to check for the properties of macro arguments.

Cross references

See also topics 'structures', 'macros', isarray(), ischar(), isdefined(), isfactor(), isfunction(), isgraph(), islogic(), ismacro(), ismatrix(), isname(), isnull(), isnumber(), isreal(), isscalar(), isvector().

2.196 isvector()

Usage:

```
isvector(arg1 [,arg2, ...] [,real:T, logic:T, char:T, integer:T,\
  positive:T, negative:T, nonneg:T])
```

Keywords: macros, general, variables

Usage

isvector(arg) returns True or False, depending on whether arg is a vector of any type, REAL, LOGICAL or CHARACTER. A matrix or array is considered to be a vector by isvector() if all dimensions except the first have length 1. In particular, if arg is a scalar, isvector(arg) returns True, while if arg is a row vector (dimensions 1,m with m > 1), isvector(arg) returns False. If arg is undefined, isvector(arg) returns False.

isvector(arg,real:T) returns True if and only if arg is a REAL vector. Similarly isvector(arg,char:T) and isvector(arg,logic:T) return True only if arg is a vector of the specified type. You can specify more than one acceptable type; for example, isvector(arg,real:T,logic:T) returns True only if arg is a REAL or LOGICAL vector.

isvector(arg, integer:T), isvector(arg, positive:T), isvector(arg, negative:T) and isvector(arg, nonneg:T) are similar, testing that arg is a REAL vector whose value has the specified property. You can use 'integer:T' with any of 'positive:T', 'negative:T' and 'nonneg:T'. You cannot use 'char:T' or 'logic:T' with these keywords.

Multiple arguments

isvector(arg1, arg2, ..., argk [,keywords]) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a vector with the properties, if any, specified by keyword phrases.

Purpose

The principal use of isvector() is in checking the arguments of a macro for appropriateness. See argvalue() for another way to check for the

```
properties of macro arguments.

                                Examples
Examples:
  Cmd> isvector(7, vector(x), matrix(x,5), array(x,5,1,1), matrix(x,1))
has value vector(T,T,T,T,F) if x has 5 elements.

In a macro
  if(!isvector($1,char:T)){
        error("$1 is not a CHARACTER vector")}
would check that argument 1 is a CHARACTER vector.

                             Cross references
See also topics 'vectors', 'macros', isarray(), ischar(), isdefined(),
isfactor(), isfunction(), isgraph(), islogic(), ismacro(), ismatrix(),
isname(), isnull(), isnumber(), isreal(), isscalar(), isstruc().
```

2.197 keyvalue()

Usage:

```
   keyvalue(keyname1:val1, [keyname2:val2, ...] targetkey [, properties]\
     [,default:defVal]), targetkey a CHARACTER scalar, Properties a
     CHARACTER scalar or vector whose elements are one or more of "array",
     "character", "count", "graph", "integer", "logic", "macro", "matrix",
     "nonmissing", "nonnegative", "notnull", "number", "positive", "real",
     "scalar", "square", "string", "structure", "TF" and "vector", and
     defVal arbitrary.
   keyvalue(str, targetkey [, properties] [,default:defVal), str a
     structure
   keyvalue(, targetkey [, properties] [,default:defVal])
```

Keywords: syntax, macros

```
                                   Usage
keyvalue(keyname1:val1, keyname2:val2, ... , TargetKey) attempts to
match keyname1, keyname2, ... with CHARACTER scalar or quoted string
TargetKey.  If no match is found, keyvalue() returns NULL.  If a match
is found, the corresponding keyword value is returned.

keyvalue(keyname1:val1, keyname2:val2, ... , TargetKey, default:defVal)
does the same except that defVal is returned if no matching keyword is
found.

keyvalue(keyname1:val1, keyname2:val2, ..., TargetKey, Properties
[,default:defVal) does the same, except that the value of a matched
keyword is returned only if it has the properties specified by CHARACTER
scalar or vector Properties.  When TargetKey is not matched and
'default:defVal' is an argument, defVal value is returned only if it has
all the properties specified by Properties.  See below for an
explanation of Properties.
```

```
The principal use of keyvalue is in a macro when keyname1:val1,
keyname2:val2, ... are supplied by $K as in
  @nsig <- keyvalue($K,"nsig","positive integer scalar", default:5)
See below for another example and topic 'macro_syntax' for an
explanation of $K.

                          Keyword 'specifying'
TargetKey, the second argument, is usually a legal keyword name such as
"nsig".  In this case an exact match of one of the keyword names is
needed.

However, TargetKey can also contain the "wild card" characters '*' and
'?' so that it provides a pattern used to match a keyword name.  '*'
will match any 0 or more successive characters and '?' will match any
single character.  For example, if TargetKey = "pow*", it matches
keywords 'pow', 'power' and 'powers', among others, but does not match
'polar'.  Similarly, if targetKey = "m??imum", it matches both keywords
'maximum' and 'minimum', among others.  See match() for more information
about inexact matching using '*' and '?'.

You can use structure(keyname1:val1, keyname2:val2, ...) instead of
keyname1:val1, keyname2:val2, ... .

keyvalue(, TargetKey [,Properties]) is legal and returns NULL.
Arguments TargetKey and Properties are checked for appropriateness.

                             Use in macros
The principal use for keyvalue() is in writing macros that expect
keyword phrase arguments.  It allows easy retrieval and checking of
keyword values.  A key point is that it is an error when the value of a
matched keyword value or the default value does not have the specified
property or properties.  When this occurs in a macro, execution of the
macro is terminated.

Here is a fragment of a macro that recognizes an optional macro argument
of the form 'weights:w' where w must be a REAL vector with positive
elements.

  @weights <- keyvalue($K,"weights","real positive vector",\
    default:rep(1,@n))

If 'weights:w' is an argument to the macro, @weights is set to w;
otherwise @weights is set to the default rep(1,@n).  Presumably @n was
set previously.  If w is not a REAL vector with positive elements the
macro will be terminated with the message
  ERROR: value of keyword 'weights' is not a vector of positive REALs

If "weights" were replaced by "weight*", keyvalue() would return w if
'weight:w' or 'weights:w' were an argument.  This is helpful in writing
a more user friendly macro, that recognizes both 'weight' or 'weights'
when 'weights' is expected.

In a macro, keyvalue(structure($K),...) is almost equivalent to
```

keyvalue($K,...), except that a warning message is printed when there are no keyword arguments to the macro ($K expands to nothing).

Description of Properties

Properties is usally a CHARACTER scalar or quoted string containing one or more of the following, separated by blanks or tabs (not commas):

"array"	"integer"	"matrix"	"notnull"	"scalar"	"vector"
"character"	"logic"	"nonmissing"	"positive"	"square"	
"graph"	"macro"	"nonnegative"	"real"	"structure"	

An example would be "nonmissing real vector". Properties can also be a CHARACTER vector with each element containing one or more properties from this list, for example, vector("nonmissing","real","vector").

In addition, there are properties thats are abbreviations for combinations of properties specifying types of scalars:

"number"	means	"nonmissing real scalar"
"count"	means	"nonnegative integer scalar"
"TF"	means	"nonmissing logical scalar"
"string"	means	"character scalar"

Not all combinations of properties are permitted. See below for details.

Any 3 character or longer initial segment of a property will match it, except that "nonnegative", "nonmissing", "string" and "structure" require 4. For example, "vec", "vect", "vecto", ... all match "vector".

Property types

Each recognized property (other than the abbreviations "TF", "number", "count and "string") is classified as to whether it describes the type, shape, value or sign of a variable:

Kind of property	Property names
Type	"real", "logic", "character", "macro", "graph", "notnull"
Shape	"scalar", "vector", "matrix", "array", "structure", "square"
Value	"integer", "nonmissing"
Sign	"positive", "nonnegative"

Property restrictions

Combinations of properties are restricted as follows:

- No more than one of each of the 4 kinds of property can be specified except that "square" and "matrix" can be used together
- Property "structure" is illegal with any Sign or Value property and with "macro", "graph" and "notnull"
- Properties "positive", "nonnegative" and "integer" imply properties "real" and "nonmissing" and are illegal with any Type property except "real". They are legal with property "number"
- Property "nonmissing" is illegal with any Type property except "real" and "logical".
- Properties "macro", "graph" and "notnull" cannot be combined with any other properties.

```
                          Cross references
See also argvalue(), getkeywords(), nameof(), isscalar(), isvector(),
ismatrix(), isarray(), isreal(), ischar(), islogic(), ismacro(),
isstruc(), isnumber(), isgraph(), isdefined(), 'keywords', 'macros'.
```

2.198 keywords

Usage:

```
   print(nsig:5,x), plot(Obs_No:x,Response:y), regress(Model, pvals:T),
     setoptions(format:"12.5g") are examples of keyword usage
```

Keywords: syntax

```
                                Description
Many functions accept 'keyword phrases' such as 'xlab:"Weight"',
'down:T' or 'nsig:7' as optional arguments.

A keyword phrase has the form Name:Value, where Name is a name of no
more than 10 characters starting with a letter (a-z or A-Z).  Value is a
variable or constant.

When used as values of a keyword phrase, 'T' and 'F' can often be
interpreted as 'yes' and 'no' or 'allow' and 'suppress'.

Keywords often provide optional information, or specify variants of the
usual computation. For example, print(nsig:7,x,y,nsig:3,z) prints x and
y with 7 significant digits and z with 3, and screen(Model,mbest:6)
specifies that screen() should find the 6 best subsets of independent
variables.

Keyword phrases may also be used to name components when using
structure() and strconcat(), to label output on most output commands
(print, write, etc.), and to provide labeling information on plotting
commands such as plot() and chplot().  The only limitation is that any
keyword that is specially recognized by the command such as 'labels' or
'format' cannot be used as such a label.

                                  Example
Example:
  Cmd> boxplot(split(wts,race),vertical:T,\
         title:"Weights of Army recruits by race",ylab:"Pounds")

                          Cross references
See also topics 'syntax', structure(), strconcat(), keyvalue()
```

2.199 kmeans()

Usage:

```
kmeans(y [,means or classes] [,kmax:k1,kmin:k2,start:method,standard:F,
  weights:wts, quiet:T]), y a REAL matrix, means a REAL matrix with
  ncols(y) columns, classes a REAL vector with nrows(y) rows, k1 and k2
  positive integers, k1 >= k2, method one of "random", "optimal",
  "means", or "classes", wts a REAL vector with nrows(wts) = nrows(y)
```

Keywords: multivariate analysis

```
                                  Usage
kmeans(y, kmax:k1 [, kmin:k2]) performs k-means clusterings of the rows
of REAL matrix y, starting with k1 clusters, and successively merging
clusters until there are k2 clusters.  By default the data are
standardized and the initial clusters are selected randomly.  At each
stage, cases are reallocated among clusters in an attempt to minimize
the sum of the within-cluster sums of squares.  If kmin:k2 is omitted,
k2 is taken to be k1.

It is an error when k2 > k1.

kmeans() returns a structure with components 'classes' and 'criterion'.
Component classes is a nrows(y) by k2-k1+1 matrix (vector if k2 = k1)
containing the cluster membership at each stage.  Component criterion is
a k2-k1+1 REAL vector containing the minimized criterion at each stage.

By default, a brief history of the merging process is printed, including
the values of the criterion being minimized.

           Keywords 'random', 'optimal', 'means' and 'classes'
kmeans(y, kmax:k1 [, kmin:k2], start:"random") is identical to kmeans(y,
kmax:k1 [, kmin:k2]).

kmeans(y, kmax:k1 [, kmin:k2], start:"optimal") attempts to select the
initial clusters so as to minimize the within-cluster sums of squares
for column 1 of y.

kmeans(y, Means [, kmin:k2], start:"means"), where Means is a k1 by
ncols(y) matrix, selects as initial cluster j those rows of y that are
closer to row j of Means than to any other row of Means using (Euclidean
distance).  If kmax:k1 is an argument with k1 != nrows(Means), a warning
message is given and nrows(Means) is used.  If there are duplicates
among the rows of Means, a warning message is printed.

kmeans(y, Classes [, kmin:k2], start:"classes"), where Classes is a
vector of nrows(y) positive integers <= 255, uses Classes to specify
initial clusters.  If kmax:k1 is an argument with k1 != max(Classes), a
warning message is given and max(Classes) is used.  If there are empty
classes (not all integers between 1 and max(Classes) are present), the
empty classes are "squeezed out", and max(Classes) reduced accordingly.

                         Additional keywords
      standard:F                Do not standardize before clustering
```

```
      weights:wts              Use weighted means and sums of squares
                               with wts a REAL vector of length
                               nrows(y) with w[i] > 0.
      quiet:T                  Suppress printing of clustering
                               history.

                          Cross references
See also cluster().
```

2.200 labels

Usage:

```
   This topic has information on coordinate labels.
   Functions for working with labels are:
     setlabels(x, labs [,silent:T])
     setlabels(x, NULL) # remove labels
     labs <- getlabels(x [,silent:T])
     labs <- getlabels(x,vector(1,3,5) [,silent:T])
     if (haslabels(x)){..do something with labels...}
     y <- vector(x,labels:labs [, silent:T])
     y <- matrix(x,labels:structure(rowLabs,colLabs) [, silent:T])
     y <- array(x,labels:structure(lab1,lab2,...) [, silent:T])
     y <- structure(comp1, comp2, ..., labels:labs [, silent:T])
     y <- strconcat(var1, var2, ..., labels:labs [, silent:T])
     y <- hconcat(x1,x2,...,labels:structure(rowLabs,colLabs) [, silent:T])
     y <- vconcat(x1,x2,...,labels:structure(rowLabs,colLabs) [, silent:T])
     y <- read(fileName,labels:structure(rowLabs,colLabs))
     y <- matread(fileName,labels:structure(rowLabs,colLabs))
```

Keywords: general, variables, output

```
                              Description
MacAnova vectors, matrices and arrays may have labels for each
dimension.  The label for dimension k is a CHARACTER vector with length
dim(x)[k].  When a variable x has any labels it has them for all
dimensions.

A structure Str may have a single vector of labels of length ncomps(Str)
to label the components.  This is distinct from labels the individual
components may have and distinct from the component names.

The primary function of the labels for a variable is to provide
informative identification of coordinates when the variable is printed.

Labels belonging to x are sometimes propagated to new variables computed
from x.  See below for details.

                            Retrieving labels
getlabels(x) retrieves all the labels, if any, associated with variable
x.  When x is a vector or a structure, the result is a CHARACTER vector;
otherwise the result is a structure with ndims(x) CHARACTER vector
```

components.

getlabels(x, 2), say, retrieves the labels for dimension 2 of x as a CHARACTER vector. getlabels(x,vector(1,3)), for example, retrieves as a structure with two components the labels associated with dimensions 1 and 3 of x. See getlabels().

haslabels(x) is True if and only if variable x has coordinate labels.

Removing labels

setlabels(x, NULL), removes the labels from a scalar, vector, matrix, array or structure x.

Attaching labels

setlabels(x, Labs) adds coordinate labels to an existing variable. When x already has labels they are replaced.

In these usages, Labs is a CHARACTER vector or scalar, or a structure with CHARACTER vector or scalar components, one for each dimension of the variable to be labeled.

Examples

Examples:

```
  Cmd> setlabels(x, vector("MN","WI","IA","ND","SD","NE")) # x a vector

  Cmd> setlabels(w, structure(vector("M","F"),\
         vector("Urban","Suburban","rural"))) # w a 2 by 3 matrix
```

It is not an error when the number of vectors of labels supplied does not match the number of dimensions. Extra labels are ignored and missing ones are assumed to be "@" which will be printed as numbers in parentheses (see below). In both cases a warning message is normally printed.

Keyword 'labels'

You can use keyword 'labels' on vector(), matrix(), array(), hconcat(), vconcat(), structure(), strconcat(), read() and matread() to create a labeled variable. The general usage

```
  xxxxxx(... , labels:Labs [,silent:T])
```

where xxxxxx() is one of these functions and Labs is as described for setlabels().

Wrong length labels

Supplying a vector of coordinate labels of the wrong length to setlabels() is an error. On other commands on which you can set labels using keyword 'labels' (matrix(), for example), providing labels of the wrong length produces only a warning message that the labels will be ignored.

Keyword 'silent'

On any of the commands which set labels, you can suppress warning messages by keyword phrase 'silent:T'.

```
                              Labels in files
When variable x has labels, matprint(fileName, x) and matwrite(fileName,
x) write the labels to the file immediately following x in a format that
allows
  Cmd> x <- read(fileName, "x")
to retrieve both data and labels for x from the file.  Topic
'matread_file' describes the file format of labeled variables.  See also
read() and matread().

                            Expanding labels
                         Expanding Scalar Labels
When you provide labels using keyword 'labels' on a command or as an
argument to setlabels(), any scalar labels (quoted strings or CHARACTER
vectors of length 1) are treated specially.

A scalar label, say "root", for a coordinate with length > 1 is expanded
to vector("root1","root2",...).  For example, labels:structure("A","B ")
generates labels vector("A1","A2",...) and vector("B 1","B 2",...).

This doesn't happen if a scalar label starts with '@' or is one of "",
"#", "(", "[", "{", "<", "/", or "\\".

Scalar label "" is expanded to rep("", length) resulting in its
dimension having no visible labels.

Scalar label "#" is expanded to vector("1","2",...).

Scalar label "(" is expanded to vector("(1)","(2)", ...) and similarly
for the other special characters, with the first element of the label
being "[1]", "{1}", "<1>", "/1/", or "\\1\\".

Scalar label "@" is expanded to rep("@", n) and a scalar label starting
with '@', say "@anything" is expanded to rep("@anything", n).  Such
labels are further expanded when they are printed.

                    Expanding labels starting with '@'
      Expansion of labels starting with '@' when they are printed
A label of the form rep("@", n) or rep("@anything", n) is interpreted
specially when it is printed.  At that time, it is further expanded
similarly to the way scalar labels that do not start with '@' are
expanded when they are created.

rep("@#", n) prints as '1', '2', ... .

rep("@[", n) prints as '[1]', '[2]', ..., and similarly with "@(", "@{",
"@<", "@/",or "@\\".

rep("@", n) prints "bracketed" labels using the default labeling style,
usually using '(' and ')'.  This style can be changed by option
'labelstyle'.  For example, after setoptions(labelstyle:"["), "@" has
the same effect as "@[".  See topic setoptions(), subtopic
'options:"labelstyle"'.
```

```
rep("@anythingelse", n) prints as 'anythingelse1', 'anythingelse2', ....

Note: The use of '@' delays the substitution of numerical indices until
they are actually used in printing, so that the same value may have
different printed labels at different times.  See the paragraph on
propagation of labels below.

When successive coordinates have the same type of "bracket" label
starting with '@' created by, say, labels:structure("@[","@[","["), the
printed labels are combined to, say, '[1,2]'.

                                Examples
Examples
  Cmd> setlabels(x, structure("#","X"))
  x now has labels vector("1","2",...) and vector("X1","X2",...).

  Cmd> y <- vector(vecread(fileName),labels:structure("Case ","Y"),\
        silent:T)
  y has labels vector("Case 1","Case 2",...) and vector("Y1","Y2",...)

  Cmd> z <- read("MacAnova.dat","irisdata",\
        labels:structure("",vector("Variety","Y1","Y2","Y3","Y3")))
  The first label of z is rep("",nrows(z)) which does not print.

  Cmd> logy <- matrix(log(y),labels:structure("@",log(getlabels(y,2))))
  The labels of logy y will be rep("@",nrows(y)) and vector("log(Y1)",
  "log(Y2)", ...).  When printed the row labels of logy or any subset of
  rows of logy will be '(1)', '(2)', ... .

                          Propagation of labels
A variable created by extracting part of a labeled variable using
subscripts is labeled with the appropriate subsets of the labels.
Examples:
After
  Cmd> x <- matrix(run(9),3,labels:structure("[","A "))[-1,-1]
x has labels vector("[2]","[3]") and vector("A 2", "A 3").

After
  Cmd> x <- matrix(run(9),3,labels:structure("@[","@A "))[-1,-1]
has labels vector("@[","@[") and vector("@A ","@A ").  When x is
printed, the row labels will be '[1]', '[2]' and the column labels will
be 'A 1', 'A 2', even though these are rows 2 and 3 and columns 2 and 3
of matrix(run(9),3).  That is, the special expansion of labels starting
with '@' occurs when they are printed, not when they are created.

cos(x), sqrt(x), and other transformation of x have the same labels as
x.

ismissing(x), rank(x), rankits(x) and halfnorm(x), but not sort(x) or
grade(x) have the same labels as x.

x' has the same label vectors as x but in reverse order
```

When the result of sum(x), min(x) and other transformation that operate along the first dimension of x is not a scalar, its label for the first dimension is "@" and the remaining labels match those of x.

+x, -x and !x have the same labels as x.

If OP is a binary operator such as '+', '-', '*', '==', ..., but not a matrix operator such as %*%, %c%, and %C%, then x OP y often has the labels of x. When x does not have labels, x OP y may have the labels of y. If both x and y are scalar variables, x OP y does not have labels, even if one or both of x and y do have labels.

When matrices x and y both have labels, x %*% y, x %c% y, and x %C% y have labels taken from the row and or column labels of x and y in the obvious way. When only one of x or y has labels, the result is labelled as if the one without the labels had row and column labels of the form rep("@",m).

In most cases, functions and operators that transform structures to similar shaped structures propagate labels for structure components. This includes max(), min(), sum(), prod(), sort(), rank(), grade(), ismissing(), and mathematical transformations such as cos(), log() and sqrt().

When x is a labelled matrix, eigen(x)$vectors and releigen(x,y)$vectors have the same row labels as x and column labels of the form vector("(1)", "(2)", ...).

When x is a labelled matrix, the matrices of left and right singular vectors as computed by svd() have labels. Their row vectors are the row and column labels of x, respectively. Their column labels are of the form vector("(1)", "(2)", ...).

If a is a labelled square matrix, the row and column labels of solve(a) are the column and row labels of a, respectively.

If a is a labelled square matrix, the row and column labels of solve(a, b) (a %\% b) are the column labels of a and b, respectively; the row and column labels of rsolve(a,b) (b %/% a) are the row labels of b and a, respectively. When b has no labels, they are assumed to have the form rep("@",m). See topics solve(), rsolve(), 'matrices'.

When x is a labelled matrix, cor(x) has row and column labels matching the column labels of x. cor(x1, x2, ...) has no labels, even if x1, x2, ... have lables.

When x is a labelled matrix, rft(x) and hft(x) have the same column labels as x with row labels of the form rep("@", nrows(x)). The same is true for cft(x) when ncols(x) is even.

If y is a response variable in a GLM command, its labels are propagated to side effect variables RESIDUALS, WTDRESIDUALS, and HII.

After regress(), COEF and XTXINV are labeled with the names of the variables (including "CONSTANT" when appropriate).

After any GLM function producing side effect variables DF and SS, DF and the first dimension of SS are labeled with TERMNAMES. After manova(), dimensions 2 and 3 of SS are labeled with the column labels of the response variable, it if has labels, and with vector("(1)","(2)", ...), otherwise. The actual brackets used in the default labelling are determined by the value of option 'labelstyle'. For example, if the value of 'labelstyle' is "[", the default labels are vector("[1]", "[2]"',...). See subtopic 'options:"labelstyle"'.

After manova(), the column labels of the response variable are attached to the last dimension of each vector, matrix, or array returned by coefs() and secoefs().

When any term name is longer than 12 characters (the maximum size for a structure component name), structure output of coefs() and secoefs() is labeled with the full term names.

When a structure with component labels is printed, the labels are printed instead of the component names.

2.201 launching

Usage:

```
Unix/Linux and DOS: macanova [-q] [File Options] [Screen Options]
                    at Unix/Linux or DOS prompt
Macintosh:          Double click on MacAnova icon or MacAnova file icon
Windows:            Double click on MacAnova for Windows icon
```

Keywords: general, files

Nonwindowed versions

For non-windowed versions (Unix/Linux or DOS), type 'macanova' at the Unix/Linux or DOS prompt. If the MacAnova directory is not in the search path, you will need to specify the complete path. See below for command line options.

For non-windowed versions, you may also use "redirected" input and output to run an entire analysis noninteractively and save the output.

```
macanova [options] < cmdFile > outputFile
```

will read commands from cmdFile and save the results in outputFile.

Microsoft Windows Version

If MacAnova is installed correctly under Windows 95/98/NT/XP there is a MacAnova entry on the Start menu, with subentries for all installed versions and possibly for browser based help files. All versions can also be launched from the DOS prompt.

In addition, the installer should also register the extentions

.mvbat, .mvsave, and .mvout for MacAnova batch files, save (workspace) files, and output files. Double clicking any file with one of those extensions should start MacAnova, execute the batch file, restore the save file, and/or open the output file.

Finally, you may drag and drop save, batch, and output files onto the MacAnova for Windows icon.

See also topic 'dos_windows', 'carapace'.

Linux GTK

Assuming the Carapace GTK version has been named macanovacpc and is in a directory in your search path, type

```
macanovacpc [-q] [File Options] [Path Options] [Screen Options]
```

at the Unix/Linux prompt, where items in [...] are options. See below for details on command line options.

See also topics 'unix', 'carapace'.

Macintosh Mac OS X

Double click the MacAnova icon. In addition, the Finder should also recognize files with extensions .mvbat, .mvsave, and .mvout (files with creator mat2 and types TeXT, S000, and TeXT) as MacAnova batch files, save (workspace) files, and output files. Double clicking any file of those types should start MacAnova, execute the batch file, restore the save file, and/or open the output file. You can, in fact, shift click on more than one such file and then choose Open from the Finder File menu to do more than one operation.

It is possible, though rather awkward, to launch MacAnova from a Terminal window. Change into the directory where MacAnova is located; it will show up as MacAnova.app. Then change into the MacAnova.app directory, and the Contents directory within that, and the MacOS directory within that. There you should find macanovacpc, which is the actual executable. Typing ./macanovacpc will start MacAnova. In this fashion you may also use command line options.

See also topic 'macintosh'.

Command line options

The windowed versions of MacAnova have a greater array of command line options than the nonwindowed versions. However, most casual users will never need these options, particularly for windowed versions.

The following tables give an option, whether it is usable in windowed (W) or nonwindowed (N) versions, and its use. There will be a 1 in the W or N column if an option can be used exactly once, and an asterisk if the option can be used multiple times (up to 50).

File related options

Option	W	N	Use
-restore filename	1	1	execute restore("filename") at startup
-batch filename	*	1	execute batch("filename") at startup
-f startfile	*	1	execute batch("startfile") silently
-help helpfile	1	1	use helpfile as the default help file
-macro filename	*	1	add filename to variable MACROFILES
-open filename	*		open filename in a command window
-data filename	*	1	add filename to DATAFILES
-addhelp filename	*		add filename to HELPFILES
-config filename	1		use filename as the Carapace preference file

Path related options

Option	W	N	Use
-home pathname	1	1	set variable HOME to pathname
-appdir pathname	1		set MacAnova root directory to pathname
-path pathname	*		add pathname to DATAPATHS
-dpath pathname		1	add pathname to DATAPATHS
-mpath pathname		1	add pathname to DATAPATHS

Other options

Option	W	N	Use
-q	1	1	suppress banner at startup
-prompt string	1	1	set the MacAnova prompt to string
-bprompt string	1	1	set the batch prompt to string (for -batch)
-e expression	1	1	execute expression at startup
-eq expression		1	execute expression and then quit
-l lines	1	1	set page height to l lines
-w columns	1	1	set page width to w columns
-hist count	1	1	set history length to count items

Option details

-restore saveFile
  The equivalent of 'restore("saveFile")' is executed at startup and
  MacAnova.ini.txt is not read and executed. See 'customize', restore().

-batch batchFile
  The equivalent of 'batch("batchFile")' is executed after
  initialization.

-f initFile
  File initFile is executed silently as a batch file at startup instead
  of file MacAnova.ini.txt (see 'customize').

-help helpFile
  Help information will be taken from file helpFile rather than the
  default help file MacAnova.hlp.txt.

-macro macroFile
  "macroFile" will be added to the beginning of Pre-defined CHARACTER
  variable MACROFILES. This will mean that pre-defined macro
  getmacros() will search the file before the standard macro files. You

```
  can accomplish the same thing after starting MacAnova by
  addmacrofile("macroFile").  See topics getmacros() and addmacrofile().

-open windowFile
  Load the contents of windowFile into a command-output window, as if
  Open were selected on the File menu.  No startup message printed.

-data dataFile
  "dataFile" will be added to the beginning of pre-defined CHARACTER
  variable DATAFILES.  DATAFILES is used by pre-defined macro
  getdata() to make it easy to read data from a standard file.  See
  topic getdata().

-addhelp helpFile
  "helpFile" will be added to the beginning of pre-defined CHARACTER
  variable HELPFILES.  Files in HELPFILES are searched when using the
  command help().  See topic help().

-config configFile
  Use configFile instead of the default file MacAnova.config.  Some
  aspects of MacAnova (fonts, window sizes, etc) can be controlled
  through the configuration file.

-home homePath
  Predefined CHARACTER variables HOME will have "homePath" as value
  instead of a default value.  HOME is used to expand file names of the
  form "~/filename".  For instance, when HOME is "dataDir", "~/filename"
  is expanded to "dataDir/fileName".  See topic 'files'.

-appdir appPath
  By default, the last element of DATAPATHS is the directory where
  MacAnova is located.  You may override that value by using -appdir.
  This is really only useful on Unix/Linux, because the location of
  MacAnova cannot be determined at execution time and a default value
  must be used instead.  Here you can override the default.

-path pathName (or -dpath dataPath  or -mpath macroPath)
  Add pathName to the front of the CHARACTER vector DATAPATHS, which
  contains a set of directories that are searched when you attempt to
  read a file (for example, by using vecread(), read(), matread() or
  macroread()). If the file cannot be found in the default directory,
  MacAnova searches in the directories in DATAPATHS.  See topic
  DATAPATHS.

If -q is present, the startup message will not be printed and the
  welcome screen is not shown.

-e Expr
  Execute the MacAnova command in Expr as if it were the contents of
  a batch file.  Expr will be executed before anything else is done.

-eq Expr
  Execute the MacAnova command in Expr as if it were the contents of
```

```
  a batch file and then immediately quit.  Expr will be executed
  before anything else is done.  This option does not make sense
  for windowed versions.

-prompt Prompt
  Sets the non-batch command line prompt.  Usually Prompt should end
  with a space, for example, -prompt "Next? ".  This becomes the default
  prompt that will be set by setoptions(default:T).  See topics
  setoptions(), 'options'.

-bprompt Prompt
  Sets a prompt to be used with echoed commands in batch files
  specified on the command line (-batch batchFile). Usually Prompt
  should end with a space, for example, -bprompt "HW 1> "..

-l Nlines
  This pre-defines option 'height' to be Nlines, where Nlines is either
  0 or an integer at least 5.  On windowed versions, this only controls
  the height of "dumb" plots. See subtopic 'options:"height"'.

-w Ncols
  This pre-defines option 'width' to be Ncols, where Ncols is an integer
  at least 20.  On windowed versions, this only controls the width of
  "dumb" plots. See subtopic 'options:"width"'.

-hist Nhist
  This pre-defines option 'history' to be Nhist, an integer >= 0.  This
  limits the number of previous command lines that can be saved and
  recalled to Nhist.  The default value is 100.  See subtopic
  'options:"history".

See also topics 'quitting', 'customize'.
```

2.202 length()

Usage:

```
length(x), x a vector, matrix, or array or any other type of variable.
```

Keywords: variables, null variables

```
                                Usage
length(x) computes the total number of elements in the vector, matrix,
or array x.  The value of length(x) is prod(dim(x)).

If x is a NULL variable, length(x) = 0.

If x is a GRAPH variable or a macro, length(x) = 1.

If x is a structure, length(x) is a structure.  If xi is component i of
x, the component i of length(x) has the same component name and value
```

```
length(xi).

                          Cross references
See also topics dim(), ndims(), ncomps(), 'NULL', 'graphs'.
```

2.203 lgamma()

Usage:

```
   lgamma(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                Usage
lgamma(x) returns the natural logarithm of the gamma function of the
elements of x, when x is a REAL scalar, vector, matrix or array.  The
result has the same shape as x.  When x is an integer, lgamma(x) =
log((x-1)!).

If any element of x is MISSING, so is the corresponding element of
lgamma(x).  If any element of x <= 0 or x > 2.5599833278516e305, the
corresonding element of lgamma(x) is MISSING.  In both cases a warning
message is printed.

When x is a structure, all of whose non-structure components are REAL,
lgamma(x) is a structure of the same shape and with the same component
names as x, with each non-structure component transformed by lgamma().

                          Cross references
See topic 'transformations' for more information on lgamma().
```

2.204 lineplot()

Usage:

```
   lineplot(x,y [,add:T,linetype:m, impulse:T] [,other graphics keyword
     phrases]), where x is a REAL vector or scalar, y is a REAL vector or
     matrix, and m >= 0 is an integer
   lineplot([Graph,] [x,y], keys:str), str a structure whose component
     names are graphics keywords
```

Keywords: plotting

```
                                Usage
lineplot(x,y) makes a connected line plot of the data in vector x and
vector or matrix y, drawing lines between the successive points.  When y
has several columns, each column is graphed separately with different
line types, solid, dashed, ..., repeating cyclically when there are
more columns than distinct line types.
```

Generally, lineplot() should be used only when the values in x are in increasing or decreasing order, although there are useful exceptions.

It is not an error when x or y is NULL; a warning message is printed and no plotting occurs.

Keywords 'linetype' and 'thickness'

lineplot(x,y,linetype:k,thickness:w), k > 0 an integer and w > 0 a REAL scalar draws lines of type k and width w. The defaults are k = 1 and w = 1. k < 0 is the same as abs(k) and k = 0 is the same as k = -1. The interpretation of k and w depend on the computer system on which MacAnova is running. See topic 'graph_keys'.

Structure argument

lineplot(Str), where Str is a structure with at least two REAL components, is equivalent to lineplot(Str[1], Str[2]). For example, lineplot(x,y) and lineplot(structure(x,y)) are equivalent. Any components of Str beyond the first two are ignored.

Variable LASTPLOT

lineplot() normally creates or replaces GRAPH variable LASTPLOT which encapsulates everything in the graph. In addition, if the graph was drawn in graphics window I, GRAPHWINDOWS[I] is made identical to LASTPLOT (I is always 1 in non-windowed DOS and Unix/Linux versions). You can suppress saving the plot information in LASTPLOT and GRAPHWINDOWS[I] by including 'keep:F' as an argument. See topics 'graphs' and 'graph_assign' for information on GRAPH variables and special variable GRAPHWINDOWS.

Graph variable argument

lineplot(Graph,x,y) or lineplot(Graph,Str), where Graph is a GRAPH variable, draws the plot encapsulated in Graph, adding to it new information.

Keywords 'add', 'impulses', 'lines' and 'symbols'

lineplot(x,y,add:T,...) is the same as lineplot(LASTPLOT,x,y,...) drawing the graph encapsulated in LASTPLOT, adding to it new information. An equivalent way to do this is addlines(x,y,...).

When option 'dumbplot' has been set False (see subtopic 'options:"dumbplot"'), the plot will be a low resolution plot unless 'dumb:F' is an argument.

lineplot(x,y,impulses:T) draws vertical lines to the points from the x-axis (y = 0 line), in addition to drawing connecting lines

lineplot(x,y,lines:F [,...]) is equivalent to plot(x,y [,...]).

lineplot(x,y,symbols:c [,...]) is equivalent to chplot(x,y,symbols:c, lines:T [,...]). In particular, when c = "###", points are labeled with the row number when y is a vector and the column number when ncols(y) > 1.

```
                          Short x argument
See topic 'graphs' for information on how a scalar or length 2 vector x
specifies equally spaced x-values, on how to save and print plots, and
on writing graphic information to a file.

                          Graph keywords
See topic 'graph_keys' for information on keywords 'title', 'xlab',
'ylab', 'xmin', 'xmax', 'ymin', 'ymax', 'logx', 'logy', 'xaxis',
'yaxis', 'dumb', 'add', 'file', 'linetype', 'thickness', 'silent' and
'notes'.

                          Keyword 'keys'
lineplot([Graph,] keys:structure(x:x,y:y [other keyword phrases)) is
equivalent to lineplot([Graph,] x,y [other keyword phrases]).  See
topic 'graph_keys' for details.

                          GRAPHWINDOWS
See topic 'graph_assign' for information on how to plot in graphics
window I by GRAPHWINDOWS[I] <- var, where var is a structure or GRAPH
variables.

                          Cross references
See also topics chplot(), plot(), showplot(), addchars(), addlines(),
addpoints(), tek(), tekx(), vt(), vtx().
```

2.205 list()

Usage:

```
list([invis:T]) or list(var1 [, var2, ...])
list([all:T, real:T or F, char:T or F, logic:T or F, macro:T or F,\
  struct:T or F, null:T or F, labeled:T or F, notes:T or F,\
  locked:T or F, keep:T, nrows:n1, ncols:n2, ndims:n3]); use F's only
  with all:T, n1, n2, n3 > 0 integers
```

Keywords: general

```
                                Usage
list() lists the name, type, and dimensions of all currently active
variables, including structures and macros, but excluding any temporary
or "invisible" variables (variables whose names start with '_').  The
maximum level of any factors is printed.  See factor() and topic
'variables:"invisible"'..

list(invis:T) does the same, but also includes temporary variables and
invisible variables.

list(var1, var2, ..., vark) gives the same information only for the
specified variables.

For a macro, list() also prints 'out-of-line' or 'in-line' depending on
whether or not it has been marked to be expanded out-of-line.  See
```

topics 'macros', macro().

For any variable, list() may also print any or all of 'labels', 'notes' or 'locked' depending on whether the variable has dimension lables, attached notes or is locked. See topics 'labels', 'notes', 'locks'.

Listing by attributes

list(size:T [,invis:T]) or list(var1,var2,...,vark,size:T) also lists the total size of each variable in bytes. In addition to the memory required for data in a variable, this total includes a fixed amount (172 bytes in one Linux implementation) for each symbol and each structure component. In addition, MacAnova prints the total size of all listed variables and the total of all memory currently used by MacAnova for variable and internal storage.

list(varType:T [,invis:T]) where varType is one of 'real', 'factor', 'logic', 'char', 'macro', 'struc', or 'null' specifies that all variables of the specified types are listed. More than one keyword phrase can appear but no variable names. For example, list(real:T, logic:T) will list all variables of type REAL or LOGICAL and list(factor:T) will list all variables that are factors.

list(all:T,varType1:F [,varType2:F...] [,invis:T]) lists all types except those specified. For example, list(all:T,macros:F) lists all objects except macros.

list(notes:T [,invis:T]) limits the listing to variables with notes. See topic 'notes'.

list(labeled:T [,invis:T]) limits the listing to variables with labels. See topic 'labels'.

list(locked:T [,invis:T) limits the listing to variables that are locked. See topic 'locks'.

list(nrows:r [,...]) lists all REAL, CHARACTER or LOGICAL variables with first dimension r.

list(ncols:c [,...]) lists all REAL, CHARACTER or LOGICAL with second dimension c. When c = 1, vectors are also listed. 'nrows:r' and 'ncols:c' can be used together.

list(ndims:d [,...]) all REAL, CHARACTER or LOGICAL variables with exactly d dimensions. For example, list(ndims:1) lists all vectors.

Keywords 'nrows', 'ncols' and 'ndims' can be used together with 'char:T', 'real:T', or 'logic:T' to limit which variables are listed.

Keywords phrases 'labeled:T', 'notes:T' and 'locked:T' can be used together with type and shape specifying keywords. They suppress listing variables that do not meet the additional restrictions.

Wildcard matching

```
list(Pattern ... [,invis:T]), where Pattern is a quoted string (but not
a CHARACTER variable) which contains one or more of the "wild card"
characters '*' and '?', lists only objects whose names match Pattern.

'*' will match any set of 0 or more consecutive characters of variable
names, and '?' will match any single character.

For example, list("x*") lists all variables whose names start with 'x',
list("*length") lists all variables whose names end in 'length', and
list("c*b???") lists all variables whose names start with 'c' and end
with 'b' followed by any 3 characters, say, "crybaby".  The last does
not match "crybabies", although "c*b???*" would.

list(pat:Pattern ... [,invis:T]) does the same, except Pattern may bea
CHARACTER scalar whose value is a pattern containing wild card
characters, not just a quoted string.

If a variable is "special" the type is preceded by '*'.  Currently the
only special variables are CLIPBOARD, SELECTION (GTK only) and
GRAPHWINDOWS.  See topics 'CLIPBOARD', 'GRAPHWINDOWS' and
'graph_assign'..

                        Examples of wildcard use
Examples:
  Cmd> list("*plot") # lists colplot but not plot1 or myplots
  Cmd> list("plot*") # lists plot1 but not colplot or myplots
  Cmd> list("*plot*") # lists all three.

                              Keyword 'keep'
list(...,keep:T [,invis:T]) suppresses the listing, but returns a
CHARACTER vector containing the names of the variables that would
otherwise have been listed; no information on type or dimensions is
returned.

                      Examples of selective listing
Example:
  Cmd> list("a*", real:T) # or list(pat:"a*", real:T)
will list all REAL variables whose names start with "a".

                              Cross references
See also delete(), listbrief(), dim().
```

2.206 listbrief()

Usage:

```
listbrief([invis:T]) or listbrief(var1 [, var2, ...])
listbrief([all:T, real:T or F, char:T or F, logic:T or F, macro:T or F,\
  struct:T or F, null:T or F, labeled:T or F, notes:T or F, \
  locked:T or F keep:T, nrows:n1, ncols:n2, ndims:n3]); use F's only
  with all:T, n1, n2, n3 > 0 integers
```

Keywords: general

```
                                Usage
listbrief() lists the names of currently active variables, including
structures and macros.  No information on type or dimension is given.

listbrief(invis:T) does the same, but also includes temporary variables
and variables whose names start with '_'.

listbrief(var1, var2, ..., vark) does the same for specified variables,
except that undefined variables in the list are identified.

listbrief(pat:Pattern [,invis:T]) lists variables whose names match the
value of quoted string or CHARACTER variable Pattern.  If Pattern is a
quoted string, 'pat:' may be ommitted.  See list() for details.

You can also use keywords 'real', 'factor', 'logic', 'char', 'macro',
'struc', 'null', 'labeled', 'notes', 'locked', 'all', 'keep', 'nrows',
'ncols', and 'ndims' as in list().

                            Cross references
See also delete().
```

2.207 loadUser()

Usage:

```
   loadUser(fileName [,reload:T or clear:T]), CHARACTER scalar fileName.
```

Keywords: general, control, files

```
                               Information
Help on loadUser() is in file userfun.hlp.  You can retrieve the help by

  Cmd> userfunhelp(loadUser)

It provides a complete description of loadUser() which you must call
prior to using an externally compiled user function.

                            Cross references
See also topics User() and 'user_fun' in file userfun.hlp.  Type
userfunhelp(User) or userfunhelp(user_fun).  Type userfunhelp() for a
complete list of topics related to user functions.
```

2.208 locks

Keywords: general, variables

```
                               Description
```

```
A locked variable is a variable that is protected from casual
destruction or modification.  You lock variables using lockvars() and
unlock them using unlockvars().  A few pre-defined variables and macros
such as PI, E, VERSION and redo() are automatically locked at start up.
Most other pre-defined constants and macros are not locked.

It is an error to attempt to assign a value to a locked variable or to a
subscript of a locked variable.

  Cmd> PI <- sqrt(2)
  ERROR: illegal assignment to locked variable near PI <-

  Cmd> a <- run(10); lockvars(a); a[3] <- PI
  ERROR: illegal to assign to subscript of a locked variable near
  a <- run(10); lockvars(a); a[3] <-

If you try to delete a locked variable, it is not deleted and a warning
message (suppressed by 'silent:T') is printed.

  Cmd> delete(a) # delete(a, silent:T) prints nothing
  WARNING: attempt to delete locked variable a

You can force deletion by keyword phrase 'lockedok:T' on delete.

  Cmd> delete(a,lockedok:T); print(a)
  ERROR: argument 1 (a) to print() is not defined

You can unlock a variable using unlockvars().

  Cmd> unlockvars(PI); delete(PI); print(PI)
  ERROR: argument 1 (PI) to print() is not defined

                         Testing locked variable
You can test whether variables are locked using islocked().

  Cmd> islocked(E, VERSION, MACROFILES, boxcox, redo)
  (1) T       T       F       F       T

                         Locked variable in file
When the header line on a data set or macro in a file readable by
read(), matread() or macroread() contains the work LOCKED, the value
returned by read(), matread() or macroread() is saved as a locked
variable.
  Cmd> doit <- macroread("macrofile.txt","doit")
  doit  MACRO LOCKED

  Cmd> list(doit)
  doit            MACRO  (in-line) (locked)

                          Unlockable variables
You cannot lock temporary variables (names starting with '@'), special
variables such as CLIPBOARD and GRAPHWINDOWS and a few other variables
such as LASTPLOT and LASTLINE.
```

```
  Cmd> lockvars(LASTLINE)
  ERROR: can't lock variable LASTLINE

                        GLM side effect variables
If you lock a GLM side effect variable such as RESIDUALS or STRMODEL, it
effectively blocks any more GLM commands such as regress(), anova() or
poisson() until the variable is unlocked or deleted.

                          Saving and restoring
save() and asciisave() save the locked/unlocked status of a variable, so
when the workspace file is restored, a locked variable is restored as a
locked variable.

restore(workspaceFile, delete:F) will not restore variables with the
same name as existing locked variables.  Unless the locked variable is a
REAL scalar and the value in the file is the same as the current value,
a warning message is printed.

                            Cross references
See also 'variables', unlockvars(), lockvars(), islocked().
```

2.209 lockvars()

Usage:

```
   lockvars(a [,b,c,...] [,silent:T]), a, b, c, ... arbitrary variables
     other than GRAPHWINDOWS, CLIPBOARD, and SELECTION
```

Keywords: general, variables

```
                                  Usage
lockvars(var1, var2, ...) marks variables var1, var2, ... as locked
variables.  This means they cannot be deleted or assigned to.  Any named
variables can be locked except for temporary variables (names beginning
with '@') and the "special" variables GRAPHWINDOWS, CLIPBOARD,
SELECTION.  A warning is printed if any argument is a built in function
name or is alreadly locked.  It is an error if any argument is an
expression or a function result.

lockvars(var1, var2, ..., silent:T) does the same except that no warning
message is printed.

When you save a locked variable (see save() and asciisave()), its locked
status is also saved so that when it is restored (see restore()), it
will still be locked.

Locking a variable does not protect it from destruction by restore().

                            Cross references
See also unlockvars(), delete(), 'variables'
```

2.210 log()

Usage:
```
log(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                Usage
log(x) returns the natural logarithm (base e log) of the elements of x,
when x is a REAL scalar, vector, matrix or array.  The result has the
same shape as x.

If any element of x is MISSING, so is the corresponding element of
log(x).  If any element of x <= 0, the corresonding element of log(x)
is MISSING.  In both cases a warning message is printed.

When x is a structure, all of whose non-structure components are REAL,
log(x) is a structure of the same shape and with the same component
names as x, with each non-structure component transformed by log().

                            Cross references
See topic 'transformations' for more information on log().
```

2.211 log10()

Usage:
```
log10(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                Usage
log10(x) returns the common logarithm (base 10 log) of the elements of
x, when x is a REAL scalar, vector, matrix or array.  The result has the
same shape as x.

If any element of x is MISSING, so is the corresponding element of
log10(x).  If any element of x <= 0, the corresonding element of
log10(x) is MISSING.  In both cases a warning message is printed.

When x is a structure, all of whose non-structure components are REAL,
log10(x) is a structure of the same shape and with the same component
names as x, with each non-structure component transformed by log10().

                            Cross references
See topic 'transformations' for more information on log10().
```

2.212 log2()

Usage:

```
log2(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                Usage
log2(x) returns base 2 log logarithm of the elements of x, when x is a
REAL scalar, vector, matrix or array.  The result has the same shape as
x.

If any element of x is MISSING, so is the corresponding element of
log2(x).  If any element of x <= 0, the corresonding element of
log2(x) is MISSING.  In both cases a warning message is printed.

When x is a structure, all of whose non-structure components are REAL,
log2(x) is a structure of the same shape and with the same component
names as x, with each non-structure component transformed by log2().

                            Cross references
See topic 'transformations' for more information on log2().
```

2.213 logic

Usage:

```
a && b, a || b, !a, where a and b are LOGICAL or structures with LOGICAL
  components
LOGICAL constants are T (True) and F (False)
```

Keywords: variables, syntax, logical variables, missing values, operations

```
                    Description of LOGICAL variables
Elements of a LOGICAL variable have only three possible values -- True,
False, and MISSING.  A LOGICAL variable may be a vector, matrix, or
array.

Logical values are printed as 'T' (True) and 'F' (False), the same
symbols as you use to enter them.  For example, 'a <- vector(T,F,F,T)'
creates a LOGICAL vector of length 4.

When used as the value of a keyword phrase, as in 'quiet:T', T and F can
usually be interpreted as 'yes' and 'no', respectively.

You can also create LOGICAL data as the result of comparing REAL,
LOGICAL or CHARACTER variables using the following comparison operators:

                        Comparison operators list
There are 6 comparison operators used with REAL, LOGICAL and CHARACTER
data and structures of such data.
```

```
 Comparison
  Operator  Precedence  Meaning
    a == b      8       Equal or same
    a != b      8       Not equal or different
    a < b       8       Less than
    a <= b      8       Less than or equal
    a > b       8       Greater than
    a >= b      8       Greater than or equal

                         Logical operators list
There are three purely operators used only with LOGICAL data.
   Logical
  Operator  Precedence  Meaning
   a || b       5       Logical Or (T||T,T||F,F||T are True, F||F False)
   a && b       6       Logical And (T&&T is True, T&&F,F&&T,F&&F False)
     !a         7       Logical Not (!T is False, !F is True)

                              Precedence
The precedence level in the lists of operators above affects the order
of evaluation when there is more than one operator in an expression.  An
operator with higher precedence is evaluated before one with lower
precedence.  For example, MacAnova interprets T && F || T && T as
(T && F) || (T && T) because '&&' has higher precedence (6) than '||'
(5).

See topic 'precedence' for a complete discussion of operator association
and precedence.

                           Comparison Operators
Comparison operators are most useful with REAL and CHARACTER data.  When
they are used with LOGICAL data, True and False are interpreted as 1 and
0, respectively, in the same way as with arithmetic operators +, -, *,
and ^ or ** (see 'arithmetic').  In particular F == F and T == T are
True and F == T and T < F are False.

Comparison operators do not "associate".  For example, an expression
like '3 < x <= 5' is meaningless and is an error.  Instead, you can use
'3 < x && x <= 5'.

As you would expect, the precedence of comparison operators is lower
than all arithmetic operations (see 'arithmetic') so that, for example,
3*4 == 14-2 is interpreted as (3*4) == (14-2) and is True.

                       Comparison of CHARACTER data
CHARACTER variables are compared using the ASCII collating sequence.
Most punctuation and all numerals are "less than" upper case letters
which in turn are "less than" lower case letters.  A space is "less
than" all printable characters.  Here is the explicit ordering starting
with space:
    !"#$%&'()*+,-./0123456789:;<=>?
   @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
   `abcdefghijklmnopqrstuvwxyz{|}~
On computers with extended character sets, the ordering is dependent on
```

the internal representation of the characters.

Comparisons with MISSING

Here's what happens when you make a comparison involving a MISSING value.

Operators '<', '<=', '>', or '>=': The result is MISSING.
Operator '==' : Result is True only when if both operands are MISSING
Operator '!=' : Result is True only when just one operand is MISSING.

Examples

Examples:

```
2 == 1, 3 != 3, 3 < 1, T == F, and T > 1 all have the value False
2 == 2, 3 != 2, 3 > -1, F == F, and 0 == F all have value True
"A" > "a", "A" != "A", and "MacAnova 2" == "MacAnova 3" are all False
```

Special characters

Because '<-' is the assignment operator, 'a<-5' will always be interpreted as "assign 5 to a" even if what is wanted is a comparison of a with -5. To obtain the latter a space must follow '<' as in 'a < -5'.

On Mac OS 9, you can use 'Option+<', 'Option+>' and 'Option+=' in place of '<=', '>=', and '!=', respectively.

Purely logical operators

You can use operators '&&', '||' and '!' (logical AND, OR and NOT) to combine several conditions to make a single condition or to specify the opposite of a condition. For example, (x > 1) && (x < 3) is True if and only if the value of x is greater than 1 and less than 4 and (x < 0) || (y < 0) is True if and only if one or both of x and y is negative. Also, !(x < 0) means the same as x >= 0.

If an operand of '||', '&&' or '!' is MISSING, so is the result .

The precedence of logical operators is lower than the precedence of arithmetic expressions (see 'arithmetic') and comparison operations. For example, 'x > 1 && x < 4' is interpreted as '(x > 1) && (x < 4)', and is True if and only if the value of x is greater than 1 and less than 4. Because the precedence of '!' is lower than the precedence of the comparison operators, expressions like '! a < b' are evaluated as '!(a < b)'. See also topic 'precedence'.

Examples:

```
F && T, F || F, and !T all have the value False
F || T and !F have the value True
```

Functions alltrue() and anytrue()

In contrast with the C programming language, both expressions that are combined with '&&' and '||' are always evaluated regardless of the value of the first expression. For example, in

```
(sqrt(2) < sqrt(3)) || (log(5) < log(6))
```

both log(5) and log(6) are evaluated although the final value of the expression (True) could have been determined once it was found that sqrt(2) < sqrt(3) was True.

```
alltrue() and anytrue() provide the C behavior.  For example
  Cmd> anytrue(sqrt(2) < sqrt(3), log(5) > log(6)) # value is True

and

  Cmd> alltrue(sqrt(4) < sqrt(2), log(5) < log(6)) # value is False

evaluate only the first arguments.

          Comparison and logical operations with non-scalars
MacAnova allows comparison of or logical combination of arrays of
different sizes entirely analogously to the way they can be combined
arithmetically by '+', '-', '*', '/', '^', and '%%'.  For instance, '2 <
run(3)' is vector(F,F,T).  See topic 'arithmetic' for details.

When one of the operands is a structure, each of its components is
combined with the other argument, producing a structure with the same
shape as the structure argument.  If both arguments are structures, they
must have the same shape and the corresponding components are combined.
In either case, all the components of a structure must have the same
type and all components must be compatible.  See topic 'structures'.
```

2.214 logistic()

Usage:

```
logistic([Model],n:Denom [, incr:T, offsets:vec, print:F or silent:T,\
  pvals:T, maxiter:m, epsilon:eps, coefs:F, problimit:smallVal]), Denom
  REAL scalar or vector > 0, vec a REAL vector, m an integer > 0, eps
  and smallVal small REAL scalars > 0.
```

Keywords: glm, regression, categorical data

```
                                  Usage
logistic(Model,n:Denom) computes a logistic regression fit of the model
specified in the CHARACTER variable Model.  If y is the response
variable in the model it must consist of integers y[i] >= 0.  Denom must
either be an integer scalar >= max(y) or a REAL vector of the same
length as y with Denom[i] >= y[i].  Estimation is by maximum likelihood
on the assumption that y[i] is binomial with Denom[i] trials (Denom
trials for scalar DENOM).

If either Denom or y contains non-integer values a warning message is
printed.

See topic 'models' for information on specifying Model.

                      Side effect variables created
logistic() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS,
DF, HII, DEPVNAME, TERMNAMES, and STRMODEL.  See topic 'glm'.  Without
keyword phrase 'inc:T' (see below), TERMNAMES has value vector("","",
```

...,"Overall model","ERROR1"), DF has value vector(0,0,...,ModelDF, ErrorDF) and SS has value vector(0,0,...,ModelDeviance,ErrorDeviance).

Output

If, say, Model is "y=x1+x2", an iterative algorithm is used to predict logit(E[y/Denom]) as a linear function of x1 and x2, where logit(p) = log(p/(1-p)). A two line Analysis of Deviance table is printed.

Line 1 is the difference 2*L(1) - 2*L(0), where L(0) is the log likelihood for a model with all coefficients 0 and L(1) is the maximized log likelihood for the model fit.

Line 2 is 2*L(2) - 2*L(1) where L(2) is the maximized log likelihood under a model fitting one parameter for every y[i]. Under certain conditions, the latter can be used to test the goodness of fit of the model using a chi-squared test.

Incremental deviances

logistic(Model,n:Denom,inc:T) computes the full logistic model and all partial models -- only a constant term, the constant and the first term, and so on. It prints an Analysis of Deviance table, with one line for each term, representing a difference 2*L(i) - 2*L(i-1) where L(i) is the maximumized log likely for a model including terms 1 through i, plus the deviance of the complete model labeled as "ERROR1". Each line except the last can be used in a chi-squared test to test the significance of the term on the assumption that the true model includes no later terms.

Omitting model

If you omit Model (logistic(,n:Denom ...)), the model from the most recent GLM command such as poisson() or anova(), or the model in CHARACTER variable STRMODEL is used.

Computations are carried out using iteratively reweighted least squares.

logistic(Model,n:Denom,...) is equivalent to glmfit(Model,n:Denom, dist:"binomial", link:"logit",...).

Problimit warning

If you get a warning message similar to the following

   WARNING: problimit = 1e-08 was hit by logistic() at least once

it usually indicates either the presence of an extreme outlier or a best fitting model in which many of the probabilities are almost exactly 0 or 1. The latter case may not represent any problem, since the fitted probabilities at these points will be 1e-8 or 1 - e-8. You can try reducing the threshold using keyword 'problimit' (see below), but you will probably just get the message again.

Other keyword phrases

Keyword phrase	Default	Meaning
		Keyword 'maxiter'
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting

```
                          Keyword 'epsilon'
  epsilon:eps    1e-6    Small positive REAL specifying relative error
                         in objective function (2*log likelihood)
                         required to end iteration

                        Keyword 'problimit'
  problimit:small 1e-8   Iteration is restricted so that no fitted
                         probabilities are < small or > 1 - small. Value
                         of small must be between 1e-15 and 0.0001.

                          Keyword 'offsets'
  offsets:OffVec none    Causes model to be fit to logit(p) to be
                         1*Offvec+Model, where OffVec is a REAL vector
                         the same length as response y.  Note OffVec is
                         in logit units.  See topic 'glm_keys' for more
                         details.

                           Keyword 'pvals'
  pvals:T or F      F    Nominal chi-squared P-values will be printed
                         for each deviance.  The default value can be
                         changed by setoptions(pvals:T).  See topics
                         setoptions(), 'options'.

               Keywords 'print', 'silent' and 'coefs'
See topic 'glm_keys' for information on keyword phrases print:F,
silent:T, coefs:F

                   Examples of the use of 'offsets'

  Cmd> logistic("y=x", n:15, offsets:3*x, inc:T, pvals:T)
The P value associated with x can be used to test the hypothesis H0:
beta1 = 3 in the model log(p/(1-p)) = beta0 + beta1*x.

  Cmd> logistic("y=1", n:20, offsets:rep(log(.25/(1-.25)),length(y)),\
       inc:T, pvals:T)
The P value associated with the CONSTANT term can be used to test H0: p
= .25, assuming y contains a random sample from a binomial distribution
with n = 20.
```

2.215 lowess()

Usage:

```
   result <- lowess(x, y [,xpred:xp] [,fract:f] [,iter:m] [, delta:Delta])
     REAL nonMISSING vectors x, y, xp, nondecreasing x, xp, length(x)=
     length(y), REAL scalars f and Delta, 0 < f <= 1, Delta >= 0, integer
     iter > 0; result is structure(x:x,y:yfit [,xpred:xp,ypred:yp])
```

Keywords: regression, descriptive statistics, plotting

```
                              Introduction
lowess() uses the LOWESS smoother algorithm to summarize the dependence
```

of a REAL vector y on a non-decreasing REAL vector x of the same length.

The assumed model is y = g(x) + e, where g(x) is a "smooth" unknown function of a predictor x and e is an random "error" with E[e] = 0. The smoothed output values are estimates of g(x).

Optionally lowess() also estimates E(yp|xp) = g(xp), where xp may differ from all x[i], assuming approximately linear dependence near xp.

LOWESS is a resistant (to outliers) locally linear smoother. See below for more information.

Usage

Result <- lowess(x,y) and Result <- lowess(structure(x,y)) use the LOWESS smoother to find a vector ys with ys[i] = smoothed y value corresponding to x[i]. x and y must be REAL vectors with the same length and with no MISSING elements. Also, x must be non-decreasing, that is x[i] <= x[i+1].

Result = structure(x:x, y:ys), where x is identical to the input x vector and ys is a REAL vector the same length as x and y.

Result <- lowess(x,y, xpred:xp) and Result <- lowess(structure(x,y), xpred:xp) do the same but also compute a vector yp of fitted values corresponding to REAL vector xp using weighted least squares. xp must have no MISSING values and must be nondecreasing.

With xpred:xp, Result = structure(x:x, y:y, xpred:xp, ypred:yp), where yp is a REAL vector the same length as xp with yp[i] the predicted value corresponding to xp[i]. When xp[i] = x[j], yp[i] = ys[j]. If xp[i] < x[1] or xp[i] > x[n], where n = length(x), yp[i] is computed by extrapolation using the straight line used to compute ys[1] or ys[n].

When there are tied x[i]'s, say x[i] = x[i+1] = ... x[i+k-1], ys[i] = ys[i+1] = ... = ys[i+k-1].

You can modify the behavior of lowess() using keywords 'fract', 'delta' and 'iter'.

When x is not sorted, you need to use lowess(sort(x),y[grade(x)]).

Keywords

The following summarizes the lowess() keyword phrases which modify the smoothing algorithm. All have REAL scalar values and can be used with 'xpred'.

Keyword	Value	Default	Description
fract:f	0 < f <= 1	2/3	Fraction of points used to compute each smoothed value
delta:Delta	0 <= Delta	x_range/100	x values within Delta of other x values have smoothed values computed by interpolation rather than additional regressions

```
  iter:m      integer m > 0  3        number of iterations of robust
                                      fitting
```

These defaults are the same as those in R.

The larger f is, the smoother ys will be. This is because each weighted regression is based on r data pairs, (x[j], y[j]), j = j1, j1+1, ..., j2 = j1+r-1, where r = max(2,round(f*n)).

Non-zero values of delta can result in faster fitting. See below.

When there are tied x's, the value of ys can differ for different orderings of the ties x's.

'fract' and 'delta' can be abbreviated to 'f' and 'del', respectively.

Method

lowess() uses iteratively reweighted least squares to fit a straight line to data near each x[i]. The weight for x[j] depends on |x[j]-x[i]| and, after the first iteration, on |y[j]-y_smooth[j]|, where y_smooth[j] is the smoothed value from the previous iteration. Larger distances and larger residuals result in lower weights. See the reference below for details on the weights used.

ys[i] is computed from r pairs, (x[j], y[j]), j=j1,...,j2 = j1 + r - 1 and r = max(2, round(f*n)) with x[j1] <= x[i] <= x[j2], where j1 <= n+1-r is chosen to minimize max(x[i] - x[j1], x[j2] - x[i]).

When these x[j]'s don't vary enough reliably to estimate a slope, ys[i] is a weighted average of y[j], j1 <= j <= j2.

Some shortcuts are made:

When there are k tied x's, that is x[i] = x[i+1] = ... = x[i+k-1], the regression computation is done only for x[i] and then ys[i+1], ..., y[i+k-1] are set equal to ys[i].

When Delta > 0 and there are near ties, that is x[i1]-x[i] <= Delta, i1 > i, ys[i1] is computed by linear interpolation between ys[i] and ys[i2] where x[i2] - x[i] > Delta (or between ys[i] and ys[n] when x[n]-x[i] < Delta.

Reference

You can find more information about the lowess() method in the following reference:

Cleveland, W. S. (1979) Robust locally weighted regression and smoothing scatterplots. J. Amer. Statist. Assoc. 74, 829-836.

The code used is adapted from ratfor subroutines lowess() and lowest() by W. S. Cleveland obtained from statlib.

Example

```
  Cmd> irisdata <- getdata(irisdata, quiet:T) # Fisher Iris Data
```

```
  Read from file "/usr/libs/macanova/auxfiles/macanova.dat"

  Cmd> variety <- irisdata[,1]

  Cmd> x <- irisdata[variety==1,2] # I. setosa sepal length

  Cmd> y <- irisdata[variety==1,3] # I. setosa sepal width

  Cmd> plot(x,y,symbols:"\1",show:F)

  Cmd> addlines(lowess(sort(x),y[grade(x)]), show:F)

  Cmd> showplot(dumb:T,width:70,xlab:"Sepal length",ylab:"Sepal width",\
       title:"Iris Setosa Sepal width vs Sepal length with smooth")
                Iris Setosa Sepal width vs Sepal length with smooth
             +---+-------+------+------+-------+------+------+-------++
           4.5+                                                    o    +
              |                                                         |
              |                                              o          |
              |                                    o                    |
    S       4+                                                   .....o+
    e         |                                          o     ....      |
    p         |                                  o      o ..o.....     o    |
    a         |           o          o  o     .......                      |
    l       3.5+                        o ..o.  o          o              +
              |           o      o     .o.  o   o      o                  |
    w         |                 ........ o   o                            |
    i         |   o ......o..o.         o                                |
    d       3+o..o.      o      o   o  o                                  +
    t         |   o                                                       |
    h         |                                                           |
              |                                                           |
              |                                                           |
           2.5+                                                           +
              |       o                                                   |
             +---+-------+------+------+-------+------+------+-------++
                4.4     4.6    4.8     5      5.2    5.4    5.6    5.8
                                  Sepal length

                            Cross references
See also regress().
```

2.216 macintosh

Keywords: general

Summary of features special to the Mac OS X version.

MacAnova for Macintosh OS X is built using Carapace (see help topic 'carapace'), so all of the Carapace features are present on the

Macintosh.

Menus are slightly different on Mac OS X to conform with standard Macintosh guidelines. In particular, the About menu item and the Quit menu item are under the program menu rather than the Help and File menus.

The standard search paths in DATAPATHS include a "user" directory (folder) and an "application" directory (see topic 'DATAPATHS'). On Mac OS X, the user directory is "~/Library/Application Support/MacAnova" where ~ is the user's home directory. The application directory is the SharedSupport folder that came with MacAnova; it must be in the same folder as the MacAnova application.

MacAnova reads from many files (macros, help, etc.). For example, MacAnova.mac.txt is the standard macro file. If you put your own modified copy of MacAnova.mac.txt into the user directory, then MacAnova will read from your file instead of its own copy, because the user directory comes before the application directory in the search order.

2.217 mac_classic

Keywords: general

Summary of features special to the Mac OS 9 version.

Windows

There can be up to nine command/output windows. Everything you type and all character output goes in the frontmost one. You can create a new window using New Window on the Windows menu or Open on the File menu. The latter loads the window with the contents of the file selected. You can switch between windows, close them or hide them using the Windows menu. You can print all of a window or a selection from a window by selecting Print Window or Print Selection on the File menu. You can save their contents as files on disk by selecting Save Window or Save Window As on the File menu.

There are eight high resolution graphics windows plus two panel windows, Panel 1-4 and Panel 5-8, which display the other graph windows in miniature. You can switch between graphics windows, close them or hide them using the Windows menu. Any graphics window including the panel windows can be printed using Print Graph... on the File menu or copied to the clipboard for pasting into the Scrapbook or other application using Copy on the Edit menu. You can direct a plot to graph window i by plotting keyword phrase 'window:i', where 1 <= i <= 8. 'window:0' puts the graph in the window most recently written in.

You can display any graph window by pressing Command+1, Command+2, ..., Command+8 or Command+F1, ..., Command+F8, and display panel windows by Command+G. Command+G also switches to the other panel window when a

panel window is displayed.

All windows have a close box in the upper left corner, a resizing box in the lower right corner, and a zoom box in the upper right corner. Output windows also have a scroll bar for moving backward and forward through output.

Executing a command

MacAnova recognizes a command as being ready to execute only if the cursor is at the end of the line when you hit Return, or if you hit Enter (see next paragraph). Of course, as in all versions, the line will not be accepted as complete if there is any '{' unmatched by a corresponding '}' or if there is an unfinished quoted string started by '"'.

Enter key

The Enter key is not equivalent to Return but might be considered an "execute" key. It behaves differently depending on whether you have selected text before the prompt in the output window. Command+Return or Shift+Return are both equivalent to Enter.

(a) If nothing is selected with the mouse in the output window, Enter is equivalent to moving the cursor to the end of the command line and pressing Return. This causes the command on that line to be executed unless there is an incomplete quoted string or unbalanced curly brackets ({...}). This is particularly useful when editing a command about to be executed. No matter where in the line the cursor is, Enter causes the edited command to be executed, while Return does not initiate execution unless the cursor is at the end of the line

(b) If you have used the mouse to select text before the current prompt, Enter causes it to be copied to the end of the current command line, followed by Return. Except when there are unbalanced curly brackets or an incomplete quoted string, the current command line is then executed. This makes for very easy re-execution of commands, possibly after editing them in place.

Item Copy and Execute (or Execute, when nothing is selected in the output window) on the Edit menu does the same thing as Enter, as does pressing Command+\ or key F6.

History of previous commands

Typed commands are automatically saved in an internal "history" list. You can move through this list, inserting previous commands after the prompt, using items Up History and Down History on the Edit menu.

By default, MacAnova saves the most recent 100 lines. To change this, say, to 150, type 'setoptions(history:150)'. See topics setoptions(), 'options'.

Up and down history

Pressing F7 or the key combination Option+up-arrow is equivalent to selecting menu item Up History. Pressing F8 or Option+down-arrow is

equivalent to selecting Down History.

Command+Option+up-arrow retrieves the oldest saved command.
Command+Option+down-arrow reinserts whatever you had originally typed,
if anything, before recalling earlier commands.

Moving around the window

Arrow keys move the cursor as you would expect. Command+B (Back) and
Command+F (Forward) are equivalent to the left- and right-arrow keys.
There is no equivalent to the up- and down-arrow keys.

Pressing the Option key while pressing a left- or right-arrow key moves
backward or forward one "name" (stretch of characters that are legal in
names). Pressing the Option key while pressing an up- or down-arrow key
inserts previous commands at the prompt similar to items Up History and
Down History on the Edit menu.

Pressing the Command key while pressing an arrow key moves the cursor to
the start or end of the line (left- or right-arrow) or the top or bottom
of the window (up- or down-arrow). Repeated use of Command+up or
Command+down-arrow scrolls through the window, moving the cursor as it
goes.

Pressing the Shift key while using an arrow key selects whatever is
between where you start and where you end.

Windows menu shortcuts

Command+A (Go To Prompt on the Windows menu) moves the cursor to the
start of the current command line, just after the current prompt.

Command+E (Go To End on the Windows menu) or End on suitable keyboards
moves it to the end of the current command line at the very end of
window.

Command+T (Scroll To Top on the Windows menu) or Home scrolls to the Top
of the current command/output window. Command+U (or Page Up) scrools
back a screenful. Command+D (or Page Down) scrolls forward a screenful.
These do not move the cursor.

Specifying file names

When you use "" as the file name in any command requiring one (for
example vecread("")), the usual Macintosh scrolling dialog box lets you
select the file. You can also use an explicit file or "path" name. In
the latter case, if the name does not contain ':' (which would identify
it as a "path" name), MacAnova will look for it first in the default
Folder (see topic 'files') and then in the Folders specified in pre-
defined CHARACTER vector DATAPATHS. See topics 'file_names',
'DATAPATHS', adddatapath(), 'customize'.

Help

Selecting item Help on the Apple menu is equivalent to typing 'help()'.
If a word, say 'matrix', is selected in the window, item Help is
equivalent to 'help("matrix"). Command+H or the Help key does the same.

Interrupting

Command+. (period) or Command+I may be used to interrupt an operation or output. Depending on the operation it may not be recognized immediately.

File menu

Open (Command+O) creates a new output window and reads a file into it. It might, for example contain output from a previous MacAnova session.

Save (Command+S) and Save As write the current command/output window to a file.

Page Setup and Print/Print Selection/Print Graph (Command+P) do what you would think they should do.

Interrupt (Command+I) is equivalent to pressing Command+. (period) (see above).

Go On resumes computing after a graphing command with keyword phrase 'pause:T'. See topic 'graph_keys'.

Save Workspace (Command+K) and Save Workspace As invoke save() (asciisave() if asciisave() was used previously). See save() and asciisave().

Open Batch File (Command+Option+O) is equivalent to 'batch("")'. See batch().

Spool Output to File (Command+Option+S) is equivalent to 'spool("")'. If a spool file has previously been specified this menu item will be Stop Spooling or Resume Spooling and is equivalent to 'spool()'.

Edit menu

In the output/command window you can use Undo/Redo (Command+Z), Cut (Command+X), Copy (Command+C), and Paste (Command+V) in the usual way. For a graph window only Copy is active.

Copy to End (Command+/) copies the current selection to the end of the command line without putting it on the Clipboard.

Copy and Execute/Execute (Command+\) is equivalent to pressing Enter (see above) in the output command window.

Up History inserts the previously typed command after the prompt. Repeated selection of Up History successively inserts older and older commands. It is equivalent to pressing the Option and up-arrow keys.

Down History moves forward through previously saved commands, inserting them after the prompt. It is equivalent to pressing the Option and down-arrow keys.

Function keys F1, F2, F3, F4, F5, F6, F7 and F8 are equivalent to

Undo/Redo, Cut, Copy, Paste, Copy To End, Execute (Copy and Execute), Up History and Down History, respectively. Enter, Shift+Return and Command+Return are additional keyboard short cuts for Execute.

Windows menu

This menu allows you to close or hide a window (Close and Hide), create a new output/command window (New Window or Command+N), select a graph window (Graph 1, Graph 2, ... Graph 8, Panel 1-4 and Panel 5-8), select an output command window by name, and move around the output/command window (Scroll To Top, Go To End, Go To Prompt, Page Up, Page Down).

Command menu

The last 8 items are pre-defioned commands that are inserted in the output/command window. You can also select them by pressing Command+ +Option+1, ..., Command+Option+8.

Edit Commands... allows you to edit or replace any or all of the 8 commands. Since each command can be a macro, this allows a great deal of flexibility.

Options menu

Significant Digits, Output Formats, Random # Seeds, Angle Units, GLM Options, Batch Options and Other Options allow you to set many of the items that can be changed by setoptions().

Font menu

Size allows you to change the font size of the text in the current output/command window.

The remaining items are the names of fonts you can select for the text in the current output/command window. You should probably restrict your choices to non-proportional (equal character width) fonts such as Monaco or Courier. The default font is McAOVMonaco 9, a modified form of Monaco 9. Courier 18 may be preferable for use with an overhead projector. You can also change fonts using setoptions() using keywords 'font' and 'fontsize'. See topic 'options'.

Other information

MacAnova can "background", that is it will continue running when you switch to another application under System 7 or when you are using Multifinder under earlier systems.

Option+'<', Option+'>' and Option+'=' are recognized as equivalent to '<=', '>=', and '!=', respectively. Option+'\' and Option+'|' (Option+Shift-'\') are recognized as equivalent to '<<' and '>>', respectively.

The shell() command and special treatment of lines starting with '!' is not available on the Macintosh. On System 7 or later, you can run other programs just be starting them. On SYstem 6 you can do the same using Multifinder. On all systems you can run Desk Accessories from within MacAnova.

```
Files produced by save() in all Macintosh versions 3.xx and 4.xx, but
not 2.xx, can be restored.  However, files produced by asciisave()
should restore correctly.

Color is not supported under MacAnova except in so far as the system
provides it automatically for all applications.
```

2.218 macro()

Usage:

```
   macro(text [, dollars:T, inline:T or F ,notes:Notes]), text a CHARACTER
     scalar, Notes CHARACTER scalar or vector
```

Keywords: macros, control, syntax

```
                          Usage with example
macro(Text) creates a macro from the commands contained in the CHARACTER
variable or quoted string Text.  Text can also be an existing macro.

Here is a short example of the use of macro().

  Cmd> vhat <-\
    macro("@x <- argvalue($1,\"argument 1\",\"real matrix nonmissing\")
    @n <- nrows(@x); @p <- ncols(@x) #get dimensions
    @x <- @x - sum(@x)/@n # compute residuals from mean
    @v <- if (@n > 1){(@x %c% @x)/(@n*(@n-1))}else{
        matrix(rep(0,@p*@p),@p)} # 0 matrix when n = 1
    delete(@n,@x,@p) # cleanup
    @v")

vhat(x) returns S/n, where S is the sample variance/covariance matrix of
matrix x.  See topics 'macros', 'macro_syntax' and argvalue() for
interpreting the text.

                        Quotes and backslashes
When you use macro() to define a macro, you should type any quotes
'"' as '\"', as in the example.  Similarly to include a backslash '\'
you must type '\\'.  In particular, quoted quotes in a macro must
be typed as '\\\"' as in
  Cmd> quotedecho <- macro("print(\"\\\"$0\\\"\")")

  Cmd> quotedecho(The answer is 42)
  "The answer is 42"

                          Keyword 'dollars'
macro(Text,dollars:T) creates a macro from Text, adding "$$" to every
temporary name (name starting with '@') that (a) is not in a quoted
string, (b) is not in a comment starting with '#' and (c) does not
already end in "$$".  When the macro is executed, "$$" is replaced by a
2 digit number unique to the particular macro invocation so that the
actual name is unique to the macro.  The length of any temporary name
```

```
you use must be no more than 10 characters, counting '@'.  The following
appends '$$' to all temporary variable names in vhat:

  Cmd> vhat <- macro(vhat, dollars:T)

The first line of vhat now starts '@x$$ <-' instead of '@x <-.  See
topic 'macro_syntax' for more information about using '$$' in macros.

                              Keyword 'notes'
You can attach explanatory notes to a macro using keyword phrase
'notes:Notes', where Notes is a CHARACTER vector.  See topic 'notes' for
details.

                              Keyword 'inline'
macro(Text,inline:F [,dollars:T]) marks the macro being created as one
to be expanded out-of-line instead of having the expansion inserted into
the input line.  This primarily means that the macro will be freshly
expanded every time it is encountered, even on multiple trips through a
loop.  An instance of an in-line macro is expanded just once.  See topic
'macros'.  inline:T marks the macro as to be expanded in-line, even if
the value of option 'inline' is False.  See subtopic 'options:"inline"'.
You can mark an existing macro myMacro() to be expanded out-of-line by

  Cmd> myMacro <- macro(myMacro, inline:F)

                           Nonstandard characters
Within quotes ("..."), any characters other than "\n" and "\t" whose
ASCII codes are either 127 or less than 32, are replaced by their
escaped octal representation of form \nnn.  For example,
  Cmd> myplot <- macro("chplot($1,$2,symbols:\"\001\",$K)")
  Cmd> myplot <- macro("chplot($1,$2,symbols:\"\x01\",$K)")
and
  Cmd> myplot <- macro("chplot($1,$2,symbols:\"\\001\",$K)")
are completely equivalent.

Non-standard characters with ASCII codes >= 128 are not treated
specially.  For example,
  Cmd> myplot <- macro("chplot($1,$2,symbols:\"\201\",$K)")
  Cmd> myplot <- macro("chplot($1,$2,symbols:\"\x81\",$K)")
are equivalent, producing a macro containing whatever character has
ASCII code 129, which has a computer specific interpretation.

                       Cross references_topic_macros
See topic 'macros' for details on writing macros, including the use of
special symbols '$0, '$1', '$2', ..., '$N', '$V', '$v', '$K', '$k', '$S'
and '$$'.

                                 Examples
Examples:
  Cmd> median <- macro("describe($1,median:T)")
  Cmd> myread <- macro("matrix(vecread(\"$1\"),$2)'") #note transpose
  Cmd> greetings <- macro("print(\"\\\"Hello\\\"\")")
  Cmd> xlogx1 <- macro("@x <- $1; @x*log(@x)")
```

```
  Cmd> xlogx2 <- macro("@x$$ <- $1; @x$$*log(@x$$)")
  Cmd> xlogx3 <- macro("@x <- $1; @x*log(@x)", dollars:T)

median(x) would compute the medians of the columns of x.  See
describe().

y <- myread(family.dat,23) would create a matrix with 23 columns from
data from file family.dat, assumed to have a total of n*23 data items
arranged in n rows.

greetings() prints "Hello", complete with the quotation marks.

xlogx1(x), xlogx2(x) and xlogx3 all compute x*log(x).  However, use of
xlogx1 might lead to a problem if some other macro also used @x as a
temporary variable.  xlogx2 avoids this problem because the temporary
variable name @x$$ will be expanded to @x51, say, a name that should not
conflict with any other temporary name.  xlogx3 is identical to xlogx2
because each instance of @x is converted to @x$$.

                          Cross references
See also topics 'macros', 'macro_syntax', macroread(), macrowrite().
```

2.219 macro_files

Keywords: macros, files, input, output

```
                          Introduction
This topic describes the format of a file to be read by macroread().
See topics 'data_files', 'vecread_file' and 'matread_file' for
information on data files, and topic 'files' for more technical
information on file names, default directories or folders, and
abbreviated file names of the form "~/filename".

                  General description of file format
A file that can be read by macroread() must be a plain text or ascii
file.  If you create it in a word processor, be sure to save it as a
text or ascii file.  On the file can be any number of macros, each with
a header line, followed by optional comment lines and macro text lines.
The header line must contain keyword MACRO and may contain one or more
keywords INLINE, OUTOFLINE, NOTES, ENDED or DOLLARS.  MACRO must be the
first keyword but the order of others is irrelevant.

                          Formats for macros
There are two possible formats for macros that can be read by
macroread().  Both can include descriptive notes (see topic 'notes').

Form A is terminated by a special line:
   Name MACRO [other keywords]
   ) 0 or more descriptive comment lines starting with ')'
   ) .......
   Line 1 of macro
```

```
   . . . . .
   Line Nlines of macro
   %Name%
```

Form B, requires a count of the lines and is considered obsolete:

```
   Name Nlines MACRO [other keywords]
   ) 0 or more descriptive comment lines starting with ')'
   ) .......
   Line 1 of macro
   . . . . .
   Line Nlines of macro
```

Name is the macro name to be searched for. Case is ignored in searching for the name, so that macroread(FileName,"mymacro") will find mymacro(), MyMacro(), or MYMACRO(), for example.

The text of the macro immediately follows the header and comment lines.

In form A, the line immediately after the last line of macro text must be %macroname%, where macroname must be identical, including use of upper and lower case letters, to the name on the name line.

In form B, Nlines must be an integer and there must be exactly Nlines of text. The count includes lines starting with "#" but not the descriptive comment lines, if any.

It is good practice to include both header comment lines starting with ")" and macro comment lines starting with "#" describing the usage of the macro. See also macrousage().

Empty macro

It is permissible for there to be no lines of macro text (%macroname% immediately following comment lines for form A or Nlines = 0 for Form B). When macroread() reads such a "macro", only the descriptive comments are printed (if 'quiet:T' is not present). It is a useful convention to make the first macro on a file have 0 length with the header describing the contents of the file, since then 'macroread(FileName)' will print the contents.

Optional keywords on the name line

OUTOFLINE:
When the name line is of the form

```
   Name MACRO OUTOFLINE [other keywords]
```

or

```
   Name MACRO Nlines OUTOFLINE [other keywords]
```

the macro will be marked so that it will always be expanded out-of-line. When INLINE appears instead of OUTOFLINE, it will not be so marked. Simply OUT or IN can be used instead of OUTOFLINE or INLINE. Since in-line expansion is the default, INLINE is never required.

NOTES:
When the name line is of the form

```
   Name MACRO NOTES [other keywords]
```

```
or
   Name MACRO Nlines NOTES [other keywords]
the macro should be followed by a data set named "Name$NOTES" containing
a CHARACTER vector of descriptive usage notes.  See topic 'matread_file'
for the format.

DOLLARS:
When the name line is of the form
   Name MACRO DOLLARS [other keywords]
or
   Name MACRO Nlines DOLLARS [other keywords]
'$$' will be appended to any temporary variable names (names starting
with '@') that don't already end with '$$'.  When DOLLARS is on the name
line of macro myMacro(), macroread(fileName,"myMacro") is equivalent to
macro(macroread(fileName,"mymacro"),dollars:T).  See topics macro() and
'macro_syntax'.

ENDED:
When a form B name line has the form
   Name MACRO Nlines ENDED [other keywords]
it signals that the macro is followed by a line starting '%macroname%',
where 'macroname' exactly matches the name on the name line.  In some
cases, this can prevent read(), matread() or macroread() from
erroneously recognizing a line of the macro as the header for another
macro.  ENDED may also be used on form A macros, but is unnecessary
since they are required to have an ending line.

LOCKED:
When the name line is of the form
   Name MACRO LOCKED [other keywords]
or
   Name MACRO Nlines LOCKED [other keywords]
and the result of macroread() or read() is assigned to create a macro,
that macro will be locked.  See topic 'locks'.

  Cmd> doit <- macroread("macrofile.txt","doit")
  doit  MACRO LOCKED

  Cmd> list(doit)
  doit            MACRO  (in-line) (locked)

                           End-of-macros line
A line starting _E_N_D_O_F_M_A_C_R_O_S_ terminates the reading of macros
or data from a file.  You can put help information for the macros after
this line.  See file macanova.hlp for a description of the required
format for the help that would follow this line.

                              Example file
Example of macro file containing a 0 length, information only, macro and
three genuine macros:

   info      MACRO
   ) This file contains macros median(), rm(), and means()
```

```
%info%

median    MACRO
) median(x) computes medians of columns of x
# usage: median(x) [it's helpful to include usage in body of macro]
describe($1,median:T)
%median%

rm        2 MACRO
) rm(a,b,...) alias for delete for Unix/Linux lovers
# usage: rm(a,b,...) equivalent to delete(a,b,...)
delete($0)

means      MACRO NOTES DOLLARS
) means(x1,x2,...) computes means of vector arguments
# usage: means(x1,x2,...) computes means of vector arguments
@args <- structure($0) # make structure of all arguments
for(@i,1,$N){
    if(!isvector(@args[@i])){
        error("Arguments to macro $S must be vectors")
    }
}
@result <- rep(0,$N) #create vector of right length
for(@i,1,$N){
    @result[@i] <- sum(@args[@i])/nrows(@args[@i])
}
@result
%means%

means$NOTES  CHARACTER
) Two lines of usage notes for macro means()
means(x1,x2,...) computes means of its vector arguments, returning
them in a vector.
%means$NOTES%

_E_N_D_O_F_M_A_C_R_O_S_
```

```
The _E_N_D_O_F_M_A_C_R_O_S_ line is optional but recommended.  It must
be used if anything else other than data sets, for example help
information, follows the macros in the file.

Note: Form A was new with Version 4.04 of MacAnova.  Keywords ENDED and
DOLLARS were new in February 1999.
```

2.220 macro_syntax

Keywords: macros, syntax, control

```
                              Introduction
This topic presumes familiarity with topic 'macros'.  It describes
the use of comments in macros, how values are returned, the special
```

symbols that may be used in macros and gives some tips on writing robust and efficient macros.

Topics include Macro Self-Documentation, The Value of a Macro, Referencing Macro Arguments, Use of Temporary Variables, Expansion of Other Special Symbols Containing '$', Recursive Use of Macros, Tips For Good Macro Writing, and Functions Useful in Macros

Macro Self-Documentation

Often the first few lines of a macro are comments ("#" at the start of the line) which describe how the macro is used. You can use command macrousage() to print such comment lines. When you write a macro, it is good practice to include such lines.

The value of a macro

The last expression appearing in a macro is returned as its value. If this expression is the name of an "invisible" variable (name starting with '_' or '@_') the macro value can be assigned or used in an expression but will not be printed. If no value is to be returned, the macro should end with ';;' which ensures the value returned is NULL.

In addition, you can use 'return(Value)' almost anywhere in a macro to immediately leave the macro, returning Value as the value of the macro. Value may be a constant, expression or variable. 'return()' and 'return' without parentheses are equivalent to 'return(NULL)'. See topic 'return'.

See below for the use of delete() with keyword phrase return:T to return a temporary variable as value.

Referencing macro arguments

In the text of a macro the place holders '$1', '$2', ... are used to refer to argument 1, argument 2, ... When the macro is executed, MacAnova examines its text and substitutes the appropriate argument, almost exactly as typed, for any such place holder. It then executes the commands as if they had been typed at the keyboard. For example, if the first argument to a macro is '3+4', '@a <- $1' gets expanded to '@a <- 3+4'.

When a macro references a missing argument (for example, $3 when there are only two arguments) it usually terminates immediately with an error message. When, however, a missing argument appears in a quoted string ('print("$3")' when there are only two arguments) it is "expanded" to nothing ('print("")'). When you use place holders of the form '$01', '$02', ..., with a leading 0 for the number, then a missing argument is expanded to NULL outside of a quoted string (see topic 'NULL'). This may or may not result in an error message if the argument is used, but you can test the value using one or more of the isxxxx() functions (see below).

Quoted string argument

When an argument which is itself a quoted string, possibly containing backslashes ('\'), is referenced in the macro as part of a quoted

string, then all instances of '"' and '\' in the argument are changed to '\"' and '\\'. For example, in a macro, 'print("$1")' expands to 'print("3+4")' when the first argument is '3+4' but expands to 'print("\"foo\"")' when the first argument is '"foo"'.

Use of temporary variables

It is good practice to use temporary variables (names starting with '@') in macros so they will be deleted automatically just before the next prompt. You can also append '$$' to names to create a name that should be unique to the macro (see below). When you use 'dollars:T' as an additional argument to macro() in creating a macro, '$$' will be automatically appended to all temporary names not already ending in '$$'. This also happens when a macro with keyword DOLLARS on its header line is read from a file.

Because temporary variables are not deleted until the next prompt, it is often a good idea to delete them before exiting a macro, except for a result being returned, of course. To delete a variable, say '@result', whose value is to be returned, the last command in the macro should be delete(@result,return:T). @result will be deleted but its value will still be returned. See delete().

Invisible variable

Occasionally you may want to create a non-temporary "invisible" variable, that is, one that is not normally listed by list() or listbrief(). You do this by assigning a name starting with '_' such as '_x'. One use for this is to create a variable to be used by several related macros but not directly referenced by the user. See also list() and listbrief(). A temporary variable is "invisible" if its name starts with '@_', say '@_x'. See topic 'variables:"invisible"'.

You can return an "invisible" result by

```
delete(@result,return:T,invisible:T)
```

Expansion of other special symbols containing '$'

In a macro, certain symbols starting with '$' have a special meaning. Although they would be errors outside a macro, in a macro these symbols are "expanded", that is, something is substituted for them in the text of the macro. In brief, these symbols are as follows:

Symbol	Expanded value
$0	The complete comma-separated list of macro arguments.
$N	The number of macro arguments = 1 + (number of separating commas) except when the agument list is empty, when the value is 0
$V	A comma separated list of all macro arguments that are not keyword phrases.
$v	The number of macro arguments that are not keyword phrases.
$K	A comma separated list of all macro arguments that are keyword phrases.
$k	The number of macro arguments that are keyword phrases.
$A	A CHARACTER vector whose elements are the character strings

```
            specifying the arguments, that is, vector("$1","$2",...).
  $S        The name of the macro.
  $$        Expands to a unique 2 digit number, 00, 01, ..., 49 in out-
            of-line macros and 50, 51, ..., 99 in in-line macros.

$N, $V, $v, $K, $k, $S and $A are not treated specially when they are
preceded or followed by a character that could be part of a variable
name.  For example, str$N would be interpreted as component N of
structure str (see 'structures') and $Na would not be changed and would
probably result in an error.

                            Dollars details
Here are fuller explanations or examples of the use of these special
symbols.  In all the following we assume the symbols are in a macro
invoked by mymacro(x, title:"Hello Dolly",run(5),new:T).

$0 (full list of arguments):
  In the example $0 is replaced by 'x,title:"Hello Dolly",run(5),new:T'.
  $0 is particularly useful in defining "aliases" of standard commands.
  For example,
   Cmd> dir <- macro("list($0)")
  defines macro dir() which is used identically to list().

  When $0 appears in a quoted string, all instances of '"' or '\' in any
  macro argument are prefixed by '\'.  In the example, "$0" is replaced
  by "x, title:\"Hello Dolly\",run(5),new:T".

$N (number of arguments):
  In the example, $N is replaced by '4'.

$V (list of non keyword arguments):
  In the example, $V is replaced by 'x,run(5)'.  When $V is used within
  double quotes, any instances of '"' or '\' are expanded as for $0.

$v (number of non keyword arguments):
  In the example, $v is replaced by '2'.

$K (list of keyword arguments):
  In the example, $K is replaced by 'title:"Hello Dolly",new:T'.  This
  is useful for passing on all the keywords to a function invoked by the
  macro.  See pre-defined macro colplot() for an example of the use of
  $K.  When $K is used within double quotes, any instances of '"' or '\'
  are expanded as for $0.

$k (number of keyword arguments):
  In the example, $k is replaced by '2'.

$A (CHARACTER vector of all arguments in quotes):
  In the example $A is replaced by vector("x","title:\"Hello Dolly\"",
  "run(5)","new:T").  To print argument 3 to a macro as it appeared in
  the call, instead of print("$3"), you might use print($A[3]).  In
  the example this would print 'run(5)'.  Any instances of '"' or '\' in
  the arguments are replaced by '\"' and '\\'.  When $A is part of a
```

```
  quoted string, it is not expanded.

$S (name of macro):
  In the example, $S is replaced by 'mymacro' whether or not it is in
  quotes.

$$ (number unique to macro expansion):
  This expands to a unique two digit number, even in a quoted string.
  The particular integer that replaces $$ remains the same throughout an
  invocation of the macro, but is incremented by 1 for each macro in
  which $$ is used.  It allows you to create temporary variable names
  that are specific to a particular execution of a macro.  For example,
  '@A$$ <- 3' might become '@A52 <- 3' or '@A57 <- 3' depending on when
  and where the macro is executed.  As long as you don't use any
  temporary names ending in two digits, rigorous use of '$$' eliminates
  the possibility of "collisions" between temporary variables names in
  different macros.

  In a macro expanded in-line, the value of $$ starts at 50.  If $$
  reaches 100, all macros abort.  In a macro expanded out-of-line, the
  value of $$ ranges from 0 to 49 and all macros abort if it reaches 50.
  This behavior limits the depth to which macros can be nested or
  recursive macros can call themselves

  Note that each invocation of a macro expanded in-line in a loop will
  have the same value for $$ since it is expanded only once, the first
  time through the loop.  This will usually be the case for macros
  expanded out-of-line, too, but it cannot be guaranteed.

                        Recursive use of macros
A macro may invoke itself directly or indirectly up to a maximum depth
of 50, provided some test is included to avoid infinite recursion.  In
practice, current limitations of the parser limit the maximum depth to
around 20.  If you need greater depth, you may be able to attain it
using evaluate().

                      Tips for good macro writing
Because macro arguments are expanded literally, their place holders
should usually be enclosed in parentheses when used in expressions.
For example, use '($1)*($2)' rather than '$1*$2'.

When a macro argument is referred to more than once in a macro it should
usually first be copied to a temporary variable.  For example, a macro
to compute a mean should be defined by
  Cmd> mean <- macro("@x <- $1;sum(@x)/dim(@x)[1]",dollars:T)
rather than by the somewhat simpler
  Cmd> mean <-  macro("sum($1)/dim($1)[1]")
If the latter definition is used, 'mean(boxcox(x[,vector(1,2,4)],.5))'
would result in boxcox(x[,vector(1,2,4)],.5) being evaluated twice.

When a macro is not intended to return a value, end it with ';;'.  For
arcane reasons, this helps MacAnova allocate memory more efficiently.
```

```
Use error() to print any messages describing errors that terminate the
macro.  It works similarly to print() but automatically prefaces the
message with "ERROR: ", appends " in macro xxxxx" and produces an error
condition that terminates the macro and returns to the prompt level.

If the error message contains the macro name (dollar symbol $S), use
keyword phrase 'macroname:F' with error() as in
  if ($v != 2){error("$S requires 2 non-keyword arguments",macroname:F)}

See topic error() for more information.

If you use anova(), regress() or any other GLM command in the macro and
you do not want to destroy information from a previous GLM command,
bracket the use of a GLM command by pushmodel() and popmodel() as in the
following fragment:
  if (pushmodel(canpush:T)){pushmodel()}
  anova(@model, silent:T)
  if (popmodel(canpop:T)){popmodel()}

If you want to have the macro run on versions dating prior to the
introduction of pushmodel() and popmodel(), use
  if (alltrue(isfunction(pushmodel),pushmodel(canpush:T))){pushmodel()}
  anova(@model, silent:T)
  if (alltrue(isfunction(popmodel),popmodel(canpop:T))){popmodel()}

                        Functions useful in macros
There are several functions whose primary usefulness is within a macro.

keyvalue() interprets arguments that are keyword phrases list and check
their values for appropriateness.

argvalue() lets you access a non-keyword argument while checking it for
appropriateness.

anymissing(), isdefined(), isscalar(), isvector(), ismatrix(),
isarray(), isfactor(), isreal(), ischar(), islogic(), isnull(),
isgraph(), isstruc(), ismacro(), isfunction(), isnumber() and isname()
are also useful in checking whether an argument is suitable.

modelvars(), modelinfo(), varnames(), xvariables() and xrows() allow a
macro to do sophisticated computations based on the results of a
previous GLM command.

pushmodel() and popmodel() allow you to save and restore results from a
previous GLM command.

setodometer() is helpful in looping over all combinations of factor
levels.

                                   Example
Here is a fragment that might be in a macro whose first argument should
be a REAL scalar and second argument should a CHARACTER vector, and
which should recognize keywords 'down' with T or F for value and 'power'
```

```
with positive REAL scalar value.  'down' is optional with default F and
'power' is required.

  @n <- argvalue($1,"sample size","positive integer scalar")
  @labs <- argvalue($2,"labels","character vector")
  @down <- keyvalue($K,"down","TF",default:F)
  @power <- keyvalue($K,"power","positive number")
  if (isnull(@power)){
     error("keyword 'power' is required by $S",macroname:F)
  }
  ....

When the value of an argument doesn't have the specified properties,
argvalue() and keyvalue() immediately terminate the macro.

                            Cross references
See macroread() for examples of macros.
```

2.221 macroread()

Usage:

```
   mymacro <- macroread(FileName,macroName [,quiet:T or F, echo:T or F,\
     silent:T, printname:F, notfoundok:T, nofileok:T, prompt:F,\
     badkeyok:T]), FileName and macroName CHARACTER scalars; FileName can
     also be CONSOLE or have the form string:charVal where charVal is a
     CHARACTER scalar or vector.
```

Keywords: macros, files, input

```
                                  Usage
mymacro <- macroread(FileName,macroName) searches the named file for a
macro whose name matches macroName.  If the macro is found, it is read
and made available under the name mymacro().  FileName and macroName
must be quoted strings or CHARACTER scalars.

Usually the name to the left of '<-' will be the same as the name of the
macro read as in
  Cmd> density <- macroread("densities.mac","density")

In searching the file for a matching name, case is ignored so that, so
that macroread(FileName, "mymacro") will find mymacro(), MyMacro(), or
MYMACRO(), for example.  See topic 'macro_files' for a description of
the required file format.

In a version with windows, when FileName is the null string "", you
will be prompted to select the file using a dialog box.

The header and comment lines are normally echoed to output, but this can
be suppressed by keywords;  see below.

If the macro has keyword LOCKED on the header line, and the result is
```

assigned to a variable, that variable will be locked. See topic 'locks'.

```
  Cmd> doit <- macroread("macrofile.txt","doit")
  doit  MACRO LOCKED

  Cmd> list(doit)
  doit            MACRO  (in-line) (locked)
```

Macro name omitted

mymacro <- macroread(FileName) with no macro name reads the first macro on the file. The first non-empty, non-blank line in the file is assumed to be the start of the macro as described in topic 'macro_files'. It is an error if it is not in the right format for the first header line of a macro or data set.

Assignment of result

Just reading a macro does not make it available; you must assign the value of macroread() using '<-'. You can usually use getmacros() to simultaneously read a macro and make it available.

Macro getmacros()

When you are reading a macro from one of the standard macro files (arima.mac, design.mac, graphics.mac, macanova.mac, math.mac, mulvar.mac, regress.mac and tser.mac) pre-defined macro getmacros() is more convenient to use than macroread(). An example of its use is

```
  Cmd> getmacros(covar)
```

This replaces to 'covar <- macroread("macanova.mac","covar")', except that you don't even need to know which file covar() is located in.

Automatic search for macros

Use of macroread() or getmacros() is sometimes not necessary, since the default behavior of MacAnova is to search the files in MACROFILES (see getmacros()) for any undefined macro you try to use. For example, even if macro covar() has not previously been read from file "MacAnova.mac", either by getmacros() or macroread(),

```
  Cmd> cov <- covar(x)
```

will read covar() and then execute it. However, both macroread() and getmacros() normally echo the header lines on macros; these often contain details about usage which you otherwise might miss.

See below for discussion of keywords 'quiet', 'echo', 'silent', 'notfoundok' and 'nofileok'.

OUTOFLINE on header

If 'OUTOFLINE' or simply 'OUT' appears on the first header line of the macro, it will be marked to be always expanded out-of-line. Otherwise, the expansion of the macro will be determined by the value of option 'inline'. See topics 'macros', 'options'.

Reading from console or batch file

macroread(CONSOLE [,prompt:F]) reads from the regular input stream allowing you to type in the macro using the format described under topic 'macro_files'.

On windowed versions, type one line at a time in the dialog box that is opened.

On Unix/Linux and DOS, type the necessary lines after the prompt. The value of CONSOLE is ignored. The first line must be of the form 'Name nLines MACRO', where nLines is the number of lines in the macro.

On any machine, when macroread(CONSOLE) is used in a batch file, it reads the macro from the lines immediately following the macroread() command. See batch(). No prompt is printed when prompt:F is an argument.

Any blank lines at the end of a macro are trimmed off when it is read.

Keywords

There are several keywords, 'quiet', 'echo', 'silent', 'notfoundok' and 'nofileok' which control what will be printed by macroread().

Keyword phrase	Meaning
quiet:T	Header and descriptive comments will not be printed
quiet:F	All header and descriptive comments will be printed
echo:T	Lines of the macro itself will be printed as they are read
silent:T	Only error messages will be printed; incompatible with quiet:F or echo:T
printname:F	The name of the file read will not be printed; printname:T is ignored with silent:T
notfoundok:T	Failure to find the macro is not considered an error so no error message is printed.
nofileok:T	Failure to open the file is not considered an error so no error message is printed.
badkeyok:T	Unrecognized or duplicate keywords are silently ignored.

Without quiet:T or quiet:F, the header and comment lines not starting with '))' preceding the macro will be echoed to output.

Even without echo:T, header lines are printed when FileName is CONSOLE and the macroread() command is in a batch file. (In windowed versions, a macro will be echoed if FileName is CONSOLE whether or not the command is in a batch file.) Such echoing can be suppressed by 'echo:F'.

When notfoundok:T is an argument and the macro is not found or nofileok:T is an argument and the file cannot be opend, macroread() returns NULL as value. When used in a macro, this feature allows special action to be taken if macroName is not found. See topic 'NULL'.

```
                      Keywords 'file' and 'string'
macroread(file:FileName,...) is equivalent to macroread(FileName,...).

macroread(string:CharVar,...) where CharVec is a CHARACTER scalar or
vector, does not read from a file.  Instead, it "reads" CharVar as if
each element were a line (or several lines if there are embedded
end-of-line characters) read from a file.  The first element or line of
CharVar must be a header line with a name and number of lines.  In
particular, mymacro <- macroread(string:CLIPBOARD) would read the first
macro on a replica of a data file in the special variable CLIPBOARD.  In
windowed versions this would be taken from the Clipboard.  In the GTK
version, you can also "read" from special variable SELECTION in a
similar way.  See topic 'CLIPBOARD'.

If either keyword 'file' or 'string' is used, they can appear in any
position in the argument list, as can setName which must be the only
non-keyword argument.  For example,
  Cmd> macroread(quiet:T,"mymacro",file:"myfile.dat")
is equivalent to
  Cmd> macroread("myfile.dat","mymacro", quiet:T).

                           Variable MACROFILES
A predefined CHARACTER variable MACROFILES contains the names of files
containing macros.  A pre-defined macro getmacros() allows easy
retrieval of macros from the files whose names are in MACROFILES.  At
startup, MACROFILES is initialized to vector("graphics.mac",
"regress.mac", "design.mac", "tser.mac", "arima.mac", "mulvar.mac",
"math.mac", "macanova.mac"), but you can change it if desired.  See
topics getmacros() and addmacrofile().

                            Cross references
See also topics 'macros', macro(), read(), matread(), inforead(),
'macro_files', 'files'
```

2.222 macros

Usage:

```
mymacro <- macro(charVar [, dollars:T])
mymacro <- macroread(fileName [,"mymacro"])
getmacros(macro1 [, macro2 ...]) (reads macro from one of files
  specified in MACROFILES)
macrowrite(fileName, mymacro)
```

Keywords: macros, control, syntax

```
                                Description
A macro is a collection of commands grouped together to make it easy to
execute them all at once.  It is used (invoked) the same way as a
function, by typing its name followed by 0 or more arguments in
parentheses.  For example, y <- boxcox(x,.5) invokes macro boxcox().
```

A macro is stored in the MacAnova workspace as a variable similar to a CHARACTER scalar. You can print its text by typing its name.

In-line expansion of a macro

By default, when a macro is invoked it is expanded in-line, that is, its arguments are literally substituted into the text of the macro and the modified text is inserted in the line being executed exactly as if it had been typed in place of the macro call.

Because an in-line macro is inserted directly in the line, a particular instance of a macro is expanded only once, even if it is repeatedly encountered during a loop (see 'for' and 'while'). This makes it impossible to redefine an in-line macro in a loop and execute the new version the next time through. However, a macro that is expanded out-of-line (see below) is expanded every time it is encountered, allowing meaningful redefinition within a loop.

Ways macros may be defined

Macros may be read from a file by macroread() or created directly using macro(). There also many pre-defined macros such as readcols() and boxcox(). Macros may be written to a file by macrowrite().

When you try to use a macro that has not been defined, the default behavior of MacAnova is to print a warning message and then search the standard macro files (specifically the files whose names are in CHARACTER variable MACROFILES; see getmacros()) for the macro. If the macro is found, it is read in (without echoing the header lines) and executed; if not, further execution of the command line is terminated. Option 'findmacros' allows you to suppress the automatic search and/or the printing of warning messages. See topic 'options'.

A macro can be a component of a structure (see 'structures') although it must be extracted in order to be used. For example, if structure boxcoxstr was created by boxcoxstr <- structure(boxcox), boxcoxstr$boxcox(x,.5) is illegal. You would have to use something like @tmpboxcox <- boxcoxstr$boxcox;@tmpboxcox(x,.5).

Out-of-line expansion of a macro

An alternative mode of macro expansion is out-of-line. In this mode, a modified copy of the macro text is created, substituting macro arguments in the text. This is then executed without being inserted in the line being executed. In a loop, a particular instance of an out-of-line macro will be expanded every time through the loop. The value of option 'inline' (default value is True) determines the default expansion mode. In addition a macro can be marked always to be expanded out-of-line by keyword phrase inline:F on macro(). See topics 'options' and macro().

Cross references

See also topics addmacrofile(), getmacros(), macro(), macrowrite(), macrousage(), 'macro_files', 'macro_syntax'.

See macroread() for examples of macros.

2.223 macrousage()

Usage:

```
macrousage(Macro1 [,Macro2, ...] [,silent:T]), Macro1, Macro2, ...,
  currently defined macros
```

Keywords: macros, general

```
                                Usage
macrousage(Macro) prints all comment lines (lines that start with "#")
in Macro().  It is an error if Macro is not the name of a macro.  These
usally describe the usage of the macro, but that may not always be the
case.

macrousage(Macro, silent:T) does the same, except any warning messages
are suppressed.  silent:T does *not* suppress printing the usage
information.

macrousage(macroNames [,silent:T]), where macroNames is a quoted string
or CHARACTER variable specifying one or more macro names, prints the
comment lines in each macro named.

macrousage(arg1, arg2, ... [, silent:T]), where each argument is either
a macro or a CHARACTER variable does the same for several macros.

macrousage() returns in invisible LOGICAL scalar whose value is True if
and only if at least on macro was found.

                              Examples
Example:
  Cmd> macrousage(colplot, rowplot)
  Cmd> macrousage(listbrief(macros:T, keep:T)) # usage for all macros
  Cmd> if (!macrousage(foo,silent:T)){print("foo not found")}

                           Cross references
See also topics help(), macro(), 'macros'.
```

2.224 macrowrite()

Usage:

```
macrowrite(fileName,a,b,... [,name:Name,header:F,comments:charVec,\
  oldstyle:T,stripdols:T]), a, b, ... macros, fileName and Name
  CHARACTER scalars, charVec a CHARACTER vector or scalar
```

Keywords: macros, files, output

```
                                Usage
macrowrite(FileName,a, b, ... ) writes macros a, b, ... on the file.
Filename must be a CHARACTER variable or quoted string and a, b,
... must be macros.  a, b, ... are written in the form recognized by
macroread.  The default is to write each with no line count in the
```

```
header and with a trailing line of the form %macroname%.

If FileName is variable CONSOLE or a CHARACTER variable whose value is
"CONSOLE", the output is written to the screen rather than to a file.
The value of variable CONSOLE is ignored.

If the macro was previously marked to be expanded out-of-line,
"OUTOFLINE" is added to the header line,

                              Keyword 'new'
Keyword 'new':
macrowrite(FileName, a, b, ..., new:T) removes all information currently
in the file before writing new information.  Without 'new:T', macros are
written at the end of the file.

                             Keyword 'comment'
Keyword 'comment':
macrowrite(FileName, a, comment:charVec) writes each element of
CHARACTER vector or quoted string charVec, prefixed by ") ", as a
comment line after the header.  If header:F appears, no such comments
are written.

                             Keyword 'header'
Keyword 'header':
macrowrite(FileName,a,b,...,header:F) writes the macros without any
header lines or any trailing %macroname%.  They will not be readable by
macroread().  Also, keyword 'comments' will be ignored.

                            Keyword 'oldstyle'
Keyword 'oldstyle'
macrowrite(FileName, a, b, ..., oldstyle:T) writes each macro in the old
style format.  A line count will be included in the header line and no
trailing %macroname% line will be written.

                            Keyword 'stripdols'
Keyword 'stripdols'
macrowrite(FileName, a, b, ..., stripdols:T) strips '$$' off the end of
variable names of the form @name$$ as each macro is written, and adds
'DOLLARS' to the header line.  When the macro is subsequently read from
the file by macroread(), '$$' will be automatically added to all
temporary variable names of the form @name.  As long a macro originally
had no temporary variables that did not end in '$$', reading the macro
restores it exactly.  See topics 'macro_files' and 'variables'.  This
feature was added because it is easier to read and edit macros if the
temporary names don't all end in '$$'.

                             Cross references
See also topics getmacros(), 'macros', macro(), macroread(), matwrite(),
read(), matread(), 'macro_files', 'files'.
```

2.225 makecols()

Usage:

```
makecols(x,var1,var2, ... [,keyword phrases]), where x is a REAL matrix,
  var1, var2, ...  unquoted or quoted variable names
makecols(x,vector("var1","var2", ... ) [,keyword phrases])
makecols(x [,keyword phrases]), x a REAL matrix with labels.
makecols(charx,var1,var2, ... [,keyword phrases]), charx a CHARACTER
  scalar or vector
makecols(charx,vector("var1","var2", ... ) [,keyword phrases])
makecols(charx [,keyword phrases]), first line of charx containing
  variable names
Keyword phrases are factors:facVec, nomissing:T, quiet:T or silent:T,
  facVec a LOGICAL vector or vector of positive integers
```

Keywords: combining variables

```
                                Usage
makecols(x,name_1,...,name_k), where x is a REAL matrix and name_1, ...,
name_k are unquoted or quoted variable names, creates new REAL vectors
name_1, name_2, ... from the columns of x.

You can think of makecols() as a sort of inverse to hconcat().  If a
name which is not a quoted string is the name of an existing variable,
only the name is used, not the value of variable.  A report is printed
of the variables created.

It's OK for the number of names to differ from the number of columns in
x.  When there are more names than columns, the extras are ignored.
When there are more columns than names, the final columns are ignored.

makecols(x,vector("name_1","name_2",...,"name_k") [keyword phrases]) is
an alternative usage.

                          Keyword 'factors'
makecols(x,name_1,...,name_k, factors:facVec) does the same, but
specifies that some of the columns of x are to be saved as factors.
facVec can be a vector of positive integer column numbers or a LOGICAL
vector of length k.  When facVec contains integers, they are the columns
of x to be saved as factors.  When facVec is LOGICAL, column i of x is
saved as a factor only if facVec[i] is True.  For example, when x has 4
columns, both makecols(x,a,b,c,y,factor:run(3)) and makecols(x,a,b,c,y,
factor:vector(T,T,T,F) saves columns 1, 2 and 3 of x as factors a, b and
c and column 4 of x as REAL vector y.

You can use keyword 'factors' with any other keyword.

                         CHARACTER argument 1
makecols(charx,name_1,...,name_k), where charx is a CHARACTER scalar or
vector does the same, except the data to be saved is obtained as
vecread(string:charx, fields:T) (see subtopic vecread:"reading_from_
character_variable").

makecols(charx) does the same, provided the first line of CHARACTER
```

scalar or vector charx consists of a list of legal MacAnova names.

Use with data on clipboard

An important special case of a CHARACTER first argument is makecols(CLIPBOARD, name_1, ... [,factors:TorFvec). You could use this when you have copied a data matrix to the clipboard, perhaps in a spread sheet program. makecols(CLIPBOARD [,factors:TorFvec[) would be appropriate when the first line of the selection copied contained column headings to be used as variable names. See topic 'CLIPBOARD'.

Keywords 'quiet' and' 'silent'

makecols(x,name_1,...,name_k,quiet:T) does the same, except no report is printed.

makecols(x,name_1,...,name_k,silent:T) does the same, except nothing is printed, not even warning messages.

'quiet:T' and 'silent:T' can always be used, no matter how names are specified.

Keyword 'nomissing'

makecols(x,name_1,...,name_k,nomissing:T) does the same except any MISSING values are removed from the variables created. If all the elements in a column of x are MISSING, the corresponding variable is NULL. A warning message is printed unless 'silent:T' is an argument.

'nomissing:T' can always be used, no matter how names are specified.

Names from column labels

makecols(x), with no names provided, is legal when x has labels. It is equivalent to makecols(x,getlabels(x2)), and creates variables using the column labels as names. See topics 'labels', getlabels().

The first argument can also be a CHARACTER scalar or vector such as CLIPBOARD; see below.

Examples

Example:

```
  Cmd> makecols(x, x1, x2, x3, x4)
  Cmd> makecols(x,"x1","x2","x3","x4")
  Cmd> makecols(x, vector("x1","x2","x3","x4"), quiet:T)
```

These all create vectors x1, x2, x3 and x4 from the first 4 columns of x. All but the last print a report that these vectors were created

```
  Cmd> makecols(x, x1, x2, x3, x4, nomissing:T,silent:T)
```

This does the same except x1, ..., x4 will have no MISSING values, with no report and any warning messages suppressed.

```
  Cmd> makecols(data,a,b,c,y,factors:run(3))

  Cmd> makecols(data,a,b,c,y,factors:vector(T,T,T,F)
```

```
These both save columns 1, 2 and 3 of data as factors a, b and c and
column 4 as vector y.

  Cmd> makecols(vector("1 2","3 4"),x,y)# makecols(" 1 2\n 3 4", x, y)
  Cmd> makecols(vector("x y","1 2","3 4"))# makecols("x y\n 1 2\n 3 4")

These both create vectors x and y of length 2 by "reading" the CHARACTER
first argument.

makecols() is implemented as a pre-defined macro.

                            Cross references
See also topics readcols(), clipreaddata(), hconcat().
```

2.226 makefactor()

Usage:

```
   makefactor(vec [,sort:F] [,labels:T or F]), vec a REAL, CHARACTER or
     LOGICAL vector
```

Keywords: glm, anova, character variables

```
                                  Usage
makefactor(vec) creates a factor constructed from the REAL, CHARACTER or
LOGICAL vector vec.  If there are m unique values in vec, the output
will be a factor with m levels in the same order as the values in vec.
MISSING values remain MISSING.

makefactor(vec, sort:F) does the same, except the factor levels may not
have the same order as the elements of vec.  Level 1 will be assigned to
value[1], level 2 to the next value in vec different from vec[1], and so
on.

If vec is a CHARACTER vector without row labels, its values are attached
to the result as row labels unless 'labels:F' is an additional argument.

                            With labeled argument
If vec has row labels, they become the row labels of the result.

makefactor(vec, labels:F, [, sort:F]) does the same, except the result
has no row labels.

makefactor(vec, labels:T, [, sort:F]) does the same, except that if vec
does not have labels, a CHARACTER representation of the values in vec is
attached to the result as row labels.

                          makefactor() vs factor()
Even when vec is REAL and consists of positive integers, it may be
preferable to use makefactor() rather than factor(), since the output
from factor() will not contain all m levels if max(vec) > m, or if some
```

```
levels are missing in vec.

                              Examples
Examples:
  Cmd> a <- makefactor(vector(5.2,2.6,3.9,1.3,3.9,1.3,5.2,2.6)); a
  (1)           4           2           3           1           3
  (6)           1           4           2

  Cmd> b <- makefactor(vector("D","B","C","A","C","A","D","B"));b
           D           B           C           A           C
           A           D           B
           4           2           3           1           3
           1           4           2

  Cmd> c <- makefactor(vector(5.2,2.6,3.9,1.3,3.9,1.3,5.2,2.6),sort:F);c
  (1)            1            2            3            4            3
  (6)            4            1            2

  Cmd> d <- makefactor(rpoi(5,11)/10,labels:T); d
           1.2         0.9           1         1.4         0.9
             3           1           2           4           1

makefactor() is implemented as a macro.

                           Cross references
See also factor().
```

2.227 makestr()

Usage:

```
   makestr(var1 [,var2,...,vark] [, KeyPhrases]), where var1, var2, ... are
     arbitrary variables
   KeyPhrases can be compnames:Charvec, labels:lab, and silent:T, where
     Charvec and lab are CHARACTER scalars or vectors.
   Use structure() instead.
```

Keywords: structures, combining variables

```
                                 Usage
makestr() is identical to structure().  See structure() information
on its use.

The use of makestr() is deprecated -- that is, it will continue to be
available for the immediate future, but at some point may be disabled.
Use structure() instead.

                           Cross references
See also topics strconcat(), 'structures', 'keywords', changestr(),
compnames().
```

2.228 makesymbols()

Usage:

```
plotsymbols(intVar), intVar a REAL variable with integer elements
  between 1 and 255
plotsymbols(charVar [,medium:T or small:T]), charVar a CHARACTER
  variable of shape names "diamond", "plus", "square", "cross",
  "triangle", "star", "dot" or "circle"
```

Keywords: character variables, plotting

```
                                Usage
makesymbols(intVar), where intVar is a REAL scalar, vector, matrix or
array whose elements are integers between 1 and 255 returns a CHARACTER
variable with the same size and shape as intVar.  Each element of intVar
is interpreted as an ASCII code and the corresponding element of the
result is the single character with that code.

  Cmd> symbols <- makesymbols(vector(2,29,65,97)); symbols
  (1) "\002"            [octal representation of 2]
  (2) "\035"            [octal representation of 29]
  (3) "A"               [ASCII code 65]
  (4) "a"               [ASCII code 97]

makesymbols(charVar), where charVar is a CHARACTER scalar, vector,
matrix or array, also returns a CHARACTER variable the same size and
shape as charVar.  Any element of charVar whose first three characters
match the first three letters of "diamond", "plus", "square", "cross",
"triangle", "star", "dot" or "circle", is replaced in the result by
"\001", "\002", "\003", "\004", "\005", "\006", "\007", or "\010",
respectively.  These are the plotting symbol codes for these shapes.
Any elements of charVar not specifying one of these shapes are put in
the result without change.

  Cmd> makesymbols(vector("diamond","dot","circle"))
  (1) "\001"
  (2) "\007"
  (3) "\010"

symbols <- makesymbols(charVar, medium:T) does the same except the shape
names are translated to "\011", "\012", "\013", "\014", "\015",
"\016", "\017", or "\020".

symbols <- makesymbols(charVar, small:T) does the same except the shape
names are translated to "\021", "\022", "\023", "\024", "\025",
"\026", "\027", or "\023".

See subtopic "chplot:drawn_plotting_symbols" for more information about
plotting symbols.  In particular, in specifying these symbols
explicitly, leading 0's can be omitted, so chplot(x,y,symbols:"\7") is
the same as chplot(x,y,symbols:"\007").

                            Use in plotting
makesymbols() is designed to be used to create a CHARACTER vector or
```

```
matrix to be used as the value of keyword 'symbols' on plotting commands
such as plot(), chplot() and addchars().  For this usage the dimensions
of intVec and charVec must match what is expected.  See subtopic
"chplot:symbol_variable_shape" for details.

Examples:
In the following, x is a vector and y a matrix with 3 columns:

  Cmd> plot(x,y[,1],symbols:makesymbols("diamond"))#or makesymbols(1)

plots column 1 of y using diamonds (code "\001") as plotting symbol.

  Cmd> lineplot(x,y, symbols:makesymbols(vector("dia","plus","squ"),\
       medium:T))

  Cmd> lineplot(x,y, symbols:makesymbols(9, 10, 11))

  Cmd> lineplot(x,y, symbols:vector("\11", "\12", "\13"))

all make line plots of the columns of y, with medium sized diamonds,
plus signs and squares used as plotting symbols for each columns

                        Difference from putascii()
When intVar is a vector, the usage makesymbols(intVar) is somewhat
similar to putascii(intVar, keep:T) since both translate ASCII codes to
characters.  The difference is that putascii() returns a CHARACTER
scalar with length(intVar) characters while makesymbols() returns a
CHARACTER vector of length(intVar), with each element a single
character.

                             Cross references
See also topics chplot(), 'plotting', putascii().
```

2.229 manova()

Usage:

```
manova([Model] [,print:F or silent:T, coefs:F, pvals:T, fstats:T,\
  byvar:T, sssp:F or T]), Model a CHARACTER scalar
```

Keywords: glm, multivariate analysis, anova

```
                                    Usage
manova(Model) computes a MANOVA table of SS/SP (sums of squares and sums
of products) matrices for the model in the CHARACTER variable Model.
The response variable should be a matrix with rows as cases and columns
the variables.

Type 'help(models)' for information on how to specify Model.

If the response is univariate (has only one column), manova() is
equivalent to anova().
```

Unless 'marginal:T' is an argument, SS/SP matrices are computed sequentially (so called SAS Type I quantities).

Normally, when each row of a SS/SP matrix will fit on a single line, all matrices are printed in their entirety. When a row would require more than one line, only the term names and the degrees of freedom are printed. This behavior can be modified by keywords 'sssp', 'byvar', 'fstats' and 'pvals'; see below. In any case the matrices are saved in the three-dimensional side effect array SS, with the first subscript indexing terms and with the first dimension labeled by TERMNAMES.

Keyword 'weights'

manova(Model,weights:Wts) does a weighted analysis. Wts must be a REAL vector with Wts[i] >= 0 and nrows(Wts) = nrows(response). The results are the same as if the i-th row of the response and all X-variables (variates and dummy variables and their products), including the constant vector were multiplied by sqrt(Wts[i]) and a least squares fit (without an intercept) computed. You can abbreviate 'weights:Wts' to 'wts:Wts'.

No model supplied

manova() or manova(,weights:Wts) (no model supplied) uses the model used by the most recent GLM command such as manova(), anova(), or poisson(). See topic 'glm'.

Side effect variables created

Side effect variables created are RESIDUALS, HII, DF, SS, DEPVNAME, TERMNAMES, and STRMODEL. When weights are specified, RESIDUALS = Response - Fitted and WTDRESIDUALS = sqrt(Wts)*RESIDUALS is an additional side effect vector. You should use WTDRESIDUALS rather than RESIDUALS in residual plots or other diagnostic procedures.

Multivariate tests

SS is a 3-dimensional array such that SS[j,,] is the sum of squares and products matrix for term j. If the appropriate error matrix for the k-th term is SS[j,,], the eigenvalues needed for several standard tests (Wilks, Roy, Pillai, Hotelling generalized T-squared) may be computed by releigenvals(SS[j,,],SS[k,,]) or you can compute some test statistics directly, for example,

```
  Cmd> T2 <- dferror*trace(solve(SS[k,,],SS[j,,])
```

or

```
  Cmd> lambda <- det(SS[j,,])/det(SS[k,,]+SS[j,,]).
```

Other Keywords

Keyword phrase	Default	Meaning
byvar:T	F	Computes a complete ANOVA table for each variable. The full SS/SP matrices are not printed although they are still available in array SS.
sssp:T	none	Forces the printing of the full SS/SP matrices, even when each row would require more than one

```
                        line.  This option is ignored with any of
                        fstats:T, pvals:T, or byvar:T.
  sssp:F                Suppresses printing of the full SS/SP matrices,
                        even if a row would fit on a single line.  Only
                        the term names and degrees of freedom are
                        printed.

See topic 'glm_keys' for information on keyword phrases 'print:F',
'silent:T', 'fstats:T', 'pvals:T', 'coefs:F' and 'marginal:T'.

                             Keyword 'byvar'
When byvar:T is an argument, options (not keywords) 'fstats' and 'pvals'
have the same effect as with anova().  If byvar:T is not an argument,
these options are ignored.  See topics 'options' and 'glm_keys'.

When byvar:T is not an argument, options 'fstats' and 'pvals' are
ignored.  If either 'fstats:T' or 'pvals:T' is an argument, univariate
SS and MS are printed for each variable and term, together with F-
statistics and/or P values .  The information is essentially the same as
with byvar:T except that all the statistics for a term are grouped
together.  The full SS/SP matrices are not printed but are available in
array SS.

                     Interaction with other functions
contrast(), coefs(), predtable(), and cellstats() work after manova().
```

2.230 match()

Usage:

```
match(x,Target [,nomatch, exact:F or fuzz:d, relative:T, ignorecase:T]),
  x REAL, LOGICAL or CHARACTER, Target a variable of the same type as x,
  nomatch and d >= 0 REAL scalars; x and Target can't both be nonvectors
```

Keywords: ordering, variables, character variables

```
                                   Usage
match(x,Target,noMatch), where x is REAL, LOGICAL or CHARACTER and
noMatch is a REAL scalar, attempts to match each element of array x with
each element of vector Target.  Target must be the same type as x.  The
result is REAL with the same dimensions as x.  When x is REAL or
LOGICAL, it is an error for Target to contain any MISSING values.

Let J represent the subscripts of an element of x.  Then result[J] has
value noMatch, when no element of Target matches x[J], and has value k
when Target[k] is the first value with Target[k] = x[J].  When x is REAL
and x[J] is MISSING, then result[J] is MISSING.

match(x,Target) does the same except that length(Target) + 1 is used as
a value for noMatch and, when there are any non-matching elements, an
advisory message is printed.
```

```
When x is a vector, Target can be a matrix or array and the matching is
done for each combination of the second and higher dimension of Target.
The result has dimensions vector(length(x), dims(Target)[-1]).  In this
case, when noMatch is omitted, the value for non-matching elements is
dim(Target)[1] + 1.

@inexact_matching
match(x,Target [,noMatch], fuzz:d), where x is REAL and d >= 0 is a
non-MISSING REAL scalar, does the same, except that x[J] matches
Target[k] when abs(x[J] - Target[k]) <= d.

match(x,Target [,noMatch], fuzz:d, relative:T) does the same, except
that x[J] matches Target[k] when (abs(x[J] - Target[k]) <= D, where
D = d*(abs(x[J]) + abs(Target[k])).

match(x,Target [,noMatch], ignorecase:T) ignores the case, upper or
lower, of any alphabetic characters in elements of CHARACTER variables x
and A.  For example, match("AbC",vector("xYz","abc","def"),
ignorecase:T) has value 2.

match(Pattern,Target [,noMatch], exact:F [,ignorecase:T]), where Pattern
is a CHARACTER scalar containing one or more of the "wild card"
characters '*' and '?', and Target is a CHARACTER vector, does the same,
except that an exact match is not required to 'hit' Target.

                    Wild card characters '*' and '?'
A '*' in Pattern will match 0 or more successive characters of an
element of Target, without regard to what they are.  A '?' in Pattern
will match any single character of an element of Target, without regard
to what it is.  For example, "start*" matches the first element of
Target that begins with "start", "*mid*" matches the first element
containing "mid", "*mid1*mid2*end" matches the first element finishing
with "end" that earlier contains "mid1" and "mid2" in that order, "p?l*"
matches the first element starting with 'p' and whose third letter is
'l', and so on.  As particular cases, "*" always matches Target[1] and
"\"*\"" matches the first element starting and ending with '"'.

                                 Examples
Examples:
  match(vector(1.3,2.4,1.3,5,5.1),vector(2.4,1.3),-1) returns
    vector(2,1,2,-1,-1)
  match(vector(1.3,2.4,1.3,5,2.4,?),vector(2.4,1.3)) returns
    vector(2,1,2,3,1,?)
  match(vector(1.3,2.4,1.3,5,2.4),run(3),-99,fuzz:.5) returns
    vector(1,2,1,-99,2)
  match(vector("A","B","A","C","B"),vector("B","A")) yields
    vector(2,1,2,3,1)
  a <- factor(match(x,sort(unique(x)))) transforms a REAL x to a factor
  unique(x)[match(x,unique(x))] yields x when x is a vector.
  match(scalarValue,vec,0) != 0 if and only scalarValue is in vec
  match("*c",vector("abc","ade","gfh"),exact:F) returns 1
  match("*d*",vector("abc","ade","gfh"),exact:F) returns 2
  match("g*",vector("abc","ade","gfh"),exact:F) returns 3
```

```
  match("g*h",vector("abc","ade","gfh"),exact:F) returns 3
  match("a*b*c",vector("abc","ade","gfh"),exact:F) returns 1
  match("a*b???",vector("aqbde","bb123", "allbdef"),exact:F) returns 3

In a macro, it can be helpful to know whether an argument is a an
explicit quoted string:
  if (match("\"*\"","$1",0,exact:F) != 0){
    print("arg 1 is a quoted string")
  }

                            Cross references
See also unique(), 'macro_syntax'.
```

2.231 mathhelp()

Usage:

```
   mathhelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
   mathhelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
     Subtopics
   mathhelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
   mathhelp(key:Key), CHARACTER scalar Key
   mathhelp(index:T [,scrollback:T])
```

Keywords: general, matrix algebra

```
                                    Usage
mathhelp(Topic1 [, Topic2, ...]) prints help on topics Topic1, Topic2,
... related to macros in file math.mac.  The help is taken from file
math.mac.

mathhelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these macros.

mathhelp(index:T) or simply mathhelp() prints an index of the topics
available using mathhelp().  Alternatively, help(index:"math") does the
same thing.

mathhelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER scalar
or vector, prints subtopics of topic Topic.  With subtopic:"?", a list
of subtopics is printed.

mathhelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where Suptopics1
and Subtopics2 are CHARACTER scalars or vectors, prints the specified
subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to mathhelp().  In windowed
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

                                Keyword 'key'
```

```
mathhelp(key:key) where key is a quoted string or CHARACTER scalar lists
all topics cross referenced under Key.  mathhelp(key:"?") prints a list
of available cross reference keys for topics in the file.

mathhelp() is implemented as a predefined macro.

                            Cross references
See help() for information on direct use of help() to retrieve
information from math.mac.
```

2.232 matprint()

Usage:

```
   matprint(fileName, a, b, ... [, new:T, format:Fmt, nsig:n, sep:sepChar,\
     quoted:T or bylines:T,missing:mVal, name:Name, comments:charVec,\
     width:w, header:F, oldstyle:T, stripdols:T]), a, b, ... arbitrary
     variables, Fmt, sepChar and Name CHARACTER scalars with sepChar only a
     single character, charVec a CHARACTER vector or scalar, mVal a REAL
     scalar, w >= 30 integer.
```

Keywords: output, files, missing values

```
                                   Usage
matprint(FileName,a,b,... [,new:T]) writes REAL, LOGICAL and CHARACTER
variables a, b,...  (scalars, vectors, matrices, or arrays) file
FileName in a form which can be read by read() and matread().  It can
also write NULL variables and structures.  GRAPH variables are legal
arguments but are currently written as NULL variables.  See topic
'NULL'.

matprint(a,b,...,file:FileName [,new:T]) is an alternative usage.

If new:T is present, anything already in the file is discarded before
writing.  Otherwise, writing is to the end of the file.

                               Keyword 'width'
If width:w, with w an integer >= 30, is not an argument, the default
value is taken from option 'width' (see subtopi 'options:"width"').
This is the presumed line length and is used to determine the maximum
number of values printed on one line.

                               Default formats
For REAL and LOGICAL variables, by default matprint() uses the format
that is used by print(), namely the format specified in option 'format';
this normally provides 5 significant digits in floating point form.

For CHARACTER variables, the default format, whenever possible, is "by
fields", that is elements are written as fields separated by spaces.
This is not feasible if there are any spaces or non-printable characters
in the data.  In that case, each element is quoted ("...").
```

```
Structures are written in a form that not only allows read() and
matread() to read all the components, but also can read individual
components if desired.  Since macros may be elements of structures,
keyword phrases 'oldstyle:T' and 'stripdols:T' may be arguments of
matprint().  See macrowrite().

See topic 'matread_file' for description of the file format.

                              Naming output
matprint(FileName,Name1:a,Name2:b,...) gives names Name1, Name2,... to
the data sets written in the file.  Name1, Name2,... must not be
keywords recognized by matprint, see below.  For example,
   matprint("Results.mat",values:releigenvals(h,e))
will write a matrix on file Results.mat with name 'values' on the first
line of the header .

                            Repeated keywords
Keywords 'nsig', 'format', 'name', 'header', 'missing' 'width',
'oldstyle', 'stripdols' and 'comments' are all recognized and can appear
more than once.  They affect the printing of objects that follow them,
until they are changed, except that the values of 'name' and 'comments'
are used only once.  Any of them that follow all items to be printed are
treated as coming before all items.  For example,
  Cmd> matprint("data.txt",x,nsig:5,y,nsig:10)
and
  Cmd> matprint("data.txt",nsig:10, x,nsig:5,y)
are equivalent.  This does not apply to keywords 'file' and 'new' which
can appear only once.

                             Keyword 'name'
Keyword 'name':
matprint(FileName,name:charVar, a, b, ... ) prints a with the name
specified by quoted string or CHARACTER scalar charVar on the header.
This is an alternative to using a keyword to specify a name and can be
used when the name is not a legal MacAnova keyword name.  For example,
matprint("myfile", name:"Residuals",r) and matprint("myfile",
Residuals:r) are equivalent.  Keyword 'name' can be used several times
in the argument list, and affects only the next item to be written to
the file.  If name:charVar is the last argument, it is treated as if it
came before all items to be written to the file.

                            Keyword 'comment'
Keyword 'comment':
matprint(FileName, a, comment:charVec) writes each element of CHARACTER
vector or quoted string charVec, prefixed by ") ", as a comment line
after the header.  If header:F appears, no such comments are written.

                            Keyword 'missing'
Keyword 'missing':
matprint(FileName,a,b,...,missing:realVal) recodes MISSING values with
REAL number realVal.  For example, matprint("mydata.txt",x, missing:-99)
substitutes -99 for every MISSING value.  If 'missing' is not used,
MISSING values will be coded as -99999.9999.  In either case, for any
```

variable with MISSING values, a comment line of the form ')MISSING value' is written before the data, where value is either -99999.9999 or the value specified by 'missing'. This enables read() and matread() to recognize missing values and read them appropriately. Keyword 'missing' can be used several times, each affecting any variables later in the argument list. If it follows all variables to be printed, as in the example, it is as if it preceded them all. The value for 'missing' cannot itself be a MISSING value.

Note this use of 'missing' differs from print(), write() and setoptions() -- its value must be a REAL scalar, not a character string.

Keyword 'header'

Keyword 'header':
matprint(FileName,a,b,...,header:F) writes the variables without any header lines. They will not be readable by read() or matread() but will be readable by other programs that can read numbers separated by spaces. The only time you need header:T is when sep:"c" is an argument and you want to force the writing of a header.

Keyword 'width'

Keyword 'width':
matprint(FileName,a,b,...,width:w) temporarily sets option 'width' to w, an integer >= 30. This affects how many items are printed per line.

Keyword 'sep'

Keyword 'sep':
matprint(FileName,a,b,...,sep:"c"), where c is an arbitrary character, writes items of data separated by c instead of by spaces. This also suppress the printing of header lines unless 'header:T' is an argument. This option is useful if you want to export data to a spreadsheet or other program that can read comma- or tab-separated items. For example, to write x with values separated by commas, use matprint("export.dat",x, sep:","). matprint("export.dat",x,sep:"\t") writes items separated by tabs.

Keywords 'quoted' and 'bylines'

Keywords 'quoted' and 'bylines':
When writing a CHARACTER variable you can also include keyword phrases quoted:T or bylines:T. matprint(fileName, charVar, quoted:T) outputs the data set in "quoted fields" format, that is with each element enclosed in double quotes ("..."). matprint(fileName, charVar, byline:T) outputs the data set in "by lines" format, with each element starting on a new line. However, if there are non-printable characters in the data, "quoted" fields format will be used. You can output a character variable in comma separated quoted fields as is required form some programs such as data bases, by matprint(fileName, charvar, quoted:T,sep:",").

File name ""

On a version with windows, if FileName is "", you will be able to specify the file name and folder using a dialog box.

```
                      Keywords 'nsig' and 'format'
Keywords 'nsig' and 'format':
matprint() uses the same default format for each item written as does
print() and has the same keywords 'nsig' and 'format'.  See print() for
information on 'nsig' and 'format'.

                             LOGICAL argument
If an argument is LOGICAL, a comment line of the form ') LOGICAL' is
added to the header lines.  This is recognized by read() and matread().

                            Writing to CONSOLE
If FileName is variable CONSOLE or a CHARACTER variable whose value is
"CONSOLE", the output is written to the screen rather than to a file.
The value of variable CONSOLE is ignored.

                          Changing default format
You can change the default format for print() and matprint() by
setoptions() using keywords 'nsig' or 'format'.  See topics 'setoptions'
and 'options'.

                             Cross references
See also topics write(), write(), matprint(), macrowrite(), read(),
matread(), 'files'.
```

2.233 matread()

Usage:

```
y <- matread(FileName,setName [,quiet:T or F, echo:T or F,printname:F,\
  labels:Labels, silent:T, notfoundok:T, nofileok:T, badkeyok:T,\
  prompt:F]),
  FileName and setName CHARACTER scalars; FileName can also be CONSOLE
  or have the form string:charVal where charVal is a CHARACTER scalar or
  vector.
```

Keywords: input, files, missing values

```
                                    Usage
In the following you can substitute read() for matread().  The only
difference between them is that read() prints no warning message when it
finds a macro rather than a data set.

x <- matread(FileName,setName) searches a file for a data set whose name
matches setName.  If the data set is found, it is read and the data are
saved in variable x.  The data set must be a REAL, LOGICAL or CHARACTER
vector, matrix, or array or a structure with REAL, LOGICAL, CHARACTER or
macro components.  FileName and setName must be CHARACTER variables or
quoted strings.  See topic 'matread_file' for the required form for the
data set.

In searching the file for a matching name, case is ignored so that, so
that matread(FileName,"mydata") will find mydata, MyData, or MYDATA, for
```

```
example.

If setName is omitted (x <- matread("mydata.txt")), matread() will read
the first dataset on the file, expecting that the first non-blank and
non-empty line is a header line of the correct form (see below).

If the data set has keyword LOCKED on the header line, and the result is
assigned to a variable, that variable will be locked.  See topic
'locks'.

  Cmd> x <- matread("datafile.txt","x")
  x   3   COLUMNS LOCKED

  Cmd> list(x)
  x               REAL   3     (locked)

If setName is the name of a macro rather than a data set, it will be
read, but a warning message is printed (no warning printed by read()).

                            Empty file name
In a version with windows, if FileName is the null string "", you will
be able to select the file using a dialog box.

                           Assigning result
Just reading a data set does not make it available; you must assign the
value of matread() using '<-'.

                            Macro getdata()
Pre-defined macro getdata() is somewhat easier to use, provided you have
set variable DATAFILE to the name of the file.  Since the default value
of DATAFILE is "macanova.dat", another way to read 'irisdata' is

  Cmd> x <- getdata(irisdata)

If you have a file of data sets you will be analyzing, say file
"mydata.txt", redefine DATAFILE by
  Cmd> DATAFILE <- "mydata.txt"
Then you can use getdata() to read data sets from your file.  See
topic getdata().

                         Too large data items
If any data items in numerical data set are too large to be represented
in the computer (for example "1e3000), they are set to MISSING.

                           Unreadable items
If any data item in the file is not a proper number (for example
3."4a5"), it, together with numbers following it on the same line, are
set to MISSING.

matread() and getdata() work only with files in a special format with
header information.  Use vecread() and readcols() to read data files
that just consist of numbers.
```

Reading from CONSOLE or batch file

matread(CONSOLE [,prompt:F]) reads from the regular input stream allowing you to type in the matrix using the format described under topic 'matread_file'. On windowed versions, type one line at a time in the dialog box that is opened. On nonwindowed versions, type the necessary lines after the prompt. The value of CONSOLE is ignored. The first line entered must be a header which includes a name and dimensions. On any machine, when matread(CONSOLE) is used in a batch file, it reads the data from the lines immediately following the matread() command. This allows even large data sets to be included directly in a batch file. See batch(). No prompt is given if you include prompt:F as an argument.

Keywords

There are several keywords, 'quiet', 'echo', 'silent', 'notfoundok' and 'nofileok' which control what will be printed by matread().

Keyword phrase	Meaning
quiet:T	Header and descriptive comments will not be printed
quiet:F	All header and descriptive comments will be printed
echo:T	Data lines will be printed as they are read
silent:T	Only error messages will be printed; incompatible with quiet:F or echo:T
printname:F	The name of the file read will not be printed; printname:T is ignored with silent:T
notfoundok:T	Failure to find the data set is not considered an error so no error message is printed.
nofileok:T	Failure to open the file is not considered an error so no error message is printed.
badkeyok:T	Unrecognized or duplicate keywords are silently ignored.

Without quiet:T or quiet:F, the header and comment lines not starting with '))' preceding the macro will be echoed to output.

Even without echo:T, data lines are printed when FileName is CONSOLE and the matread() command is in a batch file. (In windowed versions, data will be echoed if FileName is CONSOLE whether or not the command is in a batch file.) Such echoing can be suppressed by 'echo:F'.

Keyword 'notfoundok'

When notfoundok:T is an argument and the data set is not found or when nofileok:T is an argument and the file cannot be opend, matread() returns NULL as value (see topic 'NULL'). When used in macro, this feature allows special action if data setName is not found.

Keywords 'file' and 'string'

matread(file:FileName,...) is equivalent to matread(FileName,...).

matread(string:CharVar,...) where CharVec is a CHARACTER scalar or vector, does not read from a file. Instead, it "reads" CharVar as if each element were a line (or several lines if there are embedded end-of-line characters) read from a file. The first element or line of CharVar must be a header line with a name and dimensioning information.

```
In particular,

  Cmd> x <- matread(string:CLIPBOARD)

would read the first data set on a replica of a data file in the special
variable CLIPBOARD.  In a version with windows, this would be taken from
the Clipboard.  In the GTK version, you can also "read" from special
variable SELECTION in a similar way.  See topic 'CLIPBOARD'.

If either keyword 'file' or 'string' is used, they can appear in any
position in the argument list, as can setName which must be the only
non-keyword argument.  For example,

  Cmd> x <- matread(quiet:T,"mydataset",file:"myfile.dat")

is equivalent to

  Cmd> x <- matread("myfile.dat","mydataset",quiet:T).

                          Cross references
See also topics read(), vecread(), readcols(), macroread(), inforead(),
'files'.
```

2.234 matread_file

Keywords: variables, files, input, output

```
                            Introduction
This topic discusses the format of files to be read by matread() and
read().  They are plain text files which contain named data sets, each
starting with one or more header lines, such as are written by
matprint() and matwrite().

matread() and read() behave identically except that read() does not
print a warning message when the name requested belongs to a macro
rather than a data set.  In the following, you can substitute 'read()'
for 'matread()'.

No more than 50 numerical items can be on any single line of a data set
read by matread() or read().

                      General description of format
A single file can contain one or more named data sets corresponding to
any type of variable except GRAPH.  This includes REAL, LOGICAL and
CHARACTER data, as well as NULL variables, macros and structures.  Data
sets can have coordinate labels and/or descriptive notes.  See topics
'variables', 'logic', 'NULL', 'notes'.  The file can also have macros
readable by macroread() mixed in among the data sets.  See topics
'macro_files' and macroread().

Every data set must have one or more header lines which give the data
```

set name, information about its structure and internal format. The first line starts with the name and is called the 'name line'. read(fileName, setName) and matread(fileName, setName) search the file for the first line which starts with 'setName'. Such a line is assumed to be the name line for the data set.

'ENDED' header keyword

Any data set may optionally be followed by a line starting '%setName%', where 'setName' is its name. When this is done, keyword 'ENDED' should appear on the name line of the data set. When the data sets has associated labels or notes, '%setName%' should follow them. This usage allows read() and matread() to skip past a data set without examining individual lines. Even if a line in such a data set starts with the name of another data set, read() and matread() will not "see" it and mistakenly treat it as a name line.

End of macros line

A line starting _E_N_D_O_F_M_A_C_R_O_S_ terminates a search for a data set or a macro. You can put help or other information after this line without the danger that a line might be mistaken for the start of a data set.

Format of data set header

The first header line for each data set starts with its name, followed by dimension information and possibly keywords. This line may be optionally followed by descriptive comment lines starting with ')'. These may also provide information on the number of values per line and coding for MISSING values. Thus a data set starts with the following general form:

```
   Name Dims Keywords
   ) 0 or more descriptive or comment lines starting with ')', referred
   ) to as 'comment lines' below
   ) .....
```

Data set name

Name is the name of the data set ('mydata', say) to be matched to setName, the second argument to read() or matread(). Case is ignored in searching for the name, so that read(fileName, "mydata") will find mydata, MyData, or MYDATA, for example.

Data dimensions

Dims, the dimensions of the data set, is a list of 1 or more positive integers. When Dims is a single number ('mydata 20'), the data set is a vector of length Dims or a structure with Dims components. When it is two numbers, nrows and ncols, ('mydata 20 5') the dataset is a nrows by ncols matrix. If Dims consists of p >= 3 numbers, the data set is a p-dimensional array.

It is also acceptable for Dims to be '0', in which case no data is expected. When such a data set is read by read() or matread(), the comment lines are printed and NULL is returned. A useful convention is to have the first "data set" on a file be empty, with the comments describing the remainder of the file. See topic 'NULL'.

Keywords that may be put on the first line of the header

Keyword	Description
CHARACTER	The data set is a CHARACTER variable in either "by fields" or "by lines" format (see below)
COLS or COLUMNS	The data follow in transposed form. For a matrix, this is in column by column order, each column starting on a new line.
ENDED	A line starting '%setName%', where 'setName' is the name of the data set, immediately follows the data set. This does not affect how the data set is read.
FORMAT	Indicates that a Fortran format starting with '(' will follow the last comment line. It is ignored by MacAnova but might help a program written in Fortran to read the data.
LABELS	The data set has coordinate labels which follow the data in the file (see below)
LOCKED	If the result of matread(), macroread() or read() is assigned to a variable, that variable will be locked. See topic 'locks'.
LOGICAL	The data set is a LOGICAL variable, with False and True represented as zero and non-zero values, respectively. Most commonly these are 0 and 1.
NOTES	The data set has attached descriptive notes which follow the data in the file (see below).
NULL	The data set is a NULL variable, containing no data, although there may be comment lines. Dims must be 0. There can be no other keywords. See topic 'NULL'.
QUOTED	The data set is a CHARACTER variable in "by quoted fields" format (see below).
REAL	The data is a REAL variable. This is the default and hence REAL is never required
ROWS	The data follow a row at a time (constant value for first subscript), each row starting on a new line. This is the default and hence ROWS is never required.
STRUCTURE	The data set is a structure.

Upper and lower case letters are not distinguished in these keywords, so, for example, 'macro' and 'Macro' are both recognized as the same as 'MACRO'.

Vector treated like matrix

A vector (single dimension specified) is treated like a matrix with a single column. That is, if COLS or COLUMNS is specified it should all be on one line, and if not, every element must be on a separate line.

Conventions on Comment lines

) LOGICAL	The data are to be interpreted as being LOGICAL, with zero and non-zero values translated to F and T, respectively. This is retained for backward

compatibility and is ignored for CHARACTER, NULL or STRUCTURE data sets.

)"%f %f... %f" specifies a format for each row of a REAL or LOGICAL data set that is analogous to that used by scanf in the C programming language. If the data set is CHARACTER, '%f' is replaced by '%s'; see below. Let N1 and Nk be the first and last dimensions. If there are fewer "%f"'s than Nk (or fewer than N1 if COLS or COLUMNS is specified), then this indicates that, for each value of the last index (first index with COLS or COLUMNS), there are several lines in the file containing data. Each such line, except possibly the last, must have the same number of data items as there are %f's. If no explicit format is given, one with Nk or N1 (if COLS or COLUMNS is on line 1 of the header) %f's is assumed. No more than 50 values can be put on a single line.

)"NNx%f %f ... %f" where NN is an integer, causes the first NN characters of each line to be skipped. This allows you to skip case labels or line numbers. Example:)"12x%f %f" skips 12 characters at the start of each line.

)"%s %s ... %s" specifies a format for each row of a CHARACTER data set. If present, the data will be expected to be in "by fields" format or "by quoted fields" (if QUOTED is on header line) format (see below). The number of "%s"'s is the maximum number of elements that will be read per line.

)"NNx%s %s ...%s" where NN is an integer causes the first NN characters of each line read to be skipped before scanning for CHARACTER data in "by fields" or "by quoted fields" format.

) MISSING XX where XX is a number indicates that XX in the data set is to be read as MISSING. The default missing value code is -99999.9999. Because only integers can be guaranteed to be represented exactly in the computer, it is preferable for XX to be an integer, positive or negative. Example:) MISSING -99 specifies MISSING is coded as -99. This is ignored for data sets that are not REAL or LOGICAL. MISSING must be all upper case.

CHARACTER data formats

There are three possible formats for CHARACTER data, 'by lines', 'by fields' and 'by quoted fields'. 'Quoted' means enclosed in "'s as in "Regression analysis".

By lines

Each element starts on a new line and is not quoted. If an element extends over more than one line, each line except the last must end with '\'. This format is signaled by the presence of CHARACTER on

```
   the header and the absence of any ")%s..."  format among the comment
   lines.

 By fields
   Each stretch of "non-white" characters on a line is considered to be
   an element.  This format is signaled by the presence of CHARACTER on
   the header and the presence of a ")%s..." format among the comment
   lines.  The number of fields on a line is the number of "%s"'s in the
   format.  Commas are treated as non-white characters and do not
   separate fields.

 By quoted fields
   Each element must be enclosed in quotes ("...") and elements in a
   line are separated by spaces, tabs, and possibly a comma.  This
   format is signaled by the presence of QUOTED on the header.  If there
   is no ")%s..." format among the comment lines, the number of items
   expected per line is the size of the last dimension or 1 if the data
   set is a vector.  If COLUMNS or COLS is on the header, the default
   number expected per line is the size of the first dimension.  If
   there is a ")%s..."  format, the number of elements expected per line
   is no more than the number of "%s"'s in the format, with any
   additional lines, except the last, having the same number of
   elements.

                      Format for structure data sets
The name line for a structure data set must be of the form
   strName   ncomps  STRUCTURE
where strName is the name of the structure and integer ncomps > 0 is the
number of components.  There must follow ncomps data sets, each in one
of the formats just described, or in the format for a structure.  Each
component must have a name of the form strName$compName, where compName
is the name of the component.  If a component is a structure, then the
names of its components would thus be strName$compName$compName1.

Each structure component *must* be preceded by at least one blank line.
An individual component can be read like any other data set by
specifying its full name, "mystruc$b", for example.

                      Format for labels and notes
If a variable with name "x", say, has coordinate labels (see topic
'labels'), the header line must contain keyword "LABELS", and the labels
for all coordinates must be in a CHARACTER vector with name "x$LABELS"
immediately following the data associated with x.  When ndims(x) > 1,
the labels for the first dimension come first, followed by those for the
second dimension, and so on, all in one vector.  Thus the length of the
vector normally matches the sum of the dimensions of x.  Because they
are written in the usual form for a CHARACTER vector, you can read the
labels without reading the data by read(fileName, "x$LABELS").

Labels may be "expanded" similarly to the expansion done by setlabels().
Specifically, if the number of x$labels is less than the sum of
dimensions of x, any label starting with '@' or of the form '(', '[',
'{', '<', '/', or '\' is expanded to the length of the appropriate
```

```
dimension.  For example, labels vector("@[", "X1", "X2") for a 10 by 2
matrix are equivalent to vector(rep("@[",10), "X1","X2").  See topics
'labels', setlabels().

If variable x has attached notes (see topic 'notes'), the header line
must contain keyword "NOTES", and the notes must be in a CHARACTER
vector with name "x$NOTES" immediately following the data or labels.
Because notes are written in the usual form for a CHARACTER vector, you
can read them without reading the data by read(fileName, "x$NOTES").

                    Example data file, data.txt

  info           0
  ) Sample data file containing REAL data set sampledata,
  ) CHARACTER data set samplechars, and structure mystruct

  sampledata     4     3 COLUMNS LABELS NOTES ENDED
  ) Small REAL data set with one missing value coded as -99.
  ) Each line contains data for one column (COLUMNS on header)
  ) MISSING -99
  ) '4x' in the following format skips 4 characters (variable label)
  )"4x%f %f %f %f"
  Temp   34.5   45.2  23.1   20.1
  Conc   .170   -99  .883    .401
  Secs    3.5   4.7   3.2     5.8

  sampledata$LABELS    4   QUOTED COLUMNS
  ) Labels for sample data in quoted format by columns
  ) Labels are expanded to "@" "@" "@" "@" "Temp" "Conc" "Secs"
  )"%s %s %s %s"
   "@" "Temp" "Conc" "Secs"

  sampledata$NOTES     1 CHARACTER
  ) Notes for sampledata in "by line" format
  Small REAL data set with one missing value.
  %sampledata%

  samplechars    2     4 CHARACTER
  ) 4 by 2 CHARACTER matrix with each row in 2 lines containing
  ) 3 and 1 unquoted fields
  )"%s %s %s"
  This is by-fields
  format
  without any double
  quotes

  mystruc  2  STRUCTURE
  ) this is a structure with two components, a and b
  ) The blank line before the header of each component is required

  mystruc$a  2 QUOTED COLUMNS
  ) character vector of length 2
  ) Two quoted fields
```

```
"The quick brown fox" "Jumps over the lazy dog"

mystruc$b  2 STRUCTURE
) This component is a structure with two components, pi and e

mystruc$b$pi  1   1
) 1 by 1 matrix
3.14159265358979

mystruc$b$e   1
) vector of length 1
2.71828182845905

                              Examples
            Examples of reading data sets from data.txt

Cmd> sampledata <- read("data.txt","sampledata", quiet:T)

Cmd> print(sampledata) # note that labels were read
sampledata:
            Temp         Conc         Secs
(1)         34.5         0.17          3.5
(2)         45.2      MISSING          4.7
(3)         23.1        0.883          3.2
(4)         20.1        0.401          5.8

Cmd> getnotes(sampledata) # notes were retrieved as well as data
(1) "Small REAL data set with one missing value."

Cmd> notes <- read("data.txt", "sampledata$notes"); notes
(1) "Small REAL data set with one missing value."

Cmd> samplechars <- read("data.txt","samplechars",quiet:T)

Cmd> print(samplechars)
samplechars:
(1,1) "This"
(1,2) "is"
(1,3) "by-fields"
(1,4) "format"
(2,1) "without"
(2,2) "any"
(2,3) "double"
(2,4) "quotes"

Cmd> mystruc <- read("data.txt","mystruc",quiet:T)

Cmd> print(mystruc)
mystruc:
component: a
(1) "The quick brown fox"
(2) "Jumps over the lazy dog"
component: b
```

```
    component: pi
  (1,1)       3.1416
    component: e
  (1)       2.7183

  Cmd> mystruc_b <- read("data.txt", "mystruc$b",quiet:T)

  Cmd> print(mystruc_b)
  mystruc_b:
  component: pi
  (1,1)       3.1416
  component: e
  (1)       2.7183

In these examples, 'quiet:T' suppresses echoing the header line and
comments.  See read() and matread().

                            Cross references
See also topics read(), matread(), matprint(), matwrite(), 'files',
'macro_files'.
```

2.235 matrices

Usage:

Matrix transposition	x' or t(x)
Matrix multiplication	x %*% y, x %c% y, x %C% y
Matrix inversion	solve(a)
Linear equation solution	solve(a, b) or a %\% b, rsolve(a,b) or b %/% a
Extract elements	x[i,j], x[,j], x[i,], i, j integer scalars or vectors or LOGICAL vectors.
Eigen values and vectors	eigen(a), eigenvals(a), releigen(a,b), releigenvals(a,b), trideigen(diag,subdiag...)
Other decompositions	qr(x [,pivot:T]), cholesky(x), svd(x [all:T, right:T or F, left:T or F])
Other Functions of matrices	trace(x), det(x), diag(x), nrows(x), ncols(x)
Create matrices	matrix(x,nrows), hconcat(a,b,...), vconcat(a,b,...), dmat(vec), dmat(x, n)

Keywords: matrix algebra, operations, variables

```
                              Description
A matrix is a two dimensional array, that is, it has two subscripts.

If x is a REAL, LOGICAL, or CHARACTER vector of length m*n, matrix(x,m)
creates an m by n matrix from the elements of x.

A vector of length n is, in most contexts, equivalent to a n by 1
matrix.  See topic 'vectors'.
```

Generalized matrix

A generalized matrix is an array with more than two dimensions but which has no more than 2 dimensions greater than 1. With few exceptions, a generalized matrix can be used wherever a matrix can be used.

A generalized matrix with exactly two dimensions with lengths m > 1 and n > 1, is interpreted as a m by n matrix. For example, array(run(20),1, 4,1,5) is considered for most purposes as if it were a 4 by 5 matrix.

A generalized matrix whose first dimension is n and all others are 1 is iterpreted as an n by 1 matrix or, in some contexts, as a vector of length n. For example, array(run(7),7,1,1,1) is generally treated as either a 7 by 1 matrix or a vector of length 7.

A generalized matrix whose first dimension is 1 and which has a single dimension with length n > 1 is interpreted as a 1 by n matrix, that is, as a row vector. For example, array(run(5),1,1,5) is considered to be a 1 by 5 matrix.

A generalized matrix all of whose dimensions are 1 (example: array(17,1,1,1,1)) is interpreted as a 1 by 1 matrix or, in most contexts, a scalar.

If x is a generalized matrix, ismatrix(x) returns True and nrows(x) and ncols(x) return the numbers of rows and columns as just described.

If x is a generalized matrix, matrix(x) is equivalent to matrix(x, nrows(x)) and is an ordinary two dimensional matrix with the same elements as x.

Transpose of matrix

You can compute the transpose of a matrix x by either x’ or t(x). The transpose of a generalized matrix is a generalized matrix with the same dimensions in reverse order.

Multiplying matrices

You can multiply two REAL matrices or generalized matrices with conforming dimensions and no MISSING values as follows:

Operator	Precedence	Meaning
x %*% y	11	x MatMult y
x %c% y	11	transpose(x) MatMult y
x %C% y	11	x MatMult transpose(y)

where MatMult is ordinary matrix multiplication. The result is always a matrix with two dimensions, even if either x and.or y is a generalized matrix. Either or both operands can also be structures. See topic ’structures’.

It formerly was the case on some computers that, when x and y were large, x’ %c% y was considerably faster than x %*% y or x %C% y’. That is no longer the case; all three operations take about the same amount of time.

Dividing matrices

```
You can "divide" one matrix by another (in the sense of multiplying by
an inverse) if they have conforming dimensions and no MISSING values as
follows:
     Operator   Precedence  Meaning
      x %/% y      11       x MatMult inverse(y) (same as rsolve(y,x)
      x %\% y      11       inverse(x) MatMult y (same as solve(x,y)

Neither %/% or %\% can be used with structures.

Note: These 5 matrix operations are "left associative", that is, for
example that x %*% y %\% z is equivalent to (x %*% y) %\% z, not x %*%
(y %\% z).


                                Precedence
Precedence level 11 is just above the precedence level of '*', '/' and
'%%' and just below the precedence level of '^'.
  Examples:
    Expression          Interpretation     Required to be legal
    a %*% b + 3         (a %*% b) + 3      ncols(a) = nrows(b)
    3 / a %c% b^2       3 / (a %c% (b^2))  nrows(a) = nrows(b)
    a / 3 %C% b         a / (3 %C% b)      ncols(b) = 1

See topic 'precedence' for the precedence levels of other operators.

                     Functions useful with matrices
Here are some functions that are useful with matrices.  All treat
generalized matrices as matrices.
  cholesky()                        Compute Cholesky decomposition of x
  det(x)                            Compute the determinant of x
  det(x,mantexp:T)                  Compute the determinant of x in base
                                    10 mantissa and exponent form
  diag(x)                           Extract the diagonal of x.
  eigenvals(x) and eigen(x)         Compute eigenvalues and/or
                                    eigenvectors of x
  hconcat(x,y,...)                  Concatenate x, y, ... horizontally (y
                                    to the right of x, ...)
  nrows(x), ncols(x)                Find the number of rows or columns of
                                    x
  qr(x [,pivot:T])                  Compute QR decomposition of x
  releigenvals(a,b), releigen(a,b) Compute eigenvalues and/or
                                    eigenvectors of a relative to b
  rsolve(a, b)                      Solve x %*% a = b; equivalent to
                                    b %/% a.
  solve(x)                          Invert x
  solve(a,b)                        Solve a %*% x = b; equivalent to
                                    a %\% b.
  svd(x)                            Compute singular value decomposition
                                    of x
  swp(x,intvec)                     Apply Beaton SWP operator to rows
                                    and columns of x specified by intvec
  t(x)  or  x'                      Transpose of x
  trace(x)                          Compute the trace of x
```

```
  trideigen(diag,subdiag [,...])   Compute eigenvalues and/or
                                   eigenvectors of symetric tridiagonal
                                   matrix
  vconcat(x,y,...)                 Concatenate x, y, ... vertically (y
                                   below x, ...)

                         Other useful functions
The following are also useful, but they do not treat a generalized
matrix x exactly like matrix(x).
  max(x)                           Maximum of each column of x
  min(x)                           Minimum of each column of x
  prod(x)                          Product down columns of x
  sum(x)                           Sum down columns of x

These all operate on the first actual dimension of x, producing a result
with the same number of dimensions as x, but with first dimension 1.  If
you want to treat a generalized matrix as if it were a matrix, use, say,
sum(matrix(x)).  See help on the functions for information on how to
operate on dimensions other than the first.

                            Complex matrices
MacAnova stores complex data in REAL matrices in two formats, fully
complex and packed Hermitian.  See topic 'complex' for details of the
format and information about functions useful with complex data.

The following are macros explicitly for working with complex matrices A
and B stored in REAL matrices a and b
  cmatmultc(a,b [,op:OP)    matrix product A %*% B, A %c% B or A %C% B
  ctranspose(a)             A'
  cjtranspose(a)            conj(A)'
  cdiag(a)                  diag(A)
  ctrace(a)                 trace(A)
  ceigen(a)                 eigenvalues and eigenvectors of Hermitian A
  csolve(a)                 inverse of non-singular A
  csubscr(a [,i [,j])       simulated A[], A[i], A[i,], A[,j] or A[i,j]

                            Cross references
See also det(), trace(), swp(), eigen(), eigenvals(), releigen(),
releigenvals(), dim(), nrows(), ncols(), svd(), cholesky().
```

2.236 matrix()

Usage:

```
matrix(x,Rowdim [,KeyPhrases]), x a vector, Rowdim > 0 an integer
  dividing length(x)
matrix(x [,KeyPhrases]), x a generalized matrix.
KeyPhrases can be labels:structure(rowLabs,colLabs), notes:Notes and
  silent:T, where rowLabs, colLabs and Notes are CHARACTER scalars or
  vectors.
```

Keywords: matrix algebra, variables, combining variables

Usage

matrix(x,Rowdim) creates a matrix (two dimensional array) with Rowdim rows containing the data in x. x can be a vector, matrix, or higher dimensional array, all of whose elements are used, with first subscript changing fastest, second subscript, if any, changing next, and so on.

Rowdim must be a positive integer exactly dividing the length of vector, matrix, or array x.

Example

Example:

```
  Cmd> c <- matrix(vector(1,1,1, -1,1,0, -1,-1,2),3)
```

creates the following matrix:

```
            1   -1   -1
       c  = 1    1   -1 .
            1    0    2
```

No dimensions supplied

matrix(x), with no Rowdim, is equivalent to matrix(x,nrows(x)). It is valid for any one- or two-dimensional x, or for any higher dimensional array with no more than two dimensions greater than 1, that is for any x such that 'ismatrix(x)' would be True. See topics ismatrix() and 'matrices'.

Example: h <- matrix(SS[2,,]) creates a true p by p matrix from the 1 by p by p array SS[2,,], if SS is an array of SSCP matrices created as a side effect by manova(). See manova().

Although most operations, including matrix multiplication, matrix inversion, and eigenvalue computation, treat SS[2,,] and matrix(SS[2,,]) identically, there are a few that do not. Using matrix() can avoid some surprises.

Keywords 'labels' and 'notes'

On both usages, you can specify row and column labels for the output using keywords labels. See topic 'labels' for details.

You can attach a CHARACTER vector of descriptive notes to the result using keyword phrase 'notes:Notes'. See topic 'notes' for details.

When x is a matrix and either Rowdim is not specified or nrows(x) = Rowdim, any coordinate labels or descriptive notes of x are transferred to the result unless 'labels' or 'notes' provide new labels or notes or are NULL.

Cross references

See also topics array(), nrows(), 'matrices'.

2.237 matwrite()

Usage:

```
matwrite(fileName, a, b, ... [, new:T, format:Fmt, nsig:n, sep:sepChar,\
  quoted:T or bylines:T,missing:mVal, name:Name, comments:charVec,\
  width:w, header:F, oldstyle:T, stripdols:T]), a, b, ... arbitrary
  variables, Fmt, sepChar and Name CHARACTER scalars with sepChar only a
  single character, charVec a CHARACTER vector or scalar, mVal a REAL
  scalar, w >= 30 integer.
```

Keywords: files, output

```
                                  Usage
matwrite(FileName,a,b,... [,new:T]) writes REAL, LOGICAL and CHARACTER
variables a, b,...  (scalars, vectors, matrices, or arrays) file
FileName in a form which can be read by read() and matread().  It can
also write NULL variables and structures.  GRAPH variables are legal
arguments but are currently written as NULL variables.  See topic
'NULL'.

matwrite(a,b,...,file:FileName [,new:T]) is an alternative usage.

                    matwrite() compared with matprint()
matwrite() differs from matprint() only in the default format used for
REAL and LOGICAL variables.  It uses the format used by write(), namely
the format specified by option 'wformat'.  This normally provides 9
significant digits.  It is provided to make it easier for you to write
data sets with increased precision without having explicitly to provide
a format.

                          Changing default format
You can change the default format for write() and matwrite() by
setoptions() using keyword 'wformat'.  See topics setoptions() and
'options'.

                             Cross references
See matprint() for information on keywords 'missing', 'sep', 'name',
'header', 'width', 'quoted', 'bylines', and 'comments', macrowrite() for
information on 'oldstyle' and 'stripdols', and print() for information
on keywords 'nsig' and 'format'.

See topic 'matread_file' for a description of the file format.

See also topics print(), write(), matwrite(), macrowrite(), read(),
matread(), 'files'.
```

2.238 max()

Usage:

```
max(x [,squeeze:T] [,silent:T,undefval:U]), x REAL or LOGICAL or a
  structure with REAL or LOGICAL components, U a REAL scalar
max(x, dimensions:J [,squeeze:T] [,silent:T,undefval:U]), vector of
  positive integers J
max(x, margins:K [,squeeze:F] [,silent:T,undefval:U]), vector of
  positive integers K
max(x1,x2,... [,silent:T,undefval:U]), x1, x2, ... REAL or LOGICAL
  vectors, all the same type.
```

Keywords: descriptive statistics

```
                                    Usage
max(x) computes the maximum of the elements of a REAL or LOGICAL vector
x.

If x is LOGICAL, True is interpreted as 1.0 and False as 0.0 and hence
max(x) is 1.0 if any element of x is True, and 0.0 if all elements are
False.

If x is a m by n matrix, max(x) computes a row vector (1 by n matrix)
consisting of the maxima of the elements in each column of x.

If x is an array with dimensions n1, n2, n3, ..., y <- max(x) computes
an array with dimensions 1, n2, n3, ... such that y[1,j,k,...] =
max(x[i,j,k,...], i=1,...,n1).  This is consistent with what happens
when x is a matrix.  Note: MacAnova3.35 and earlier produced a result
with dimensions n2, n3, ... .

max(x, squeeze:T) does the same, except the first dimension of the
result (of length 1) is squeezed out unless the result is a scalar.  In
particular, if x is a matrix, max(x,squeeze:T) will be identical to
vector(max(x)), and if x is an array, max(x,squeeze:T) will be identical
to array(max(x),dim(x)[-1]).

max(NULL) is NULL.

max(a,b,c,...) is equivalent to max(vector(a,b,c,...)) if a, b, c,
... are all vectors.  They must all have the same type, REAL or LOGICAL,
or be NULL.  max(NULL,NULL,...,NULL) is NULL.

min(x, silent:T) or min(a,b,c,...,silent:T) does the same but suppresses
warning messages about MISSING values.

If all the elements of a vector x are MISSING, max(x) is MISSING.

max(x, undefval:U), where U is a REAL scalar does the same, except the
returned value is U when all the elements of x are MISSING.

                            Keyword 'dimensions'
max(x, dimensions:J [,squeeze:T] [,silent:T] [undefval:U]) finds the
maximum over the dimensions in J = vector(j1,j2,...,jn) where j1, ...,
```

jn are distinct positive integers <= ndims(x). Without 'squeeze:T', the result has the same number of dimensions as x, with dimensions j1, j2, ..., jn of length 1. With 'squeeze:T', these dimensions are removed from the result. The order of j1, j2, ... is ignored.

It is an error if max(J) > ndims(x) or if there are duplicate elements in J.

For example, if x is a matrix, max(x, dimensions:2) computes the row maxima as a nrows(x) by 1 matrix and max(x, dimensions:2,squeeze:T) computes them as a one dimensional vector.

Keyword 'margins'

max(x, margins:K [,squeeze:F] [,silent:T] [undefval:U]) finds the maxima over the dimensions not in K = vector(k1, k2, ..., km), where k1, ..., km are distinct positive integers <= ndims(x). This computes maxima for the margins specified in K.

Without 'squeeze:F', only the dimensions in K are retained in the result. Otherwise the other dimensions are retained but have length 1. This is opposite from the default with 'dimensions:J'.

It is an error if max(K) > ndims(x) or if there are duplicate elements in K.

Structure argument

If x is a structure, max(x [dimensions:J or margins:K] [,squeeze:T or F] [,silent:T] [undefval:U]) computes a structure, each of whose components is max() applied to that component of x.

Example

Examples:

```
  Cmd> x # matrix with labels
              B1          B2
  A1          18          15
  A2          17          26
  A3          18          19

  Cmd> max(x) # maxima down columns
               B1          B2
  (1)          18          26

  Cmd> max(x,dimensions:2) # maxima across rows; 3 by 1 matrix
             (1)
  A1          18
  A2          26
  A3          19

  Cmd> max(x,dimensions:2,squeeze:T) # same, length 3 vector
            A1          A2          A3
            18          26          19

  Cmd> max(x,margins:1) # same as preceding
```

```
           A1          A2          A3
           18          26          19

                                Example
See also topics min(), 'NULL'.
```

2.239 memory

Keywords: general

```
                          Memory usage information
The MacAnova "workspace", (all variables and macros; see topic
'workspace'), "resides" in memory (RAM) and not on your hard disk.

You can use keyword phrase size:T on list() to get information on the
amount of memory used by individual variables and the total used by all
variables and internal MacAnova storage.  This does not include the
memory required for the program itself nor, in a windowed version,
memory associated with windows and graphs.  See list().

memoryinfo() returns a vector summarizing several aspects of memory
usage.  See memoryinfo() for details.

                          Not enough memory problem
Because RAM is a finite resource, you may sometimes be unable to carry
out a computation because there is no room for intermediate and/or final
results.  When this happens you get a message similar to the following:
   ERROR: Not enough memory.  Try deleting variables

When this happens, use delete() to get rid of the largest variables you
can do without.  If you really need to keep them, use save() or
asciisave() to save some or all of your variables on disk before
deleting them.
  Cmd> save("myvars.sav", x, y, residplot:LASTPLOT)

  Cmd> delete(x, y, LASTPLOT)

                            Other work arounds
You can free up memory used to store information related to the most
recent GLM command such as regress() or anova() by
  Cmd> delete(STRMODEL)
This will make certain functions such as secoefs() and modelinfo()
unavailable until after another GLM command.

In Windowed versions, memory can sometimes be made available by closing
unneeded graphics and/or command/output windows.

                           Comparison of versions
The amount of memory available for individual variables and the entire
workspace differs between versions, primarily because of differences in
operating systems.
```

All versions have no effective limit on the size of variables other than the availability of memory on your computer.

The Macintosh classic (OS 9) version runs in its own memory "partition". When MacAnova is launched, its partition is allocated a specific amount of space for program and data and MacAnova cannot exceed the limit, even if a lot more memory is installed. When the limit is, say, 1000 KB (1 KB is 1024 bytes), then MacAnova has N KB for variables, where, because memory is needed for the program, N in the neighborhood of 600. When the limit is 2500 KB = (1000 + 1500) KB, then MacAnova has N + 1500 KB of memory for its workspace. When the partition is allocated 5000 KB = (1000 + 4000) KB, available memory is N + 4000 KB, and so on.

Changing Macintosh partition size

As distributed, the Mac OS 9 version of MacAnova has a default maximum partition size of 5000 KB. However, it is quite easy to change the limit from the Finder. Open the folder where MacAnova is installed and select the MacAnova icon with the mouse. Select Get Info on the File menu to display a dialog box containing a 'Show:' popup menu on which you should select 'Memory'. This will display a Memory Requirements panel. (In older systems, the Memory Requirements panel may be immediately displayed by Get Info.) Type a new number in the "Suggested size" box to set a new maximum partition size and then close the dialog box. That's all there is to it.

Cross references

See also topics delete(), save(), asciisave(), 'workspace'

2.240 memoryinfo()

Usage:

```
memoryinfo()
```

Keywords: general

Usage

info <- memoryinfo() sets info to a vector of length 5 containing the following information about memory usage:

- info[1] Total memory used by named variables, excluding built-in functions but including macros and special variables such as CLIPBOARD and GRAPHWINDOWS
- info[2] Total memory currently used by scratch variables. This includes the results of any computations not yet in named variables but not scratch variables used in invoking memoryinfo()
- info[3] Total memory currently used by internal copies of what was typed at the prompt
- info[4] Total memory allocated by MacAnova. This excludes memory required by the program itself and some memory that is

```
            allocated at the start and kept for the entire run.  On
            windowed system, it excludes memory for command and graphics
            windows and menus.
  info[5]   The maximum memory so far allocated by MacAnova, always >=
            info[4]

The values returned are the number of bytes used or allocated.

memoryinfo() is primarily useful in (a) tracking down bugs in memory
allocation by MacAnova itself, and (b) determining why a macro uses more
memory than you think it ought to.

                              Cross references
See also topic 'memory'.
```

2.241 min()

Usage:

```
   min(x [,squeeze:T] [,silent:T,undefval:U]), x REAL or LOGICAL or a
     structure with REAL or LOGICAL components, U a REAL scalar
   min(x, dimensions:J [,squeeze:T] [,silent:T,undefval:U]), vector of
     positive integers J
   min(x, margins:K [,squeeze:F] [,silent:T,undefval:U]), vector of
     positive integers K
   min(x1,x2,... [,silent:T,undefval:U]), x1, x2, ... REAL or LOGICAL
     vectors, all the same type.
```

Keywords: descriptive statistics

```
                                    Usage
min(x) computes the minimum of the elements of a REAL or LOGICAL vector
x.

If x is LOGICAL, True is interpreted as 1.0 and False as 0.0 and hence
min(x) is 0.0 if any element of x is False, and 1.0 if all elements are
True.

If x is a m by n matrix, min(x) computes a row vector (1 by n matrix)
consisting of the minima of the elements in each column of x.

If x is an array with dimensions n1, n2, n3, ..., y <- min(x) computes
an array with dimensions 1, n2, n3, ... such that y[1,j,k,...] =
min(x[i,j,k,...], i=1,...,n1).  This is consistent with what happens
when x is a matrix.  Note: MacAnova3.35 and earlier produced a result
with dimensions n2, n3, ... .

min(x, squeeze:T) does the same, except the first dimension of the
result (of length 1) is squeezed out unless the result is a scalar.  In
particular, if x is a matrix, min(x,squeeze:T) will be identical to
vector(min(x)), and if x is an array, min(x,squeeze:T) will be identical
to array(min(x),dim(x)[-1]).
```

```
min(NULL) is NULL.

min(a,b,c,...) is equivalent to min(vector(a,b,c,...)) if a, b, c,
... are all vectors.  They must all have the same type, REAL or LOGICAL,
or be NULL.  min(NULL, NULL, ..., NULL) is NULL.

min(x, silent:T) or min(a,b,c,...,silent:T) does the same but suppresses
warning messages about MISSING values.

If all the elements of a vector x are MISSING, min(x) is MISSING.

min(x, undefval:U), where U is a REAL scalar does the same, except the
returned value is U when all the elements of x are MISSING.

                          Keyword 'dimensions'
min(x, dimensions:J [,squeeze:T] [,silent:T] [undefval:U]) finds the
minimum over the dimensions in J = vector(j1,j2,...,jn) where j1, ...,
jn are distinct positive integers <= ndims(x).  Without 'squeeze:T', the
result has the same number of dimensions as x, with dimensions j1, j2,
..., jn of length 1.  With 'squeeze:T', these dimensions are removed
from the result.  The order of j1, j2, ... is ignored.

It is an error if max(J) > ndims(x) or if there are duplicate elements
in J.

For example, if x is a matrix, min(x, dimensions:2) computes the row
minima as a nrows(x) by 1 matrix and min(x, dimensions:2,squeeze:T)
computes them as a one dimensional vector.

                           Keyword 'margins'
min(x, margins:K [,squeeze:F] [,silent:T] [undefval:U]) finds the minima
over the dimensions not in K = vector(k1, k2, ..., km), where k1, ...,
km are distinct positive integers <= ndims(x).  This computes minima for
the margins specified in K.

Without 'squeeze:F', only the dimensions in K are retained in the
result.  Otherwise the other dimensions are retained but have length 1.
This is opposite from the default with 'dimensions:J'.

It is an error if max(K) > ndims(x) or if there are duplicate elements
in K.

                          Structure argument
If x is a structure, min(x [dimensions:J or margins:K] [,squeeze:T or F]
[,silent:T] [undefval:U]) computes a structure, each of whose components
is min() applied to that component of x.

                                 Example
Examples:
  Cmd> x # matrix with labels
              B1          B2
  A1          18          15
```

```
  A2          17          26
  A3          18          19

  Cmd> min(x) # minima down columns
               B1          B2
  (1)           17          15

  Cmd> min(x,dimensions:2) # minima across rows; 3 by 1 matrix
              (1)
  A1           15
  A2           17
  A3           18

  Cmd> min(x,dimensions:2,squeeze:T) # same, length 3 vector
            A1          A2          A3
            15          17          18

  Cmd> min(x,margins:1) # same as preceding
            A1          A2          A3
            15          17          18

                             Cross references
See also topics max(), 'NULL'.
```

2.242 modelinfo()

Usage:

```
   modelinfo([all:T] keyword1:T or F, keyword2:T or F ...[,nomodelok:T]\
     [,missing:missvalue]), missvalue a REAL scalar and the keywords
     are one or more of 'aliased', 'bitmodel', 'coefs', 'colcount',
     'distrib', 'link', 'parameters', 'scale', 'sigmahat' 'strmodel',
     'termnames', 'weights', 'xtxinv', 'xvars', and 'y'
```

Keywords: glm

```
                                    Usage
modelinfo(keyword1:T, keyword2:T, ... ) computes one or more vectors or
matrices associated the most recent GLM (generalized linear or linear
model) command such as regress(), anova(), poisson(), or glmfit().  This
gives you direct access to such things as the design variables or
X-variables (xvars:T), the estimated coefficients of the X-variables
(coefs:T), and the inverse of the X'X matrix (xtxinv:T).

Permissible keyword names specifying quantities returned are 'aliased',
'bitmodel', 'coefs', 'colcount', 'distrib', 'link', 'parameters',
'scale', 'sigmahat' 'strmodel', 'termnames', 'weights', 'xtxinv',
'xvars', and 'y'.  You can also use 'all:T'; see below.

You can't use modelinfo() after fastanova(), ipf(), or screen().

Any component requested that is not available is set to NULL.  In
```

particular this happens for components 'coefs' and 'xtxinv' after anova() when the model is balanced, or after any GLM command with 'coefs:F', and for component parameters after any GLM for which a sample size or other parameter is specified such as for logistic() and probit().

When more than one of the keywords specifying type of output has value True, modelinfo() returns a structure with component names the same as the keywords. Otherwise it returns a vector or matrix.

Keyword 'missing'

modelinfo(xvars:T, ..., missing:missval) returns the matrix of X-variables associated with the active model with all the values for a case in which there was missing data set to REAL scalar missval. The default value is 0.

Keywords 'all' and 'nomodelok'

modelinfo(all:T) is equivalent to modelinfo(xvars:T,y:T,coefs:T, xtxinv:T,colcount:T,weights:T,parameters:T,strmodel:T,bitmodel:T, termnames:T, scale:T, sigmahat:T,aliased:T). To suppress any particular components, say 'strmodel', 'bitmodel', and 'termnames', use modelinfo(all:T, strmodel:F,bitmodel:F,termnames:F). Use of all:F is an error.

Normally, it is an error if there is no active GLM model. However, if nomodelok:T is an argument, when there is no active model modelinfo() returns NULL without printing an error message. For example, you can test for the existence of an active model by

```
Cmd> if(isnull(modelinfo(strmodel:T,nomodelok:T))){...do something...}
```

You can use 'nomodelok:T' with any of the keywords specifying components to return.

See isnull().

Permissible Keyword Phrases

Keyword 'aliased'

aliased:T
A LOGICAL vector whose length is the number of X-variables in the model. The i-th variable is True if and only if the i-th X-variables as returned by xvars:T is aliased with previous X-variables. If there was no aliasing every element should be False.

Keyword 'bitmodel'

bitmodel:T
A REAL vector or matrix with as many rows as there are terms in the model, including the CONSTANT term, if any, but excluding the final error term. This encodes the model in a special form. See below for details.

Keyword 'coefs'

```
coefs:T
  The vector of coefficients of the X-variables in the fitted model.
  Coefficients corresponding to aliased X-variables (those that are
  apparently linearly dependent on previous X-variables) are set to
  zero.  After manova() with a p-dimensional response matrix, the
  coefficients form a matrix with p columns.  Note: If there are
  factors in the model, some of the coefficients computed will differ
  from the coefficients computed by coefs() and secoefs().

                          Keyword 'colcount'
colcount:T
  A REAL vector containing the numbers of X-variables assocated with
  each term in the active model.  The numbers of the first column
  associated with each term can be obtained by autoreg(1,
  modelinfo(colcount:T)).  If no X-variables are aliased with earlier
  X-variables, these values are the degrees of freedom associated
  with each term.

                         Keyword 'parameters'
parameters:T
  A REAL vector containing the sample sizes or other distribution
  parameters after logistic() or probit(), or after glmfit() with
  keywords n or parameters.

                           Keyword 'scale'
scale:T
  A REAL factor or factors to multiply the square roots of the
  diagonals of the inverse of the X'X matrix so as to obtain
  estimated standard errors for the estimated coefficients.  After
  logistic(), poisson(), probit(), or glmfit(), scale will be the
  default value, unless changed by keyword 'scale' on the GLM
  command.  After other GLM commands, including robust(), scale will
  be sqrt(SSerror/DFerror), where DFerror and SSerror come from the
  final line of the ANOVA table.  After manova(), scale will be the
  vector consisting of the square roots of diagonal elements of
  SSerror/DFerror, where SSerror is the error matrix.

                          Keyword 'sigmahat'
sigmahat:T
  The robust estimate of sigma printed after the robust() ANOVA
  table.  It is not usable after any other GLM command.  Note that
  this is not a suitable value to use in computing standard errors.

                          Keyword 'strmodel'
strmodel:T
  A CHARACTER variable containing the current model taken from
  STRMODEL.

                          Keyword 'termnames'
termnames:T
  A CHARACTER vector of the terms in the model taken from TERMNAMES.
  This includes the name (usually "ERROR1") of the final error term,
  and thus the length of the result is 1 greater than the number of
```

rows in modelinfo(bitmodel:T).

Keyword 'weights'

weights:T
A REAL vector containing the weights associated with each case. If there are no missing data and no weights were specified, either explicitly or implicitly, this is a vector of 1's. Otherwise it contains either the weights specified by keyword 'weights' or 'wts' in anova(), manova() or regress() or the implicit weights from the final iteration of poisson(), logistic(), or robust(). The weight for any case with MISSING data is always zero.

Keyword 'xtxinv'

xtxinv:T
The inverse of the X'X matrix computed from the X-variables. The row and column corresponding to any aliased X-variable is set to zero. If the previous GLM command specified weights either explicitly (keyword 'weights' or 'wts' on anova(), regress(), manova()) or implicitly (poisson() or logistic()) the matrix computed is the inverse of X'WX, where W is the diagonal matrix of the weights. After robust(), the matrix computed is the inverse of X'X, ignoring the implicit weights. The weights may be obtained by keyword phrase weights:T (see above).

Keyword 'xvars'

xvars:T
The matrix of X-variables associated with the active model. If there is no active model, but STRMODEL is defined and there are no other keywords, it works identically to xvariables().

When 'missing:missValue' is an argument, where missValue is a REAL scalar, the X-variable values for a case with any missing data will be set to missValue. In particular, you can used 'missing:?' to set the X-variables for a case with MISSING data to MISSING. The default value is 0. See xvariables() for more information.

Keyword 'y'

y:T
The dependent variable in the model as a vector or matrix. modelinfo(y:T) is thus equivalent to modelvars(0).

More on keyword 'bitmodel'

Each row of modelinfo(bitmodel:T) corresponds to a term in the model and consists of one or more integers between 0 and 4294967295 = 2^32 - 1, the bits of whose binary representation encode the variates and/or factors in that term. If there are 33 to 64 or 65 to 95 variates and factors, the result is a matrix with 2 or 3 columns. Thus each row of the output has room for Nvar bits, where Nvar is the number of variates and/or factors in the model.

The bits of each row should be considered to be numbered from 1 to Nvar. Bit 1 is the least significant and bit 32 is the most significant bit of the first element; bit 33 is the least significant and bit 64 is the

```
most significant bit of the second element, if any; and bit 65 is the
least significant and bit 96 is the most significant bit of the third
element, if any.  Bit i of row j of the result is 1 if and only if the
i-th variable or factor is in the j-th term of the model, following the
order in which variables and factors first appear in the model.  All the
elements in a row corresponding to the CONSTANT term (usually row 1) are
zero.

                            Cross references
See topics 'bit_ops' and nbits() for information on how to extract
information from the result of modelinfo(bitmodel:T).

See also topics varnames(), modelvars(), xvariables(), coefs(),
secoefs(), glmpred(), glmtable(), regpred(), predtable(), popmodel(),
pushmodel(), 'models'.
```

2.243 models

Usage:

```
   Regression:  regress("y=x1+x2+...+xk")
   One-way ANOVA: anova("y=A"), factor A
   Randomized block ANOVA: anova("y=Repl+A"), factors Repl and A
   Nested ANOVA: anova("y=A/B") or anova("y=A+A.B")
   Two-way factorial: anova("y=A*B") or anova("y=A+B+A.B"), factors A and B
   Completely randomized Split plot ANOVA: anova("y=A+E(Repl.A)+B+A.B"),
     factors A, B, and Repl
   Analysis of covariance: anova("y=x+A"), factor A, variate x
   Transform variables on the fly: regress("{log10(y)}={sqrt(x)}")
   Polynomial regression: regress("y=P3(x)")
   Periodic regression: regress("y=C2(2*PI*hour/24)")
```

Keywords: glm, anova, regression

```
                            Form of GLM model
All of the GLM (generalized linear or linear model) commands such as
regress(), anova(), or poisson() require you to specify a model in a
quoted string or CHARACTER variable.

A model can be specified as
   "Response = Term"   or   "Response = Term1 + Term2 + ..."
where 'Response' is the name of the dependent or response variable
and each term is of the form Name or Name1.Name2. ... .Namek, where
Name, Name1, Name2 ... are the names of variables.  A period or dot (.)
between the variable names is interpreted as a product operator
indicating that all combinations of the variable values are included in
the model.

                            Model Variables
Variables in model terms, including those computed on the fly (see
below), may be either factors (vectors of positive integers created
using factor()) or variates.  No more than one variate may appear in a
```

single term. Up to 95 variables may appear in a model, including no more than 31 factors.

Factors and variates must be vectors or matrices with one column. They must all have the same number of rows as the response variable.

Any factors must have been created using factor() or been selected from such a variable using subscripts. For balanced designs with factor levels in a reasonable order a factor may often be computed by factor(rep(run(r),s)), factor(rep(run(s), rep(r,s))), or something similar. See factor(), rep().

The constant term may be specified as 1, but is always included by default, that is "y = Model" is equivalent to "y = 1 + Model". You can omit a constant term by "y = Model - 1" or move it to the end by "y = Model - 1 + 1".

Computing Variables "on the fly"

You can transform or otherwise compute model variables "on the fly." In place of the name of a variable, including the response variable, you can use {Expr}, where Expr is a MacAnova expression such as x^2 or log10(y). If the same expression, say {sqrt(x)}, appears more than once in a model, it is evaluated only once and only one model variable is introduced. In comparing expressions, leading and trailing spaces are ignored, so that { sqrt(x) } is considered the same as {sqrt(x)}; however, other differences in the presence or placement of spaces will cause expressions to be considered different variables. For example, {sqrt(x)} is not recognized to be the same as {sqrt(x)}. The only limitation on Expr is that it may not directly or indirectly execute another GLM command.

Since subscripted factors remain factors (see 'subscripts'), when groups is a factor, anova("{y[-3]} = {groups[-3]}") computes a one factor analysis of variance omitting case 3.

Examples

a and b are factors and y, x1, x2, and x3 are REAL vectors

Model	Description
"y = a + b + a.b"	Two factor model with both main effects and interaction
"y = a + a.b"	Two factor model with b nested in a
"y = x1 + x2 + x3"	Three variable multiple regression
"{sqrt(y)} = x1 + {x1^2}"	2nd order polynomial regression of square root of y on x.

Shortcuts for Polynomial and Periodic Regression

You can use special short cuts of the form Pn(expr) and Cn(expr) to specify a polynomial term or a periodic term, respectively, where n is an integer between 1 and 95 and expr is a MacAnova expression. For example, P4(x-10) expands to ({x-10}+{(x-10)^2}+{(x-10)^3}+{(x-10)^4}) and C2(2*PI*x/24) expands to ({cos(2*PI*x/24)}+{sin(2*PI*x/24)}+ {cos(2*(2*PI*x/24))}+{sin(2*(2*PI*x/24))}).

Pn(expr) and Cn(expr) can be used wherever a variable name can be used on the right side of '=', except not in a {...} expression. Thus the last example in the preceding list could have been written "{sqrt(y)} = P2(x1)". They can be "dotted" with a factor. For example, P2(x).a expands to {x}.a + {(x)^2}.a.

If you are doing a regression on a subset of cases uses subscripts, the subscripts must be applied to x, not Cn(x) or Pn(x). For example,

```
  Cmd> regress("{y[-run(3)]} = P3(x[-run(3)])")
```

fits a cubic polynomial omitting the first 3 rows of x and y.

See below for other shortcuts you can use to specify models.

Combining Variables

Parts of terms can be replaced by 'submodels', enclosed in parentheses, for example,

```
  Cmd> anova("y = a + b + c + d + (a + b).(c + d)")
```

is equivalent to

```
  Cmd> anova("y = a + b + c + d + a.c + b.c + a.d + b.d")
```

The product of a factor or variate with itself (a.a) is equivalent to the variate or factor itself. For example,

```
  Cmd> anova("y = (a + b).(a + c)")
```

is equivalent to

```
  Cmd> anova("y = a + b.a + a.c + b.c")
```

The order of factors and variates in a term is immaterial. That is a.b is equivalent to b.a.

Order of terms

The order of terms in a model is very important since fitting a model is done sequentially, term by term. For example, although "y = a + a.b" is a model with b nested in a, "y = a.b + a" is computationally equivalent to "y = a.b", since after fitting all combinations of a and b, there is nothing left for 'a' to fit.

If a term in a model is duplicated, only the first occurrence is retained. For example, (a + b).(a + b) expands to a.a + b.a + b.a + b.b which is equivalent to a + a.b + a.b + b which is trimmed to a + a.b + b (which is computationally equivalent to a + a.b).

Order of terms in expanded models

If M1, M2, ..., Mk, N1, N2, ..., Nl are terms or submodels,
(M1 + M2 + ... + Mk).(N1 + N2 + ... + Nl) is equivalent to
M1.N1 + M2.N1 + ... + Mk.N1 + M1.N2 + M2.N2 + ... + Mk.N2 + ... + Mk.Nl

If M1, M2, ..., Mk are terms or submodels, M1.M2.M3. ... .Mk is expanded as (...((M1.M2).M3) ...).Mk.

Short cut formulas for combining terms or submodels

In the following, M1, M2, ... are terms or submodels.

M1*M2 is an abbreviation for M1 + M2 + M1.M2

M1*M2* ... *Mk is an abbreviation for (...((M1*M2)*M3) ...)*Mk. In particular, M1*M2*M3 is an abbreviation for (M1*M2)*M3, that is for M1 + M2 + M1.M2 + M3 + M1.M3 + M2.M3 + M1.M2.M3

M1^N is an abbreviation for M1.(1+M1). ... .(1+M1), where there are N factors. N must be a digit between 1 and 31. This contains the same terms as M1*M1*...*M1 (N factors) but in a different order. For example, (a+b+c+d)^4 has main effects followed by 2-way interactions followed by 3-way interactions followed by a.b.c.d. Note that M1^N is usually not equivalent to and does not contain the same terms as M1. ... .M1 (N dot factors).

M1/M2 is an abbreviation for M1 + MM1.M2 where MM1 has the form a.b. ... .z, where a, b, ..., z are all the factors and/or variates in M1. For example, (a+b+c)/(d+e) is equivalent to a+b+c+a.b.c.d+ a.b.c.e. Note: Earlier versions of help and other documentation had a different definition which was correct only in some common simple cases.

M1 - M2 is an abbreviation for a model containing all the terms in M1, omitting any term in M2. In particular Model - 1 specifies a model with no constant term or intercept and Model - 1 + 1 specifies a model with a contant term that is fit after all other terms in Model.

M1 -* M2 is an abbreviation for a model containing all the terms in M1, but omitting any terms containing all the variables in any term of M2.

Examples of use of shortcuts

"y = a*b" is equivalent to "y = a + b + a.b"
"y = a/b" is equivalent to "y = a + a.b"
"y = a*b*c" is equivalent to "y = a + b + a.b + c + a.c + b.c + a.b.c"
"y = (a+b+c)^2" is equivalent to "y = a + b + c + a.b + a.c + b.c"
"y = (a+b+c)^3" is equivalent to "y = a + b + c + a.b + a.c + b.c + a.b.c"
"y = a*b*c - a.b.c" is equivalent to "y = a + b + a.b + c + a.c + b.c"
"y = a*b*c -* (a.b + a.c)" is equivalent to "y = a + b + c + b.c"

Note the order of the exanded terms. In particular, observed that, although "y=a*b*c" and "y=(a+b+c)^3" contain the same terms when expanded, they are in a different order.

Error Terms

In the output from commands such as anova() or poisson() that produce an analysis of variance or deviance table, there is always one line, usually labeled "ERROR1", following all the terms explicitly or implicitly specified in Model. It consists of the sum of squares or deviance associated with all the degrees of freedom not included in the model. If the model fitted uses up all the degrees of freedom this line will still be present, but will have 0 degrees of freedom.

```
You can also label other terms as ERROR.  If a term is of the form
E(Term) (for example, E(a.b.c)), it will be labeled "ERRORn" in the
ANOVA table, where n is 1, 2, ..., .  The final error line will still be
printed but will be labeled "ERRORm", where m-1 is the number of error
terms you specified.  E(1) is not legal, nor is it legal to specify a
term as an error term more than once (E(a.b) + E(a.b)).  Moreover, once
a term is designated as an error term, it cannot be deleted by '-' or
'-*'.  Term in E(term) must be a single factor or a pure product of
factors.  For example, E(a.b+a.b.c) is illegal.

                            Comments in models
A '#' in Model marks the end of the model, allowing models to be self-
documenting as in anova("y = a + b #additive model").

                            Variable STRMODEL
Any GLM command sets the CHARACTER variable STRMODEL to the specified
model as a "side effect" of the analysis.  If no model is specified on a
subsequent GLM commands (for example anova()), it is taken from this
variable.  Alternatively, if you set STRMODEL directly, for example
  Cmd> STRMODEL <- "y = x1 + x2 + x3"
then the value of STRMODEL will be used by the next GLM command if it
has no model as argument.  Note, however, when you assign a value to
STRMODEL, MacAnova discards the internal information saved by the most
recent GLM command that is used by functions such as secoefs() and
contrast().

                          Examples of GLM Models
  Cmd> anova("y = a + b + a.b") # or anova("y = a*b")
will produce a two-way analysis of variance with interaction for the
response in y, provided vector y is defined and a and b are factors with
the same length as y.

  Cmd> anova("y = a + a.b") # or anova("y = a/b")
where a and b are factors will produce a nested analysis of variance
with b nested within a.

  Cmd> anova("y = blk + a + E(a.blk) + b + a.b")
would be appropriate for the analysis of a two factor split plot
experiment with the whole plot treatments in a randomized block design.
Do not attempt to use the name 'rep' for a blocking factor, since 'rep'
is the name of a built-in operation.
```

2.244 modelvars()

Usage:

```
modelvars(varList [,Model]), varList a vector of integers >= 0, Model a
  CHARACTER scalar
modelvars(y:T or x:T or variates:T or factors:T or all:T [, Model])
modelvars(nx:T or nvariates:T or nfactors:T or hasconst:T [, Model])
```

Keywords: glm

Usage

modelvars(VarList,Model), where VarList is a vector of non-negative integers, say vector(i1,i2,i3,...), returns a vector or matrix whose columns are the variables in the model specified by the scalar CHARACTER variable or quoted string Model. Variable 0 is the dependent variable (the variable before '=' in Model) and, if i > 0, variable i is the i-th variate or factor appearing on the right hand side of Model (after '=').

See topic 'models' for information on specifying Model.

Keywords 'x' and 'y'

modelvars(y:T,Model) returns a vector or matrix containing the dependent variable of Model. This usage yields the same result as modelvars(0, Model).

modelvars(x:T,Model) returns a vector or matrix containing the independent variates and factors of on the right hand side of Model. This yields the same as modelvars(run(nv), Model), where nv is the number of variates and factors. When there are no variates and factors ("y=1"), NULL is returned. See topic 'NULL'. See keyword 'nx' below for determining the total number of variates and factors.

Keywords 'factors' and 'variates'

modelvars(factors:T,Model) returns a vector or matrix containing the factors on the right hand side of Model. When there are no factors in the model, NULL is returned. See keyword 'nfactors' below for determining the total number of factors.

modelvars(variates:T,Model) returns a vector or matrix containing the variates on the right hand side of Model. When there are no variates in the model, NULL is returned. See keyword 'nvariates' below for determining the total number of variates.

Keyword 'all'

modelvars(all:T,Model) returns a matrix containing the dependent variable followed by the independent variates and factors of Model. Equivalent to this is modelvars(run(0,nv),Model).

Omitted model

When Model is omitted (modelvars(VarList) or modelvars(keyword:T)), variables are taken from internal copies of the variables in the current active model. This allows retrieval of the dependent variable and/or model variables even if they were temporary variables (their names started with '@'). When there is no active model but variable STRMODEL exists, modelvars(keyword:T) and modelvars(VarList) are equivalent to modelvars(keyword:T,STRMODEL) and modelvars(VarList,STRMODEL). Here keyword is one of 'x', 'y', 'factors', 'variates', or 'all'.

Examples

```
modelvars(vector(1,2,0),"y=x+a") is equivalent to hconcat(x,a,y)
modelvars(x:T,"y=x+a+a.x") is equivalent to hconcat(x,a)
modelvars(all:T,"y=x1+x2") is equivalent to hconcat(y,x1,x2)
```

```
Note: Any variables that are factors are returned unchanged.  This is
very different from xvariables(), which computes dummy X-variables
associated with a factor.

                    Counting Factors and Variables
You can also use modelvars() to determine how many factors and variates
there are in a model or to check whether the constant term is in the
model.  This can be useful in a macro using the results of a GLM command
to do further analyses.

modelvars(nx:T [,Model]) returns the number of independent variates and
factors on the right hand side of Model, that is, what would be computed
by ncols(modelvars(x:T [,Model])).  When there are no variates and
factors ("y=1"), 0 is returned.

modelvars(nfactors:T [,Model]) returns the number of factors on the
right hand side of Model, that is, what would be computed by
ncols(modelvars(factors:T, [,Model])).  When there are no factors 0 is
returned.

modelvars(nvariates:T [,Model]) returns the number of variates on the
right hand side of Model, that is, what would be computed by
ncols(modelvars(variates:T [,Model])).  When there are no variates 0 is
returned.

modelvars(hasconst:T [,Model]) is True if and only if the constant term
is in the model.

                                 Examples
  Cmd> x <- run(4);y <- rnorm(4)

  Cmd> a <- factor(1,1,2,2); b <- factor(1,2,1,2)

  Cmd> modelvars(nfactors:T,"y=x+a+b+a.x")
  (1)           2

  Cmd> modelvars(nvariates:T,"y=x+a+b+a.x")
  (1)           1

  Cmd> modelvars(nx:T,"y=x+a+b+a.x")
  (1)           3

  Cmd> modelvars(hasconst:T, "y=x")
  (1) T

  Cmd> modelvars(hasconst:T, "y=x-1")
  (1) F

                            Cross references
See also topics 'models', varnames(), xvariables().
```

2.245 more()

Usage:

```
more(x [, nsig:n, format:Fmt, missing:M ,stripdol:T]), where x is a
  macro or is a REAL, CHARACTER, or LOGICAL variable, n > 0 is an
  integer, Fmt and M are CHARACTER scalars
```

Keywords: output, general

```
                                  Usage
more(x) displays object x using a 'paging' program that displays a
screenful at a time.  On Unix/Linux by default it uses Unix/Linux
program 'more'.  If variable PAGER exists and is a CHARACTER scalar,
then it is assumed to specify a paging program.  For example, on some
Unix/Linux systems, if the value of PAGER is "less -x4", then more()
invokes Unix/Linux program 'less' with tab stops set every 4 positions.

By default, when x is a macro, more(x) displays it after stripping off
'$$' from temporary variable names.  Use more(x,stripdol:F) To see the
actual '$$' in the macro.

When x is REAL, you may use any of keywords 'nsig', 'format', and
'missing' as on print(), but not 'name' and 'file'.

The lines written to the screen are not written to a spool file, nor are
they redisplayed after a plot.  See spool().

More is implemented as a pre-defined macro and is not available in all
versions of MacAnova.
```

2.246 Mouse()

Usage:

```
Str <- Mouse([getpoints:T or getlines:T or getbox:T] [,xyonly:T or n:N]\
  [,cancelok:T] [graphics keyword phrases]), positive integer N <= 20
```

Keywords: plotting

```
                              Introduction
Mouse() provides a way to get x-y coordinates from a graphics window so
that you can add points, lines, boxes or character information at points
whose x- and y-coordinates are chosen interactively.

Mouse() is implemented only in windowed versions and on Unix/Linux
versions implementing Tektronix 4014 emulation.  (Of course, it cannot
workout a Tektronix terminal emulator that allows graphical input (GIN)
mode.)

In a Unix/Linux version with Tektronix emulation, Mouse() automatically
activates the graphics screen.  In windowed versions, you have to select
the graphics window where you want to get x-y coordinates.  In some
versions, pressing 'q' at any time before the operation is finished
```

aborts Mouse() and pressing 'r' restarts it, forgetting any coordinates already selected.

Usage

The basic usages of Mouse() are as follows

Usage	Returns
Str <- Mouse([xyonly:T])	x-y coordinates of 1 point
Str <- Mouse(getpoints:T [,xyonly:T])	x-y coordinates of 1 point
Str <- Mouse(getpoints:T, n:N [,xyonly:T])	x-y coordinates of N points
Str <- Mouse(getlines:T [,xyonly:T])	2 endpoints of a line
Str <- Mouse(getlines:T, n:N [,xyonly:T])	N+1 points defining segmented line
Str <- Mouse(getbox:T [,xyonly:T])	x-y coordinates of the corners of a rectangular box, plus the first corner repeated.

N must be an integer between 1 and 20.

With each of these usages, you can also use keyword phrase 'cancelok:T'; without 'cancelok:T', cancelling Mouse() by pressing 'q' is considered an error; with 'cancelok:T', pressing 'q' causes Mouse() immediately to return NULL.

Result

After you select one or more positions in graphics window I, Str becomes a structure whose first two components, 'x' and 'y', are REAL scalars or vectors containing the x- and y-coordinates being returned.

With keyword phrase 'xyonly:T', 'x' and 'y' are the only components. Otherwise the structure also has component 'window' with integer value I, and component 'add', a LOGICAL scalar with value True. With getlines:T and getbox:T, 'lines' is an additional LOGICAL scalar component having value True.

Graphics keyword use

When 'xyonly:T' is not an argument, you can include additional "extra" keyword phrases, usually graphics keyword phrases, after the Mouse() keyword phrases just described. These will be made part of the output structure. For example, Mouse(getbox:T,linetype:2,thickness:3) returns structure(x:xcoord, y:ycoord, window:I, add:T,lines:T, linetype:2, thickness:3). If keyword 'add' is among these "extra" keywords, Mouse() does not include its own component 'add' in the structure returned, and similarly for keywords 'lines' and 'window'. For example,

```
Cmd> Str <- Mouse(getlines:T, symbols:"\1",add:F, lines:F)
```

sets Str to structure(x:xcoord, y:ycoord, window:I, symbols:"\1", add:F, lines:F).

Comparison of versions

Using Mouse() differs slightly among different versions. On windowed versions you select positions by clicking and releasing the mouse button with the cursor or pointer in a graphics window. In a version running

under Xterm on a Unix/Linux workstation, you need to press Enter or Return after releasing the mouse button. On other Tektronix terminal emulators pressing the mouse button may be sufficient. On some emulators, it may not work at all.

The action of Mouse() starts when you click and release the mouse with the cursor or pointer in a graphics window. The action then taken depends on whether 'getpoints:T', 'getlines:T' or 'getbox:T' is an argument to Mouse() (default is 'getpoints:T'). The action ends when you depress and release the button the final time. In the windowed versions, Mouse() draws temporary lines as you move the mouse in the window, erasing them before it returns.

In windowed versions, at any time before the final click you can restart selection of locations by moving the cursor outside the window. Temporary lines or marks are erased and you can select a different window if you want.

In windowed versions, you can interrupt and terminate the action at any time by selecting Interrupt on the File menu or pressing Ctrl+I (Command+I on Macintosh). In unwindowed versions, pressing the interrupt key (usually Ctrl-C) terminates the action.

Keyword 'getpoints'

Mouse(getpoints:T [,n:N] ...) and Mouse([n:N] ...).
As you move the mouse, cross hairs (horizontal and vertical lines) appear and follow the cursor until you click and release at which time the point selected is marked temporarily. When N > 1, the cross hairs appear again until you click and release to mark the next point. This continues for N points. After the final click, Mouse() erases the marked points and returns structure(x:xcoord, y:ycoord, window:I, add:T) where xcoord and ycoord are the x- and y-coordinates of the point or points selected.

Mouse(getpoints:T,xyonly:T [,n:N]) or Mouse(xyonly:T [,n:N]) returns only structure(x:xcoord, y:ycoord).

Keyword 'getlines'

Mouse(getlines:T [,n:N])
As you move the mouse until you click and release it again, a continuously updated line is drawn between the current position of the mouse and the first location clicked. When N > 1, another continuously updated line is drawn between the current position and the second location selected, and so on until N connected line segments have been drawn. After the final click and release, Mouse() erases the lines drawn and returns structure(x:xcoord, y:ycoord, window:I, add:T, lines:T) where xcoord and ycoord contain the x- and y-coordinates of the N+1 points defining the segmented line.

Mouse(getlines:T,xyonly:T [,n:N]) returns only structure(x:xcoord, y:ycoord).

If you press the shift key while tracing a line, the line being drawn

```
 if forced to be either horizontal or vertical.

                              Keyword 'getbox'
Mouse(getbox:T)
 As you move the mouse, a continuously updated rectangular box is drawn.
 One corner is at the first position clicked on and the opposite corner
 is the current position.  After a second and final click and release,
 Mouse() returns structure(x:vector(x1,x1,x2,x2,x1), y:vector(y1, y2,
 y2,y1,y1),window:I, add:T,lines:T), where (x1,y1) is the initial
 position clicked on and (x2,y2) is the final position of the opposite
 corner.  Note that the initial position is repeated as the last point
 and that the points trace out the entire border of the box.  Keyword
 phrase 'n:N' is illegal with 'getbox:T'.

 Mouse(getbox:T,xyonly:T) returns only
   structure(x:vector(x1,x1,x2,x2,x1),y:vector(y1,y2,y2,y1,y1))

 If you press the shift key while moving the mouse, the rectangle drawn
 is forced to be square (not in Tektronix emulation).

The form of the output from Mouse() is designed to make it easy inter-
actively to add information to a plot, either using one of the graphics
commands (see topic 'graphs') or assignment to GRAPHWINDOWS (see topic
'graph_assign').

                                  Examples
                                  Examples
  Cmd> s <- Mouse(getpoints:T, symbols:"*") # or s <- Mouse(symbols:"*")

  Cmd> GRAPHWINDOWS[s$window] <- s

This plots "*" at the position selected with the cross hairs.  The
second command can be replaced by addpoints(GRAPHWINDOWS[s$window],
keys:s).

  Cmd> s <- Mouse(getlines:T, linetype:2); GRAPHWINDOWS[s$window] <- s

This draws in the window the line determined by two click and releases
of the mouse in a graphics window.

  Cmd> s <- Mouse(getbox:T); GRAPHWINDOWS[s$window] <- s

This draws in the window the box determined by the mouse positions.

                            Use with GRAPHWINDOWS
If you know the window you will draw into, say window 1, you can plot a
point, line or box simply by GRAPHWINDOWS[1] <- Mouse(KEY:T ...), where
KEY is 'getpoint', 'getline' or 'getbox':

  Cmd> s <- Mouse(getlines:T, n:5, linetype:2,thickness:3)

  Cmd> addlines(GRAPHWINDOWS[s$window], keys:s)
```

```
This draws 5 connected line segments with line type 2 and thickness 3.

  Cmd> s <- Mouse(getlines:T, show:F)

  Cmd> addlines(GRAPHWINDOWS[s[3]], keys:s[-3])

  Cmd> addstrings(s$x[1],s$y[1], "Interesting feature",\
       window:s$window, justify:"l")

This draws the line and then draws "Interesting feature" at the first
position clicked on.  Component 'window' is omitted in addlines() (by
using s[-3] as an argument) because 'window:n' can't be used with
'show:F'.

                            Cross references
See also 'structures', strconcat(), addlines(), addstrings().
```

2.247 movavg()

Usage:

```
movavg(Theta,A [,reverse:T, limits:vector(i1 [,i2]), start:startVals,\
  seasonal:L]), REAL vector or NULL Theta, REAL vector or matrix A, REAL
  startVals the same size and shape as A, positive integer L
```

Keywords: time series

```
                              Introduction
movavg() is designed to implement a moving average operator as the term
is used in ARIMA time series analysis.  For a more ordinary moving
average, convolve() is preferable.  movavg() can also be used to compute
differences of a series or, together with autoreg(), to find the power
series coefficients of rational functions.

                                  Usage
movavg(Theta,A) applies the moving average operators specified by the
columns of the REAL matrix Theta to the columns of the REAL matrix A.
If ncols(Theta) = 1, Theta is applied to every column of A and if
ncols(A) = 1, each column of Theta is applied to A.  The result is a
matrix with nrows(A) rows and max(ncols(Theta), ncols(A)) columns.  If
both Theta and A have more than one column, they must both have the same
number of columns.

Specifically, assuming for simplicity that both Theta and A are vectors
so that the result x is a vector, then
    x[i] = A[i] - sum(Theta[k]*A[i-k],1<=k<=nrows(theta)),
with A[l] taken to be 0 for l < 1.

When Theta is a vector, movavg(Theta,A) can be expressed in matrix terms
as Theta1 %*% A, where Theta1 is a nrows(A) by nrows(A) matrix.  For
example, when nrows(Theta) = 2,
         [   1          0        0       0   ...    0        0       0 ]
```

```
         [-Theta[1]     1        0       0   ...    0        0       0 ]
Theta1 = [-Theta[2] -Theta[1]    1       0   ...    0        0       0 ]
         [   0      -Theta[2] -Theta[1]  1   ...    0        0       0 ]
         [ ..................................................... ]
         [   0          0        0       0   ... -Theta[2] -Theta[1] 1 ]
```

NOTE: The sign assumed for Theta is not affected by variable MASIGN which is recognized by several macros in file Arima.mac. Type arimahelp(MASIGN) for details.

If Theta is NULL, the result is the same as A, stripped of labels or notes, if any. Also, the result is a true vector or matrix (ndims = 1 or 2).

First and higher differences

A common usage is movavg(1,A), where A is a vector or matrix. This computes the first differences A[1,] - 0,A[2,]-A[1,], ...,A[n,]-A[n-1,]. Second differences can be computed by movavg(vector(2,-1),A), third differences by movavg(vector(3, -3, 1), A), and so on.

Keywords 'reverse' and 'seasonal'

movavg(Theta,A,reverse:T) applies the moving average operator in reverse:

    x[i] = A[i] - sum(Theta[k]*A[i+k],1<=k<=nrows(phi))

with A[l] = 0 for l > nrows(A).

movavg(Theta,A,seasonal:L [,reverse:T) does the same, except that the computations are of the forms

    x[i] = A[i] - sum(Theta[k]*A[i-k*L],1<=k<=nrows(Theta)).

Keywords 'limits' and 'start'

movavg(Theta,A,limits:vector(i1,i2),start:StartVals [,reverse:T, seasonal:L]) is the same except that x[i] is computed as just described only for i1 <= i <= i2, with the remaining values copied from rows 1 to i1-1 and rows i2+1 to nrows(A) of matrix StartVals.

The value for limits can also be a scalar j between 1 and nrows(A). In this case, with reverse:T, i1 = 1, i2 = j, and without reverse:T, i1 = j, i2 = nrows(A).

StartVals must have the same number of columns as A and usually has the same number of rows. When nrows(StartVCals) != nrows(A), without reverse:T, i2 must be nrows(A) and with reverse:T, i1 must be 1. In this case, the elements of StartVals are copied to the rows not included between i1 and i2 and hence nrows(start) must match nrows(A) - (i2 - i1 + 1).

Unlike what happens with autoreg(), the values computed for rows i1 to i2 are unaffected by the values of StartVals.

Examples

Examples (theta and theta1 vectors of same length):

```
  Cmd> m <- nrows(theta); n <- 300
```

```
  Cmd> movavg(theta,rnorm(n+m))[-run(m)]
     generates a moving average series with normal innovations.
  Cmd> movavg(theta,matrix(rnorm(10*(n+m),10))[-run(m),]
     generates 10 independent moving average series
  Cmd> movavg(hconcat(theta,theta1),rnorm(n+m))[-run(m)]
     generates two moving average series with the same innovations
  Cmd> movavg(.3,movavg(-.1,rnorm(230),seasonal:4))[-run(30)] generates
      a (0,0,1)x(0,0,1)-4 seasonal ARMA time series

                       Relationship with autoreg()
movavg() is the inverse of autoreg() and vice versa, in that
   movavg(phi,autoreg(phi,x))   and   autoreg(phi,movavg(phi,x))
both reproduce x, except for rounding error.

                            Cross references
See also autoreg().
```

2.248 mulvarhelp()

Usage:

```
   mulvarhelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
   mulvarhelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
     Subtopics
   mulvarhelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
   mulvarhelp(key:Key), CHARACTER scalar Key
   mulvarhelp(index:T [,scrollback:T])
```

Keywords: general, multivariate analysis

```
                                  Usage
mulvarhelp(Topic1 [, Topic2, ...]) prints help on topics Topic1, Topic2,
... related to macros in file mulvar.mac.  The help is taken from file
mulvar.mac.

mulvarhelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these macros.

mulvarhelp(index:T) or simply mulvarhelp() prints an index of the topics
available using mulvarhelp().  Alternatively, help(index:"mulvar") does
the same thing.

mulvarhelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER
scalar or vector, prints subtopics of topic Topic.  With subtopic:"?", a
list of subtopics is printed.

mulvarhelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where
Suptopics1 and Subtopics2 are CHARACTER scalars or vectors, prints the
specified subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to mulvarhelp().  In windowed
```

```
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

                              Keyword 'key'
mulvarhelp(key:key) where key is a quoted string or CHARACTER scalar
lists all topics cross referenced under Key.  mulvarhelp(key:"?") prints
a list of available cross reference keys for topics in the file.

mulvarhelp() is implemented as a predefined macro.

                            Cross references
See help() for information on direct use of help() to retrieve
information from mulvar.mac.
```

2.249 nameof()

Usage:

```
   nameof(var1, var2, ... )
```

Keywords: variables, character variables

```
                            Usage and example
nameof(arg1, arg2, ..., argk) returns a CHARACTER vector containing the
names of the arguments.  If an argument is the result of a computation
or is a quoted string, its name will be descriptive.

If an argument is missing, the name returned is "".

Example:
  Cmd> nameof(x,cos,17,3+5,vector(x),matrix(run(10),5),,"Hello")
  (1) "x"
  (2) "cos"
  (3) "NUMBER"
  (4) "NUMBER"
  (5) "VECTOR"
  (6) "MATRIX"
  (7) ""
  (8) "STRING"

                            Cross references
See also compnames(), isname(), typeof(), varnames(), rename()
```

2.250 nbits()

Usage:

```
   nbits(x), where x consists of 1 or more integers between 0 and
     4294967295
```

Keywords: operations, transformations, glm

```
                                  Usage
nbits(x), where x is an integer with value between 0 and 4294967295
(2^32-1), computes the number of non-zero bits in the binary
representation of x.  For example, nbits(123455) is 11 since 123455 has
binary representation 00000000000000011110001000111111b.

If x is not an integer or x < 0 or x > 4294967295, a warning message is
printed and the result is set to MISSING.

If x is a REAL vector, matrix or array or a structure all of whose
components are REAL, nbits(x) is a variable or structure of the same
size and shape as x, each element of which is the number of bits in the
corresponding element of x.

nbits() is useful with the output of modelinfo(bitmodel:T).

                                 Examples
Examples:
After anova("y=(a+b)^2 + ((a+b)^2).x",silent:T) the following commands
compute the number of variables or variates in term 5 and in all terms:

  Cmd> nvars5 <- sum(vector(nbits(modelinfo(bitmodel:T)[5,]))); nvars5
  (1)           2

  Cmd> vector(sum(nbits(modelinfo(bitmodel:T)')),labels:TERMNAMES[-8])
    CONSTANT           a           b         a.b         a.x         b.x
       a.b.x
           0           1           1           2           2           2
           3

                             Cross references
See also topics 'bit_ops', modelinfo().
```

2.251 ncols()

Usage:

```
   ncols(x) where x is a matrix or generalized matrix
```

Keywords: variables

```
                             Usage and examples
ncols(x) returns the number of columns of matrix argument x.  If x has
more than two dimensions, no more than two dimensions may exceed 1 and
it is treated as described in topic 'matrices'.

If x is a structure, ncols(x) is a structure.  If xi is the i-th
component of x, the i-th component of ncols(x) is ncols(xi).

                                 Examples
```

```
  Cmd> x <- run(7); vector(nrows(x),ncols(x),nrows(x'),ncols(x'))
  (1)           7           1           1           7

  Cmd> y <- array(run(24),1,6,1,4);vector(nrows(y),ncols(y))
  (1)           6           4

  Cmd> ncols(structure(x,y))
  component: x
  (1)           1
  component: y
  (1)           4

                            Cross references
See also topics nrows(), dim(), ndims(), length(), 'structures'.
```

2.252 ncomps()

Usage:

```
   ncomps(Str) where Str is a structure
```

Keywords: variables, structures

```
                            Usage and example
ncomps(Str) returns the number of components in structure Str.  It is an
error if Str is not a structure.

Example:
  Cmd> ncomps(describe(run(10))) # describe has structure result
  (1)           8

                            Cross references
See also topics ndims(), dim(), length(), isstruc(), describe(),
'structures'.
```

2.253 ndims()

Usage:

```
   ndims(x)
```

Keywords: variables, null variables

```
                                 Usage
ndims(x) computes the number of dimensions of x.  If x is a vector,
ndims(x) is 1; if x is a matrix, ndims(x) is 2, even if has column
dimension 1.

If x is a NULL variable, ndims(x) is 0.
```

```
If x is a structure, ndims(x) is a structure.  If x_i is the i-th
component of x, the i-th component of ndims(x) is ndims(x_i).

                              Examples
Examples:
  Cmd> x <- run(7); vector(ndims(x),ndims(x'), ndims(x''))
  (1)           1           2           2

  Cmd> y <- array(run(24),1,6,1,4); ndims(y)
  (1)           4

  Cmd> ndims(structure(x,y))
  component: x
  (1)           1
  component: y
  (1)           4

                           Cross references
See also topics length(), dim(), 'NULL'.
```

2.254 next

Usage:

```
for(i,run(n)){if(x[i] < 0){next} .... }
for(i,run(n)){for(j,run(m)){if(x[i,j] < 0){next 2} .... }}
```

Keywords: control, syntax

```
                                  Usage
'next', when used in a 'while' or 'for' loop, skips to just before the
'}' that terminates the loop, ignoring any intervening commands.
Execution resumes immediately before the '}'.

'next n', where n is a positive integer, skips to the end of the n-th
enclosing loop.  For example, 'next 1' is equivalent to 'next' and will
skip to the end of the current loop; 'next 2' will exit the current loop
and skip to the end of the loop enclosing it; and so on.  n must be a
literal integer ('1', '2', ...) and not a variable with integer value.

It is an error to use 'next' outside of a loop or to use 'next n' when
not enclosed in at least n loops.

                     In a macro or evaluated string
In an evaluated string or out-of-line macro, 'next' can be used only to
skip to the end of a loop that started in the macro or evaluated string.
It is an error to try to skip to the end of a loop that started outside
the macro or evaluated string.  See evaluate() and 'macros'.

Using 'next' in an in-line macro to skip to the end a loop that started
outside the macro will work, but is a bad programming practice.
```

```
                                Examples
Examples:
  for(i,run(100)){... compute x ...;if(x<0){next};... do something ...;}
Whenever the computed x becomes negative, no further computation is done
for that value of i.

  for(i,run(10)){for(j,run(5)){...;if(x<0){next 2}; ... }; ...}
Whenever x becomes negative on any pass through the j loop, that loop is
terminated and execution resumes before the '}' which ends the i loop.

                             Cross references
See also topics 'if', 'for', 'while', 'break', 'breakall', batch().
```

2.255 notes

Usage:

```
   This topic has information on" notes" attached to variables.
   Functions for working with such notes are:
     attachnotes(x, Notes) # attach Notes to x
     attachnotes(x, NULL)  # remove notes from x
     notes <- getnotes(x)  # retrieve notes from x
     if (hasnotes(x)){...do something with notes...}
     y <- vector(x, notes:Notes) # create vector with notes Notes
     y <- matrix(x, notes:Notes) # create matrix with notes Notes
     y <- array(x, notes:Notes)  # create array with notes Notes
     plot(x,y [,... ],notes:Notes)     # Notes attached to LASTPLOT
     chplot(x,y,symbols:ch [,...],notes:Notes) # Notes attached to LASTPLOT
     lineplot(x,y [,...], notes:Notes) # Notes attached to LASTPLOT
     showplot([...,] notes:Notes)      # Notes attached to LASTPLOT
```

Keywords: general, macros, variables

```
                          Using notes on variables
You can attach CHARACTER vectors as "notes" to almost any variable,
including GRAPH variables and macros.  These can be used to record
descriptions of data or plots or usage notes for macros.

When x is an existing variable

  Cmd> attachnotes(x, vector("Heights of Stat 1001 students",\
     "collected Fall 1997"))

attaches the descriptions to variable x.

You can append additional notes to a variable using appendnotes():

  Cmd> appendnotes(x, "There were 29 women and 21 men")

You retrieve notes from a variable by, for example,
```

```
  Cmd> getnotes(x)
  (1) "Heights of Stat 1001 students"
  (2) "collected Fall 1997"
  (3) "There were 29 women and 21 men"
```

Use with inforead()

One useful trick is to use inforead() to retrieve the comment lines associated with a data set in a file readable by read() and matread() (see topics 'matread_file' and inforead()) and then use attachnotes() to attach them to the matrix:

```
  Cmd> y <- read("macanova.dat", "irisdata", quiet:T)

  Cmd> attachnotes(y, inforead("macanova.dat", "irisdata", quiet:T))
```

Keyword 'notes'

You can also attach notes using keyword phrase 'notes:Notes' on vector(), matrix(), array(), structure(), macro() or any of the plotting commands, where Notes is a CHARACTER scalar or vector. For example,

```
  Cmd> x <- array(x, notes:Notes) # Notes a CHARACTER vector

  Cmd> plot(x, y, notes:"Plot of height vs weight")
```

Notes in files

Commands matprint(), matwrite() and macrowrite() automatically write any attached notes in a form that is readable by read(), matread() and macroread(). See topic 'matread_file'

Propagation of notes

Generally notes do not "propagate" except in a few situations when the result of a function is essentially the same as an argument to that function. Here are the specific situations when a copy of any notes attached to x is attached to y.

```
    y <- x
    y <- vector(x [,labels:Labels]), when isvector(x) is True
    y <- matrix(x [,labels:Labels]), when ismatrix(x) is True
    y <- matrix(x, nrows(x) [,labels:Labels]), when ismatrix(x) is True
    y <- array(x, [,labels:Labels])
    y <- array(x,dim1,dim2,.. [labels:Labels]), when all the new
         dimensions match those of x
    y <- strconcat(x [,labels:Labels] [,compnames:Names]) when
         x is a structure.
```

Cross references

See also topics attachnotes(), getnotes(), appendnotes(), vector(), matrix(), array(), structure(), macro(), list(), 'graph_keys'.

2.256 nrows()

Usage:

```
nrows(x), x a matrix or generalized matrix
```

Keywords: variables

```
                          Usage and examples
nrows(x) returns the number of rows of matrix argument x.  If x has more
than two dimensions, no more than two dimensions may exceed 1 and it is
treated as described in topic 'matrices'.

Examples:
  Cmd> x <- array(run(24),1,6,1,4); y <- run(7)

  Cmd> vector(nrows(x),ncols(x))
  (1)           6           4

  Cmd> vector(nrows(y),ncols(y))
  (1)            7            1

  Cmd> vector(nrows(y'),ncols(y'))
  (1)           1           7

If x is a structure, nrows(x) is a structure.  If xi is the i-th
component of x, the i-th component of nrows(x) is nrows(xi).

                          Cross references
See also topics ncols(), dim(), ndims(), length(), 'structures'.
```

2.257 NULL

Usage:

```
   x <- NULL creates a NULL variable
   isnull(x) tests whether a variable is NULL.
```

Keywords: null variables, variables

```
                             Description
A NULL variable is a special type of variable.  Unlike REAL, LOGICAL or
CHARACTER variables, a NULL variable contains no data.  You might think
of it as an completely empty variable.  Many functions and commands such
as anova(), regress() and print() that are primarily executed for their
"side effects" return a NULL variable as value.

                       Working with NULL variables
You can explicitly create a NULL variable by
   Cmd> nullvar <- NULL
or
   Cmd> nullvar <- print("Hello!") # value of print() is NULL
```

```
You can use isnull() to test whether a variable is NULL.

  Cmd> isnull(NULL, PI, T, "hello", nullvar)
  (1) T       F       F       F       T

See isnull() for details.

                              Limitations
For obvious reasons, you can't do arithmetic or comparisons with NULL
variables and most commands and functions do not accept NULL variables
as arguments.

                            NULL arguments
A few functions such as vector(), hconcat(), vconcat(), sum(), prod(),
min() and max() do accept NULL arguments.  For example, vector(NULL,a,b)
and min(NULL,a,b) are equivalent to vector(a,b) and min(a,b),
respectively.  Here is an example where this might be useful.

  Cmd> x <- run(10); fstats <- NULL # or fstats <- vector(NULL)

  Cmd> for(i,run(1000)){regress("{.1*x+rnorm(10)}=x",silent:T)
       fstats <- vector(fstats,SS[2]/(SS[3]/DF[3]));;}

This creates a random sample of F-statistics based on a regression of y
on x where y = .1*x + rnorm(10) (see regress(), 'models', rnorm())
Although fstats starts out NULL, by the end of the first trip through
the loop, it contains the first F-statistic.  Without NULL variables,
the loop would have to be something like the following:

  Cmd> for(i,run(1000)){
     regress("{x+rnorm(10)}=x",silent:T); fstats <- \
      if(i==1){SS[2]/(SS[3]/DF[3])}else{vector(f,SS[2]/(SS[3]/DF[3]))}}

A better way to implement this example would probably be initialize by
fstats <- rep(0,1000), and save each value by fstats[i] <-
SS[2]/(SS[3]/DF[3]).

                            Cross references
See also topics 'for' and 'if'.
```

2.258 number

Keywords: variables, syntax, missing values

```
                    Entering numbers and missing values
You enter numbers as integers with or without out a decimal point, as
decimal numbers, or using exponential notation (X.XXeY or X.XXEY is XX.X
* 10^Y).

  Cmd> a <- 65535 # same as a <- 65535. or a <- 65535.0000
```

```
  Cmd> a <- -3141.592654; b <- .0000415; c <- 1000000

  Cmd> a <- -3.141592654e3; b <- 4.15E-5; c <- 1e6# same as preceding
```

For greater readability of long numbers, you can use '_' to separate digits in the mantissa (the entire number without 'e' or 'E' or the part before 'e' or 'E').

```
  Cmd> a <- -3_141.592_654e4 # same as a <- -3141.592654e4
```

Warning: _3_141, for example, is a legal variable name, not a number.

Fortran style double precision numbers like -3.14592654d3 and 4.15D-5 are read as if the 'd' or 'D' were 'e' (except by read() and matread()).

It is an error to attempt to enter a number that is too large to be represented in the computer. For example,

```
  Cmd> d <- 3.1e5000
```

is an error. On most computers the largest numbers are about +-2^1024 = +-1.79769e+308 and the smallest nonzero numbers are about +-2^(-1024) = 5.56268e-309.

You enter MISSING values using the symbol '?'.

```
  Cmd> e <- vector(1,2,3,?,4,5,?) # vector with 2 MISSING values.
```

Other representations for MISSING such as '*' and '.' which are recognized by vecread() are not recognized in MacAnova commands. However, since NA is a predefined locked REAL scalar with value MISSING,

```
  Cmd> e <- vector(1,2,3,NA,4,5,NA) # vector with 2 MISSING values
```

works fine (unless you have deleted NA).

Numbers and missing values in output

By default, most numeric output is printed using a hybrid between integer, decimal and exponential format. MISSING values are normally printed as 'MISSING'.

```
  Cmd> vector(100*PI, 1e6*PI, PI/100, PI/1e6)
  (1)       314.16   3.1416e+06     0.031416   3.1416e-06

  Cmd> vector(34, ?)
  (1)           34      MISSING
```

You can change these defaults by setoptions() keywords 'format' and 'missing'. Here is an example:

```
  Cmd> setoptions(missing:"NA", format:"12.4f")

  Cmd> vector(100*PI, 1e6*PI, PI/100, PI/1e6)
```

```
  (1)     314.1593 3141592.6536       0.0314       0.0000

  Cmd> vector(34, ?)
  (1)      34.0000           NA

See topics setoptions(), 'options', print().

                             Cross references
See also topic 'syntax'.
```

2.259 options

Usage:

```
setoptions(option1:value [,option2:value ... ] [,badoptok:T]) option1,
  option2, ...  names of options, sets option values
getoptions(option1:T [,option2:T ... ] [,badoptok:T]), option1, option2,
  ...  option names of options, retrieves option values
List of options that may be set and retrieved
  Option name          Values (* = default)
  angles               "radians"*, "degrees" or "cycles"
  batchecho            T* or F
  dumbplot             T* or F
  errors               integer >= 0
  findmacros           "yes"*, "silent" or "no"
  font                 quoted string ("McAOVMonaco"*) [Mac]
  fontsize             integer > 0 (9) [Mac]
  format               quoted string ("12.5g"*)
  fstats               T or F*
  height               integer >= 0 (screen size*)
  history              integer >= 0 (100*)
  inline               T or F*
  keyboard             integer >= 0 (2*) [Windows]
  labelabove           T or F*
  labelstyle           "("*, "[", "{", "<", "/", "\\"
  lines                integer >= 0 (screen size*) [same as 'height']
  matchdelay           integer >= 0 [windowed versions only]
  maxlinelen           integer >= 80
  maxwhile             integer >= 10 (1000*)
  minpvalue            0 <= number <= .001 (1e-8)
  missing              quoted string <= 20 characters ("MISSING"*)
  nsig                 0 < integer <= 20 (5*)
  prompt               quoted string <= 20 characters ("Cmd> "*)
  pvals                T or F*
  quiet                T or F*
  restoredel           T* or F
  savehistry           T* or F [not limited memory DOS]
  scrollback           T or F* [windowed versions only]
  seeds                2 integers > 0 or both 0*
  tekset               CHARACTER vector of length 2 [non-windowed
                       Unix/Linux only]
  traceback            T or F*
  update               T or F [not windowed versions]
  vecread              "byfields" or "notbyfields"*
  warnings             T* or F
  wformat              quoted string ("16.9g"*)
  width                integer >= 30 (screen size*)
For details on an option, type help(options:Option) where Option is a
quoted option name.  Example: help(options:"nsig").
```

Keywords: control, missing values, output, random numbers, general

```
                          Introduction
MacAnova has a variety of options you may set. These control or affect
```

output formatting, "dumb" graph size, the units used by trigonometric functions, the string printed for MISSING values, the command line prompt, macro creation, while loops, random number generation and the action of save(), asciisave() and restore() (not an exhaustive list).

It may be helpful to think of options as hidden variables with LOGICAL, CHARACTER or REAL values. You use setoptions() to change their values and getoptions() to retrieve their values.

You can get a brief list of options names with permissible values by typing 'usage(options)'.

You can get an description of an individual option by typing help(options:optionName), where optionName is the quoted name of the option. For example, help(options:"format") gives information on option 'format'.

This topic lists all option that may be set. Permissible values for each option are in parentheses after the option name, usually with the default value indicated. A quoted string ("radians", for example) can be replaced by a CHARACTER scalar and T or F can be replaced by a LOGICAL scalar.

Options that may be set

Option 'angles'

angles ("radians", "degrees" or "cycles", default = "radians")
Example: setoptions(angles:"cycles")

The value specifies the angular units assumed for sin(), cos(), tan(), asin(), acos(), atan(), cpolar(), hpolar(), crect(), hrect() and unwind(). The default value is "radians". When 'angles' is "degrees", 360 is equivalent to 2*PI radians; when it is "cycles", 1 is equivalent to 2*PI radians. See also topic 'transformations'.

Option 'batchecho'

batchecho (T or F, default = T) Example: setoptions(batchecho:F)

The value determines whether command lines that are read from a file by batch() are echoed to output. True means echo; False means don't echo. It is ignored when you use keyword 'echo' on batch(). See batch().

Option 'dumbplot'

dumbplot (T or F, default = F) Example: setoptions(dumbplot:F)

The value affects the default behavior of all plotting commands (plot(), chplot(), lineplot(), addpoints(), boxplot(), ...). When it is True, these commands make "dumb" plots, that is plots using only characters such as could be produced on a printer with no graphics capability. When 'dumbplot' is False, high resolution graphs are drawn. It is ignored when you use keyword 'dumb' on a plotting command. In non-interactive mode, 'dumbplot' is always True.

When you use spool() to save your output to disk and 'dumbplot' is True, copies of all your plots will be spooled.

```
                              Option 'errors'
errors (integer >= 0)              Example: setoptions(errors:20)
  The value n is the maximum number of errors tolerated.  n = 0 means
  errors will not be counted.

  When setoptions(errors:n) is executed from a batch file, n is used
  only until entry from the keyboard is next needed.  If more than n
  errors are found while a batch file is being read, reading from all
  current batch files is terminated.

  In interactive mode, 'errors' is initially set to 0 (ignore errors).
  If it is not reset, use of batch() temporarily sets the error limit to
  1; it reverts to 0 when commands from the batch file are completed.

                            Option 'findmacros'
findmacros ("yes", "silent" or "no", default = "yes")
                                   Example: setoptions(findmacros:"no")
  The value controls the behavior of MacAnova when it encounters what
  looks like a call to a macro, but no macro with the name is defined.
  When 'findmacros' is "yes" or "silent", MacAnova searches all the
  files in pre-defined CHARACTER variable MACROFILES (see getmacros()
  and addmacrofile()) for the macro.  If a macro with that name is
  found, it is read in and then executed.  When 'findmacros' is "no", no
  search is made, resulting in an error.  When 'findmacros' is "yes", a
  warning message is printed before the search, and another one if the
  search is unsuccessful.  When 'findmacros' is "silent" or "no", no
  special messages are printed.

                              Option 'font'
font (quoted string)               Example: setoptions(font:"Courier")
  This option is only available on Mac OS 9.  The value is the name of
  the font used in the active command/output window and any additional
  windows that you may open from this window.  The value should be the
  name of an available font such as "Courier" or "Monaco".  You can set
  both options 'font' and 'fontsize' by, for example,
  setoptions(font:"Courier 12") (see 'fontsize' below).  The default
  value of 'font' is "McAOVMonaco".

  setoptions(font:"default") restores the default font and size.  See
  options 'fontsize' below.

  When you change the font, you will almost certainly want to choose a
  non-proportional font such as "Courier" or "Monaco", that is, a font
  for which all characters, including spaces, have the same width.  If
  not, columns will not line up and, in general, output will be hard to
  read.

  If you regularly prefer a font other than the default font, you might
  want to set this option in your startup file.  See topic 'customize'.

                             Option 'fontsize'
fontsize (integer > 0)             Example: setoptions(fontsize:18)
  The value is the size of the font used in the command/output window
```

and any additional windows you may open from this window. Option 'fontsize' is available only in Mac OS 9 versions, for which the default value is 9.

Option 'format'

format (quoted string, default "12.5g")
Example: setoptions(format:"13.6f")

The value specifies the default format for printing. For example, if the value of 'format' is "12.5g", most output will be in floating point form with 5 significant digits and a maximum width of 12 characters.

The value must be of the form "w.dg" (floating point with d significant digits and width w characters, including sign and exponent), or "w.df" (fixed point format with d digits after the decimal and minimum width w). If w is omitted (for example, ".5g") it is taken to be d+7. Examples are "10.5f" (fixed point with 5 decimal places and total width of 10) and ".15g" (floating point with 15 significant digits and width 22). See print() for more discussion of format specification.

When setting 'format', you can put the format type specifier ('f' or 'g') at the start of format. For example, setoptions(format:"f10.6") is equivalent to setoptions(format:"10.6f"). When you retrieve the value using getoptions(format:T), 'f' or 'g' is always at the end.

If you omit the width w (setoptions(format:".10g")), w = d+7 is assumed ("17.10g"). If you try to set the format so that w > 27 or d > 20, 27 and 20 are used, respectively.

The 'g' format is a hybrid of integer format, decimal format with no exponent, and exponential format. There is no purely exponential format available. See topic 'number'.

Option 'fstats'

fstats (T or F, default F) Example: setoptions(fstats:T)

The value affects the default behavior of anova(), robust() and fastanova(). F-statistics are printed only when the value is True. The value of option 'fstats' is ignored when keyword 'fstats' is used on these commands.

Option 'fstats' affects manova() only when 'byvar:T' is an argument.

Option 'height'

height (integer >= 0, default depends on screen size)
Example: setoptions(height:20)

The value is the assumed number of lines of output that will fit on the screen or in the command/output window.

If nLines is the value of 'height', under Unix/Linux and DOS, whenever nLines-1 lines of output printed by a single command line, MacAnova will print "Hit RETURN to continue or q RETURN to go to next command line:" and then pause. If command line editing is available, the

```
  message is "Press 'q' to quit, 'j' or 'n' to see next line, any other
  key to continue".

  The value of 'height' also affects the default size of stemleaf
  displays and "dumb" plots.

  When the value of 'height' is 0, counting of output lines is
  suppressed and the default size of stemleaf displays and "dumb" plots
  is 24.

  On Mac OS 9, the value of 'height' may be reset when the output
  window is resized; the only effect of the value is on the number of
  lines in a "dumb" plot.

  The value of 'height' can be predefined using the -l command line
  option.  See topic 'launching'.

  The value of option 'height' is ignored when keyword 'height' is used
  on print(), write(), matprint(), matwrite() and error() or on plotting
  functions such as plot() and boxplot() (see 'graph_keys').

  See topics stemleaf(), 'graphs'.

                            Option 'history'
history (integer >= 0)           Example: setoptions(history:200)
  The value is the number of command lines that are saved and may be
  retrieved.  The default value for history is 100.

  When the value of 'history' is 0, no command lines are saved.

                             Option 'inline'
inline (T or F, default = F)     Example: setoptions(inline:T)
  The value specifies the default method of macro expansion, in-line
  when the value is True or out-of-line when it is False.  The value of
  option 'inline' is ignored when you use keyword 'inline' on macro() or
  when you use macroread() to read a macro with 'INLINE' or 'OUTLINE' on
  its header line.  See topics macro(), macroread(), 'macros',
  macro_files.

                            Option 'keyboard'
keyboard  (integer >= 0)         Example: setoptions(keyboard:1)
  This currently has an effect only in the Windows version when using a
  non-US keyboard with an Alt Gr key.  The value determines whether
  codes generated by key combinations involving the Alt Gr key are
  recognized.  When keyboard = 2 (the default), any such key
  combinations are recognized.  When keyboard = 1, only key combinations
  associated with ASCII codes between 32 (space) and 126 (~) are
  recognized.  When keyboard = 0, no Alt Gr key combinations are
  recognized.

  Note: This value of 'keyboard' should not affect US keyboard
  configurations.  The option is provided to allow modification of
  recognition of Alt Gr combinations in case there are problems with the
```

```
  default behavior.  Other values may have special meaning in the
  future.

                            Option 'labelabove'
labelabove (T or F, default = F)   Example: setoptions(labelabove:T)
  This option specifies the placement of default coordinate labels when
  printing variables without labels (see topic 'labels').

  When the value is True, the values for the last subscript (only
  subscript in the case of vectors) are printed above the data instead
  of all the subscripts being printed in the left margin.  This has no
  effect on the printed labels of CHARACTER varables which are always at
  the left.  The effect is the same as if the variable had labels which
  were all of the form rep("@",dimsize).  The default value is False.

                            Option 'labelstyle'
labelstyle ("(", "[", "{", "<", "/", "\\", default = "(")
                                   Example: setoptions(labelstyle:"[")
  This option specifies the style used in printing coordinate labels for
  unlabelled variables (see topic 'labels').  For example, when its
  value is "[", '[' and ']' are used instead of the default '(' and ')'.

                              Option 'lines'
lines (integer >= 0, default depends on screen size)
                                   Example: setoptions(lines:20)
  Option 'lines' is the same as option 'height' (see above).  It is
  retained for compatibility with earlier versions.

                            Option 'matchdelay'
matchdelay (integer >= 0, default 20) (Macintosh OS 9 only)
                                   Example: setoptions(matchdelay:30)
  This sets the amount of time a matching left parenthesis or bracket is
  highlighted when you type a right parenthsis or bracket.  A value of 0
  turns off such highlighting.  The unit is ticks (1/60 second).

                            Option 'maxlinelen'
maxlinelen (integer >= 80, default 2000)
                                   Example: setoptions(maxlinelen:5000)
  This provides an upper limit on the length of lines from a file that
  can be completely read by vecread(), read(), matread() and
  macroread().  It also applies when reading from a CHARACTER variable
  using keyword 'string'.  If a longer line is found, you are warned
  that not everything in the line was read.

                             Option 'maxwhile'
maxwhile (integer >= 10, default 1000)
                                   Example: setoptions(maxwhile:50000)
  The value is the default upper limit on the number of repetitions of a
  while(){...} loop.  When the limit is exceeded the loop is terminated
  with an error message.

                            Option 'minpvalue'
minpvalue (0 <= number <= .001, default .00000001)
```

```
                                    Example: setoptions(minpvalue:5e-6)
  The value is the default lower limits for P-values to be printed as
  computed by GLM commands such as regress(), anova() or logistic().
  Values below the limit are printed as "> 1e-8", for example.  When the
  value is 0, P-values are always printed as computed.

  If the value provided has more than 2 significant digits (2.54e-6, for
  example) it is rounded (to 2.5e-6, for example) and a warning message
  is printed.

  Changing option 'minpvalue' doesn't affect values computed by cumxxx()
  functions (cumF(17.975,5,45,upper:T), for example) or by macro
  twotailt().

                               Option 'missing'
missing (quoted string with length <= 20 characters, default "MISSING")
                                    Example: setoptions(missing:"NA")
  The value is the default string used to print MISSING REAL or LOGICAL
  values.  For example, after setoptions(missing:"NA") any MISSING
  values will be printed as "NA" instead of the usual "MISSING".  The
  value of option 'missing' is ignored when you use keyword 'missing' on
  print() or write().  See print(), write().

  Changing option 'missing' does not affect the internal representation
  of MISSING or what gets written by matprint() and matwrite().

                                Option 'nsig'
nsig (0 < integer <= 20, default = 5)
                                    Example: setoptions(nsig:10)
  The value is the maximum number of significant digits or decimals to
  be used in most numerical output.  Its value is linked to that of
  option 'format'.  If the value of format is "w.dg" or "w.df", then the
  value of nsig is d.  setoptions(nsig:d) is equivalent to
  'setoptions(format:"w.dg")' where d is a positive integer and w = d+7.
  See option 'format' above.

                               Option 'prompt'
prompt (quoted string of no more than 20 characters, default = "Cmd> ")
                                    Example: setoptions(prompt:">> ")
  The value is printed at the start of each command line.  For example,
  after setoptions(prompt:"Next? "), each line will start with "Next? "
  instead of "Cmd> ".  In all versions except Mac OS 9, the initial
  value of 'prompt' can be set by command line flag -prompt (see
  'launching').

  When setoptions(prompt:Prompt) is executed in a batch file (see
  batch()), the new prompt remains in effect only until the commands in
  the file are finished.  Since a startup file (see 'customize') is
  executed as a batch file, this option cannot be usefully set in a
  startup file since the prompt is forgotten when the batch file is
  completed.

                                Option 'pvals'
```

```
pvals (T or F, default = F)      Example: setoptions(pvals:T)
  The value affects the default behavior of all the GLM commands except
  for robust() and screen().  It affects manova() only when keywords
  'byvar' or 'fstats' are used.  When the value is True, P values for
  t-, F- and chi-squared statistics are printed; when it is False, P
  values are not printed.  The value of option 'pvals' is ignored when
  you use keyword 'pvals' as an argument to a GLM command.

                              Option 'quiet'
quiet (T or F, default = F)       Example: setoptions(quiet:T)
  When True, all output is suppressed except for error messages and the
  input prompt (Cmd> ).  Option 'quiet' differs from other options in
  that its value is associated with a the macro currently running.

  A macro inherits the value of quiet from the prompt level, if called
  there, or from the value in a macro that calls it.

  When 'quiet' is set by setoptions() in a macro, it reverts to the
  former value when leaving the macro.

                            Option 'restoredel'
restoredel (T or F, default = T)  Example: setoptions(restoredel:F)
  The value determines whether existing variables will be deleted by
  restore().  When the value is True, they will be deleted; when it is
  False they will be deleted only when they are overwritten.  The value
  of 'restoredel' is ignored when you use keyword 'delete' on restore().

                            Option 'savehistry'
savehistry (T or F)   (note spelling)
                                  Example: setoptions(savehistry:F)
  The value determines whether a history of recent command lines will be
  saved by save() and asciisave().  When the value is True such a
  history is saved and will be automatically restored by restore(); when
  the value is False, the history is not saved.  Option 'savehistry' is
  ignored when keyword 'history' is used on save() and asciisave().  The
  default value of 'savehistry' is True except in non-interactive mode.

  'savehistry' is not available on versions such as the limited memory
  DOS version that do not maintain such a history.

                            Option 'scrollback'
scrollback (T or F, default = F) Example: setoptions(scrollback:T)
  The value determines when the command/output window will be
  automatically scrolled back to the beginning of output that is longer
  than the window can hold.  When the value is True, the window is
  scrolled back so that the most recent command line is visible.  When
  the value is False, no such scrolling takes place.  On help(), this
  default behavior can be overridden by the help() keyword 'scrollback'.
  Option 'scrollback' is available only in windowed versions.

                              Option 'seeds'
seeds (vector of 2 integers > 0 or both 0, default = vector(0,0))
                   Example: setoptions(seeds:vector(6542821,6228765))
```

```
  The value is a REAL vector of length 2, say vector(n1,n2), where n1 >=
  0 and n2 >= 0 are integers <= 2147483399.  These values are used and
  updated by random number generators runi(), rnorm(), rbin() and
  rpoi().

  You normally set this option by setseeds() since setseeds(n1,n2) is
  equivalent to setoptions(seeds:vector(n1,n2)).

  When the value is vector(0,0), the first use of runi(), rnorm(),
  rpoi() or rbin() causes the seeds to be set to pseudo-random values
  determined from the time of day.

                              Option 'tekset'
tekset (CHARACTER vector of length 2)
       Example: setoptions(tekset:vector("\033[?38h\0338","\033\003"))
  The value is a CHARACTER vector of length 2, say vector(ToTek,
  FromTek).  ToTek is the CHARACTER string that switches the terminal
  emulator you are using to Tektronix 4014 mode and FromTek is the
  CHARACTER string that switches out of Tektronix 4014 mode.  These are
  used before starting and after finishing a plot, and are used by
  Mouse().

  When ToTek is "", no code is sent to switch to Tektronix mode.
  Similarly, when FromTek is "", no code is sent to switch back.

  When MacAnova is running under Xterm on a Unix/Linux workstation, the
  default is as in the example above, which could also be set by
   Cmd> setoptions(tekset:vector(putascii(27,91,63,51,56,104,27,56,\
      keep:T), putascii(27,3,keep:T)))

  When MacAnova is not running under Xterm, the default is
  vector("\035\0338", "\0332") which is equivalent to
  vector(putascii(29, 27, 56, keep:T), putascii(27, 50, keep:T))

  Both defaults initialize the character size to large.

  The non-Xterm default works for Macintosh program Versaterm.  For
  other terminal emulators, this option should be initialized in a
  startup file.  See topic 'customize'.  For example, for public domain
  program Kermit 3.0 for Windows/DOS computers, your startup file might
  contain the command (not tested)
    setoptions(tekset:vector("\33[?38h\0338","\033[?38h"))

  For NCSA Telnet 2.6 for a Macintosh, use
    setoptions(tekset:vector("\033\014","\030"))

  Option 'tekset' is available only on Unix/Linux versions using
  Tetronix 4014 emulation for high resolution graphs.

                             Option 'traceback'
traceback (T or F)                Example: setoptions(traceback:F)
  The value controls whether the chain of calling macros will be printed
  when an error occurs in a macro called by another macro.  It has no
```

```
  effect when an error does not occur in a macro.  When an error occurs
  in a macro and the value is F (the default), only the name macro is
  printed.  When the value is true the name of the macro and, when that
  macro was called in a macro, the chain of calling macros is printed.

  This option makes use of the fact that expanded macros are prefaced by
  the header '{#)#macroname'. Because some error messages append about
  50 characters of input preceding the error, you may see some calling
  macro names in short macros.  Here is an example of what this means:

  Cmd> a <- macro("q + PI"); b <- macro("a()"); c <- macro("b()")

  Cmd> setoptions(traceback:F); c() # no traceback (q not defined)
  ERROR: arithmetic with undefined operand
  UNDEF + REAL in macro a near setoptions(traceback:F); {#)#c
  {#)#b
  {#)#a
  q + PI

  Cmd> setoptions(traceback:T); c() # traceback
  ERROR: arithmetic with undefined operand
  UNDEF + REAL near setoptions(traceback:T); {#)#c
  {#)#b
  {#)#a
  q + PI
   in macro a
   called by macro b
   called by macro c

  There may be a few error messages unaffected by the value of
  'traceback'.

  See topic macro_syntax for information about writing macros.

                              Option 'update'
update (T or F)                   Example: setoptions(update:F)
  The value controls whether the screen will be updated (previous
  commands and output re-printed) after a high resolution plot (True
  means update; False means don't update).  The default value of update
  is True in DOS versions and False in non-windowed Unix/Linux versions.

  Option 'update' is not available in a Windowed version.

                             Option 'vecread'
vecread ("byfields" or "notbyfields", default = "notbyfields")
                                  Example: setoptions(vecread:"byfields")
  The value specifies the default mode for vecread() when reading REAL
  data.  When it is "notbyfields", the default value for vecread()
  keyword 'byfields' is False; when it is "byfields" the default value
  of 'byfields' is True.

  Option 'vecread' is ignored by vecread() when it is reading CHARACTER
  data.
```

```
                          Option 'warnings'
warnings (T or F, default = T)   Example: setoptions(warnings:F)
  The value controls whether MacAnova will suppress the printing of
  warning messages (those that start with "WARNING:").  Such lines will
  be printed only when the value of 'warnings' is True.  Setting
  'warnings' to False can be useful when doing arithmetic with and
  transformations of vectors or matrices with missing data.  However,
  since many warning messages are quite important, it should be used
  with caution.  When you change 'warnings' to False, you should use
  setoptions(warnings:T) to restore the usual behavior as soon as
  possible.

                          Option 'wformat'
wformat (quoted string, default = "16.9g")
                                Example: setoptions(wformat:".17g")
  The value specifies the default format for the commands write() and
  matwrite().  For example, if the value is "16.9g", most output will be
  in floating point form with 9 significant digits and a maximum width
  of 16 characters.

  See option 'format' above for more information about permissible
  values.

                           Option 'width'
width (integer >= 30, default depends on screen width)
                                Example: setoptions(width:65)
  The value is the number of characters assumed to fit on a line on the
  screen or in the window.  This number, together with the current
  formatting options, determines how many items are printed per line and
  the width of "dumb" plots.

  The value of option 'width' is ignored when keyword 'width' is used on
  print(), write(), matprint(), matwrite() and error() and on plotting
  functions such as plot() and boxplot() (see 'graph_keys').

  On Unix/Linux and DOS 'width' may be initialized by the -w command
  line flag (see topic 'launching').  On Mac OS 9, 'width' may be
  reset when the output window is resized.

  Some output does not respect this limit.
```

2.260 outer()

Usage:

```
outer(x1, x2, ...), x1, x2, ... REAL
```

Keywords: matrix algebra

```
                                Usage
```

```
outer(x1, x2) returns a matrix or array which is the "outer product" of
REAL variables x1 and x2.  The dimensions of the result are the
joined dimensions of the arguments.

After result <- outer(x1,x2), the elements of the result are
  result[i,j,k,...,l,m,n,...] = x1[i,j,k,...] * x2[l,m,n,...].

outer(x1, x2) is equivalent to array(vector(x1)*vector(x2)',
vector(dim(x1), dim(x2))).

outer(x1, x2, x3) is mathematically equivalent to outer(outer(x1,x2),
x3) and in general outer(x1, x2, x3, ..., xk) is mathematically
equivalent to outer(outer(...(outer(x1,x2), x3), ...), xk).  The
elements of the result are all possible k-way products of elements from
each of the arguments.

                      Multi dimensional contrasts
One use for outer() is to construct multidimensional contrasts that are
products of 1 dimensional contrasts.  Suppose c1, c2 and c3 are vectors
of main effect contrast coefficients for each factor for a 3 factor
design.  Then after anova("y=a*b*c"),

  Cmd> contrast("a.b.c", outer(c1,c2,c3))

computes results for the three way product contrast that is part of the
a.b.c interaction.

                           Cross references
See also topics contrast(), array(), 'matrices'
```

2.261 padto()

Usage:

```
padto(x,n), x a REAL vector or matrix, n > 0 an integer
```

Keywords: time series

```
                           Usage and example
padto(x,n) creates a new matrix or vector from REAL vector or matrix x
by adding n - nrows(x) rows of all zeros so as to bring the total number
of rows to n.  When n < nrows(x), the last nrows(x) - n rows of x are
deleted to bring the number down to n.  n must be a positive integer.

The principal use of padto() is to add zeros to a time series after
subtracting the mean or other estimate of trend but before computing its
Fourier transform, as in rft(padto(x-sum(x)/nrows(x),S)), where S is the
number of frequencies desired.  If x has several columns, they all get
padded simultaneously.

Example:
```

```
Cmd> padto(vector(1,2,2)',4) # pad row vector to matrix with 4 rows
(1,1)           1           2           2
(2,1)           0           0           0
(3,1)           0           0           0
(4,1)           0           0           0
```

2.262 partacf()

Usage:

```
partacf(rho), rho a REAL matrix whose columns are autocorrelations
partacf(phikk, inverse:T), phikk a REAL matrix whose columns are
  partical autocorrelations
```

Keywords: time series

```
                                  Usage
partacf(rho), where rho is a REAL vector, computes the partial
autocorrelations corresponding to the autocorrelation function in the
REAL vector rho.  Row k of rho should contain the lag k autocorrelation.
The Levinson-Durbin algorithm is used.

If rho is a matrix, partacf(rho) is a matrix of the same shape whose
j-th column contains partial autocorrelations corresponding to
autocorrelations in column j of rho.

If any column of rho is not a valid autocorrelation function, that is,
it does not define a positive definite Toeplitz matrix, a warning
message is printed.

partacf(phikk,inverse:T) is the inverse function to partacf.  Each
column of REAL vector or matrix phikk is considered to be the partial
autocorrelation function of a time series.  The corresponding column of
the result is the corresponding autocorrelation function.  All the
elements of phikk must be less than 1 in absolute value.

                           Cross references
See also yulewalker().
```

2.263 paste()

Usage:

```
paste(arg1, arg2, ... [,format:Fmt,sep:C,intwidth:Iw,charwidth:Cw,\
  missing:S]), Fmt, C, and S CHARACTER scalars, Iw and Cw integers > 0
paste(arg,multiline:T [,format:Fmt,sep:Cs,linesep:Cl,missing:S]), where
  Cs and Cl are CHARACTER scalars consisting of a single character.
'iw', 'cw', 'fmt', and 'just' are synonyms for 'intwidth', 'charwidth',
  'format' and 'justify'
```

Keywords: output, missing values

Usage

paste(arg1, arg2, ...) returns a CHARACTER scalar concatenating the arguments. For example, the value of

```
    paste("The answer is", run(7), "; ok?")
```

is the string "The answer is 1 2 3 4 5 6 7 ; ok?" .

An important use of paste() is in constructing labels for graphs (for example, title:paste("Variable",j,"vs variable",i)). It is also useful for producing informative messages to be printed in a macro and can be used to prepare nicely formatted lines for output.

The default behavior is to print the arguments separated by single spaces, with exact integers printed as such, non-integers printed using the default print format (see print(), subtopi 'options:"format"') with leading spaces trimmed off, and missing values printed as "MISSING". If you have used setoptions() to replace "MISSING" by a different default string, the replacement will be used.

A NULL non-keyword argument is ignored unless it is the only argument, in which case paste(NULL) returns "".

Keyword 'sep'

paste(arg1, arg2, ..., sep:S), where S is a CHARACTER variable or quoted string, uses S to separate arguments rather than a space. For example, paste(run(5),sep:",") produces the string "1,2,3,4,5" and paste("A","B","C","D","E",sep:"") produces the string "ABCDE". No separator is ever put before the first item in the output variable. You can have several instances of 'sep:S' with different separators, each affecting later arguments until changed.

Keyword 'intwidth'

paste(arg1, arg2, ..., intwidth:w), where w is a positive integer prints each exact integer using at least w positions, padding on the left with spaces if necessary. For example, paste(12,intwidth:5) produces " 12". You can use iw:w instead of intwidth:w.

Keyword 'format'

paste(arg1, arg2, ..., format:Fmt), where Fmt is a CHARACTER variable or quoted string representing either a f-format ("10.5f" or "f10.5") or g-format ("11.7g" or "g11.7") (see print()) prints any non-integer REALs using format Fmt If the width is omitted (".7f") leading spaces will be stripped off. If the width is not omitted ("10.7f"), any non-integer REAL variables printed will be formatted so as to use at least this width. If 'intwidth:w' has not previously appeared, this width will also be used for integers. You can use fmt:Fmt instead of format:Fmt.

Keyword 'charwidth'

paste(arg1, arg2, ..., charwidth:w), w is a positive integer, uses at least w character positions for any CHARACTER argument, padding on the right with spaces if necessary, unless justify:"r" or justify:"c" is an argument. You can use cw:w instead of charwidth:w.

Keyword 'justify'

paste(arg1, arg2, ..., justify:C), where C is "right", "left", or "center" (or simply "r", "l", or "c"), specifies that any strings are to be right justified, left justified or centered, respectively. This has no effect unless charwidth:w is specified and a string is shorter than w. The default is justify:"left". You can use just:C instead of justify:C.

Keyword 'missing'

paste(arg1, arg2, ..., missing:String), where String is a quoted string or CHARACTER scalar such as "?", uses String to represent a missing value instead of "MISSING". If a format has been specified with width > 0 longer than the length of String, String will be padded on the left to make it have this width.

Keywords used more than once

You can use any of the keywords more than once anywhere in the argument list, with each usage affecting the formatting of subsequent items until changed by a new keyword phrase. Putting it after all non-keyword arguments, as illustrated above, is equivalent to putting it before them. For example, paste(sep:",",charwidth:5,a,b,c) is equivalent to paste(a,b,c,sep:",", charwidth:5).

Keyword 'multiline'

paste(arg,multiline:T) has somewhat different behavior. The result is much as before, except, if arg is other than a row vector, the result is a CHARACTER vector, with one element for each value of the first dimension that is greater than 1. Thus, if arg is a matrix or vector, each element of the output is a character representation of a row of arg. When you use 'multiline:T', there must be exactly 1 non-keyword argument. If arg is LOGICAL, it is first translated to REAL with True and False becoming 1 and 0, respectively. If arg is NULL, the result is "".

paste(arg,multiline:T,linesep:Char), where Char is a string with just one character such as ";" or "\n" (the end-of-line character), combines the rows into a single CHARACTER scalar with each row separated by Char. If arg is a row vector (just 1 row), Char is ignored.

Along with 'multiline:T', you can use keywords 'sep', 'format', and 'missing', but not 'intwidth' and 'charwidth'. If it appears, the value for 'sep' must be a string with just one character, for example, " " (the default) or "\t". Any leading or trailing blanks in each numerical field are trimmed off when 'sep' is used.

paste() does not recognize keyword 'nsig'.

Examples without 'multiline:T'

```
Cmd> paste(sep:"-","tick","tock",sep:", ","ding",sep:"-","dong")
(1) "tick-tock, ding-dong"
```

The first 'sep:"-"' could also come at the end.

```
Cmd> paste("PI is",PI,format:".10f") # or "Pi is",PI,fmt:".10f"
```

```
  (1) "PI is 3.1415926536"

  Cmd> paste(format:"10.2f",sqrt(2),format:".15g",sqrt(2))
  (1) "      1.41 1.4142135623731"

  Cmd> paste("Blocks",DF[2],SS[2],SS[2]/DF[2],format:"7.3f",\
        (SS[2]/DF[2])/mse,charwidth:8,format:"13.6g",intwidth:2)
  (1) "Blocks    4       48.0368       12.0092  18.782"

In the preceding, 'cw', 'iw', and 'fmt' could replace 'charwidth',
'intwidth' and 'format'.

                       Examples with 'multiline:T'
  Cmd> paste(x, multiline:T) # 3 by 5 matrix x
  (1) "10.322 9.5278 10.636 10.411 9.6343"
  (2) "9.9979 10.606 8.1604 MISSING 8.5926"
  (3) "8.6147 11.212 9.4683 7.7964 10.489"

  Cmd> paste(x,multiline:T,linesep:"\n",sep:",",\
        missing:"-99",format:"0.4f")
  (1) "10.3222,9.5278,10.6357,10.4106,9.6343
  9.9979,10.6057,8.1604,-99,8.5926
  8.6147,11.2120,9.4683,7.7964,10.4889"

                            Cross references
See also print().
```

2.264 plot()

Usage:

```
   plot(x,y [,add:T,impulses:T, lines:T] [other graphics keyword phrases]),
     where x is a REAL vector or scalar, y is a REAL vector or matrix
   plot([Graph,] [x,y], keys:str), str a structure whose component names
     are graphics keywords
```

Keywords: plotting

```
                                    Usage
plot(x,y) makes a scatter plot of the data in vector x and vector or
matrix y using characters such as asterisks or diamonds as plotting
symbols.  If y has several columns, they are plotted with symbols
asterisk, diamond, cross, square, X, triangle, asterisk, dot, small
cross, diamond,..., thereafter cycling through the plotting symbols.

It is not an error when x or y is NULL; a warning message is printed and
no plotting occurs.

plot(x,y, symbols:c), where c is a CHARACTER or integer scalar, vector,
or matrix with ncols(c) == ncols(y), uses the elements of c as plotting
symbols as for chplot().  In particular, if c is a CHARACTER scalar
other than "###", it is used as a plotting symbol for all points (at
```

most first three characters).

plot(x,y,symbols:"###") labels each point with the row number when y is a vector and with the column number when ncols(y) > 1.

plot(x,y,impulses:T [,symbols:c]) makes an "impulse" plot of y vs x, drawing vertical lines from the x = 0 line to each point. If y has several columns, a different line type is used for each column. However, since the lines will probably be superimposed, it may be hard to interpret the resulting plot.

plot(x,y, lines:T [,impulses:T, symbols:c) does the same except the points or impulses will be connected by lines similarly to lineplot(). You can also use keywords 'linetype' and 'thickness'. See topic 'graph_keys'.

Structure first argument

plot(Str [,impulses:T]), where Str is a structure with at least two REAL components, is equivalent to plot(Str[1], Str[2] [,impulses:T]). For example, plot(x,y) and plot(structure(x,y)) are equivalent. Any components of Str beyond the first two are ignored.

LASTPLOT and GRAPHWINDOWS

plot() normally creates or replaces GRAPH variable LASTPLOT which encapsulates everything in the graph. In addition, if the graph was drawn in graphics window I, GRAPHWINDOWS[I] is made identical to LASTPLOT (I is always 1 in non-windowed DOS and Unix/Linux versions). Saving the plot information in LASTPLOT and GRAPHWINDOWS[I] can be suppressed by including 'keep:F' as an argument. See topics 'graphs' and 'graph_assign' for information on GRAPH variables and special variable GRAPHWINDOWS.

Graph variable first argument

plot(graph,x,y [,impulses:T]) or plot(graph,Str [,impulses:T]), where graph is a GRAPH variable, draws the plot encapsulated in graph, adding to it the new information. See topic 'graphs' for details on adding information to a plot.

Keyword 'add'

plot(x,y [,impulses:T],add:T,...) is the same as plot(LASTPLOT,x,y [,impulses:T],...) drawing the graph encapsulated in LASTPLOT, adding to it new information. An equivalent way to do this is addpoints(x,y [,impulses:T],...).

Low resolution plot

If option 'dumbplot' has been set False (see subtopi 'options:"dumbplot"'), the plot will be a low resolution plot unless 'dumb:F' is an argument.

Short vector for x

See topic 'graphs' for information on how a scalar or length 2 vector x specifies equally spaced x-values, on how to save and print plots, and on writing graphic information to a file.

```
                          Graphics keywords
Keywords 'dumb', 'lines', 'linetype', 'thickness', 'impulse', 'xmin',
'xmax', 'ymin', 'ymax', 'logx', 'logy', 'xlab', 'ylab', 'title',
'xaxis', 'yaxis', 'borders', 'ticks', 'xticks', 'yticks', 'xticklen',
'yticklen', 'xticklabs', 'yticklabs', 'height', 'width', 'pause',
'silent' and 'notes' may be used as for other plotting commands.  See
topics 'graph_keys', 'graph_border' and 'graph_keys'

plot([Graph,] keys:structure(x:x,y:y [other keyword phrases)) is
equivalent to plot([Graph,] x:x,y:y [other keyword phrases]).  See topic
'graph_keys' for details.

See topic 'graph_assign' for information on how to plot in graphics
window I by GRAPHWINDOWS[I] <- var, where var is a structure or GRAPH
variables.

                               Examples
Examples:
  Cmd> plot(yhat1:yhat[,1],resid1:RESIDUALS[,1],\
        title:"Residuals vs yhat")

  Cmd> plot(X:1,run(20)^(.2*run(5)'),ylab:"Powers of X",\
        title:"X^.2, X^.4, X^.6, X^.8, and X", file:"ps.out",new:T)

  Cmd> plot(X:1,run(20)^(.2*run(5)'),ylab:"Powers of X",\
        title:"X^.2, X^.4, X^.6, X^.8, and X", logy:T)

                          Cross references
See also topics chplot(), lineplot(), addpoints(), addlines(),
addchars(), showplot(), colplot(), rowplot(), tek(), tekx(), vt(),
vtx().
```

2.265 poisson()

Usage:

```
   poisson([Model] [, print:F or silent:T, incr:T, offsets:vec, pvals:T,\
     maxiter:m, epsilon:eps, coefs:F]), vec a REAL vector, m an integer >
     0, eps REAL > 0
```

Keywords: glm, regression, categorical data

```
                               Usage
poisson(Model) computes a log linear regression fit of the model
specified in the CHARACTER variable Model.  If y is the response
variable in the model it must be a REAL vector with y[i] >= 0.
Estimation is by maximum likelihood on the assumption that y[i] is
Poisson.  If any y[i] is not an integer a warning message is printed.

See topic 'models' for information on specifying Model.
```

poisson(Model,...) is equivalent to glmfit(Model,dist:"poisson", link:"log",...).

Side effect variables created

poisson() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS, DF, HII, DEPVNAME, TERMNAMES, and STRMODEL. The elements of WTDRESIDUALS are the final weighted residuals in the iteratively reweighted least squares fit to log(response). See topic 'glm'. Without keyword phrase 'inc:T' (see below), TERMNAMES has value vector("","", ...,"Overall model","ERROR1"), DF has value vector(0,0, ...,ModelDF,ErrorDF) and SS has value vector(0,0,...,ModelDeviance, ErrorDeviance).

Analysis of deviance

If, say, Model is "y=x1+x2", an iterative algorithm fits log(y) as a linear function of x1 and x2. A two line Analysis of Deviance table is printed, with line 1 the diffence between the deviance from a model with all coefficients 0 and the deviance of the estimated model, and line 2, labeled "ERROR", the deviance of the estimated model. Under appropriate assumptions, the latter can be used to test the goodness of fit of the model.

Incremental fitting

poisson(Model,inc:T) computes the full Poisson model and all partial models -- only a constant term, the constant and the first term, and so on. It prints an Analysis of Deviance table, with one line for each term, plus the deviance of the complete model labeled as "ERROR" . Each term's deviance is the reduction in deviance associated with that term.

Omitting model

If you omit Model (poisson()), the model from the most recent GLM command such as poisson() or anova(), or the model in CHARACTER variable STRMODEL is assumed.

Algorithm

Computations are carried out using iteratively reweighted least squares with starting values derived from an unweighted least squares fit of log(y + .25).

Other keyword phrases

Keyword phrase	Default	Meaning
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting
epsilon:eps	1e-6	Small positive REAL specifying relative error in objective function (2*log likelihood) required to end iteration
offsets:OffVec	none	Causes model to be fit to log(p) to be 1*Offvec + Model, where OffVec is a REAL vector the same length as response y. Note OffVec is in log units.

```
See topic 'glm_keys' for information on keyword phrases print:F,
silent:T, coefs:F and pvals:T.

The default value for pvals can be changed by setoptions(pvals:T).  See
topic setoptions(), subtopic 'options:"pvals"'.

                    Examples of the use of 'offsets'
  Cmd> poisson("y=x", offsets:3*x, inc:T, pvals:T)
The P value associated with x can be used to test the hypothesis H0:
beta1 = 3 in the model log(E[y]) = beta0 + beta1*x.

  Cmd> poisson("y=1", offsets:rep(log(10), length(y), inc:T, pvals:T)
The P value associated with the CONSTANT term can be used to test H0:
E[y] = 10, assuming y contains a random sample from a Poisson
distribution.
```

2.266 polygamma()

Usage:

```
polygamma(x [,n]), x REAL with positive elements or a structure with
  REAL components with positive elements, integer n >= 0
```

Keywords: transformations

```
                                  Usage
polygamma(x,0) and polygamma(x) both return the digamma function (first
derivative of log(gamma(x))) of the elements of x, when x is a REAL
scalar, vector, matrix or array with positive elements.  The result has
the same shape as x.  You can use digamma(x) instead.

polygamma(x, n), where n > 0 is an integer returns the n-th derivative
of the digamma function ((n+1)-th derivative of log(gamma(x))).

polygamma(x, n, scale:T) returns (-1)^(n+1)*n!*polygamma(x,n).

For n >= 1, polygamma(x, n, scale:T) = sum((x+k)^(-n-1),k=0,1,2,...,oo).
In particular, polygamma(1,n,scale:T) computes zeta(n+1), where zeta(s)
is the Riemann Zeta function.

                           Structure argument
When x is a structure, all of whose non-structure components are REAL
with positive elements, polygamma(x [,n] [,scale:T]) returns a structure
of the same shape and with the same component names as x with each
non-structure component transformed by polygamma().

                           CHARACTER argument
polygamma(x, n) can also be used when x is a CHARACTER variable and n,
if present, is a quoted string or CHARACTER scalar or REAL scalar.  The
result is a CHARACTER variable of the same shape as x describing the
transformation.  See example below.
```

```
Any element of x that is "" or starts with '@', '(', '[', '{', '<', '/'
or '\' is not modified.  This can be useful for creating labels for a
transformed variable.

                                Examples
Examples:
  Cmd> polygamma(run(10)) # or polygamma(run(10),0), or digamma(run(10))
  (1)    -0.57722     0.42278     0.92278      1.2561      1.5061
  (6)      1.7061      1.8728      2.0156      2.1406      2.2518

  Cmd> polygamma(run(1,2,.25),1) # trigamma
  (1)      1.6449      1.1973      0.9348      0.7641     0.64493

  Cmd> polygamma(vector("x","y"),3) # or polygamma(vector("x","y"),"3")
  (1) "polygamma(x,3)"
  (2) "polygamma(y,3)"

  Cmd> print(nsig:17,polygamma(1,23,scale:T),name:"zeta(24)")
  zeta(24):
  (1)       1.0000000596081891

                             Cross references
See also digamma(), lgamma(), 'transformations'.
```

2.267 polyroot()

Usage:

```
   polyroot(coefs), coefs a REAL matrix
```

Keywords: time series, complex arithmetic

```
                                   Usage
polyroot(Coef) computes the real and possibly complex roots of the
polynomials specified by the columns of REAL matrix Coef.  If c[i] is
Coef[i,j], then the polynomial whose roots are found is
  x^n - c[1]*x^(n-1) - c[2]*x^(n-2) - ... - c[n-1]*x - c[n],
where n = nrows(Coef).  Note that the leading coefficient (of x^n) is 1,
and the coeficients are associated with descending powers of x.

NOTE: The sign assumed for Coef is not affected by variables ARSIGN or
MASIGN which are recognized by several macros in file Arima.mac.  Type
arimahelp(MASIGN) for details.

If Coef is n by m, the result returned is a n by 2*m matrix with the
real and imaginary parts of the roots associated with column j of Coef
in columns 2*j-1 and 2*j, that is in the standard fully complex form.
See topic 'complex'.

To find the roots of polynomial d[1]*x^n+d[2]*x^(n-1)+...+d[n]*x+
```

```
d[n+1], use polyroot(-d[-1]/d[1]) (when d is a matrix use
polyroot(-d[-1,]/d[1,])).

To find the roots of polynomial d[1]+d[2]*x+d[3]*xˆ2...+d[n+1]*xˆn, use
polyroot(-reverse(d[-(n+1)])/d[n+1]) (when d is a matrix use
polyroot(-reverse(d[-(n+1),])/d[n+1,])).  See reverse().

                    Use with autoreg() and movavg()
The form of the argument to polyroot is adapted to its use in evaluating
autoregressive and moving average operators.  If phi is a REAL vector
and x is a vector of white noise, autoreg(phi,x) generates a stationary
autoregressive series if and only if the roots computed by polyroot(phi)
are inside the unit circle, that is, max(creal(cpolar(polyroot(phi)))) <
1.  Similarly, movavg(theta,x) generates an invertible moving average
model if and only if all the roots computed by polyroot(theta) lie
inside the unit circle.

                           Cross references
See also topics autoreg(), movavg(), 'complex' and subtopic
'matrices:"complex_matrices".
```

2.268 popmodel()

Usage:

```
popmodel([all:T])
popmodel(canpop:T)
```

Keywords: anova, glm, multivariate analysis, regression, residuals

```
                              Introduction
All GLM commands except screen(), for example regress() and anova(),
retain information (GLM information) internally for use by certain
functions such as coefs() and modelinfo() that provide results from the
most recent GLM command.  When a GLM command is run, information from
the previous GLM command, if any, is replaced.  When the GLM command is
in a macro, this may confuse the user who may expect that coefs(), say,
gives the same answer after the macro is used as before.

Commands pushmodel() and popmodel() allow a macro to save and restore
the current GLM information.

pushmodel() saves the current GLM information so that it can
subsequently be restored by popmodel().  Until a new GLM command has
been run or GLM information has been restored by popmodel(), there is no
GLM information available to commands such as secoefs() and modelinfo().

After the next prompt following the use of pushmodel(), the active model
before the first use of pushmodel() is restored (equivalent to an
automatic execution of popmodel(all:T)).

pushmodel() and popmodel() are intended to be used in macros.  A macro
```

```
can run pushmodel() before a GLM command and then run popmodel() before
finishing to ensure that the current GLM information is not changed.  Of
course, if the purpose of the macro is to change the information, it
should not use pushmodel() and popmodel().

The default maximum number of sets of GLM information that can be saved
is 2 (0 in limited memory DOS version).  On versions allowing command
line arguments you can change the default by '-savemodels N' where N >=
0 is an integer.  See 'launching'.

                                  Usage
popmodel() replaces the current GLM information by the GLM information
saved most recently by pushmodel().  It also deletes all GLM side effect
variables such as RESIDUALS and STRMODEL and then replaces them by side
effect variables appropriate to the model being restored.

popmodel(all:T) discards the current GLM information and all but the
first GLM information saved by previous use of pushmodel().

With either usage, when there is no information that has been saved by
pushmodel(), a warning message is printed and the current GLM
information is not discarded nor are side effect variables modified.

popmodel(canpop:T) returns True if there is saved GLM information that
could be restored by popmodel().  It does not change the current model
or side effect variables.

                                 Example
Type help(pushmodel:"example") for an example.

                             Cross references
See also pushmodel(), modelinfo(), coefs(), secoefs(), contrast(),
'macros'.
```

2.269 power()

Usage:

```
power(noncen,ngrp,nrep,alpha [,design:"rbd"]), noncen >= 0, 0 <  alpha
  < 1, integers ngrp > 0 and nrep > 0; some or all arguments may be
  vectors
```

Keywords: probabilities, glm, anova

```
                                  Usage
power(noncen,ngrp,nrep,alpha) computes the power of an F-test with
significance level alpha in a balanced one-way analysis of variance
(completely randomized design) for ngrp groups of size nrep (ngrp
treatments with nrep replications) with the n=1 noncentrality parameter
noncen.

The noncentrality parameter is noncen = sum(effects_i^2)/sigma^2 = the
```

```
sum of the squared treatment effects divided by the error variance.
This is sometimes called the "n=1 noncentrality parameter."  It differs
from the definition of the noncentrality parameter for power2() which
includes a factor of n.

power(noncen,ngrp,nrep,alpha,design:"rbd") computes power for a
randomized block design with nrep blocks and ngrp >= 2 treatments.

power(mu_aˆ2/sigmaˆ2,1,n,alpha) computes the power against the
alternative hypothesis H_a: mu = mu_a of a single-sample two-tail t-test
of H_0: mu = 0 based on a sample of size n.  To compute the power of a
one-tail t-test, see cumstu:"non_central_t".

Some or all of the arguments of power may be vectors, in which case all
non-scalars must be the same length, which will also be the length of
the result.  For example, you can compute the power of randomized block
designs with 2 to 20 blocks, g treatments and noncentrality parameter 2
by
  Cmd> power(2.5, g, run(2,20), .05, design:"rbd")

This is exactly equivalent to
  Cmd> power2(run(2,20)*2.5, g-1, (g-1)*(run(2,20) - 1), .05)

If nrep was computed as samplesize(noncen,ngrp,alpha,pwr), the value of
power(noncen,ngrp, nrep,alpha) should be approximately equal to pwr, but
no smaller.

                            Cross references
See also power2() and samplesize().
```

2.270 power2()

Usage:

```
power2(noncent2,numDF,denomDF,alpha), noncent2 >= 0, 0 <  alpha < 1,
  numDF >0, denomDF > 0; some or all arguments may be vectors
```

Keywords: probabilities, glm, anova, regression

```
                                  Usage
power2(noncen2,numDF,denomDF,alpha) computes the power for an F test
with numDF numerator degrees of freedom, denomDF denominator degrees of
freedom, a significance level of alpha, and noncentrality parameter
noncen2.

The noncentrality parameter noncen2 = sum(n_i*(effect_i)ˆ2)/sigmaˆ2,
where n_i and effect_i = mu_i - mu_all are the sample size and treatment
effect for group i, with mu_i = treatment i mean and mu_all =
sum(n_i*mu_i)/sum(n_i).

An more mathematical definition is
   noncen2 = numDF*(E[Numerator MS]/E[denominator MS] - 1);
```

```
Note that this differs from the n=1 non-centrality parameter expected by
power() which does not include a sample size.  For example, you will get
the same answers from
  Cmd> power2(nrep*noncen,ngrp-1,(nrep-1)*ngrp,alpha)
and
  Cmd> power(noncen,ngrp,nrep,alpha)

power2(n*mu_aˆ2/sigmaˆ2,1,n-1,alpha) computes the power against the
alternative hypothesis H_a: mu = mu_a of a single-sample two-tail t-test
of H_0: mu = 0 based on a sample of size n.  To compute the power of a
one-tail t-test, see cumstu:"non_central_t".

Some or all of the arguments of power2() may be vectors, in which case
all non-scalars must be the same length, which will also be the length
of the result.  For example, you can compute a power as a function of
noncen2 by, say
   Cmd> power2(run(0,100)*1.2,10,20,.05)

If nrep was computed as samplesize(noncen,ngrp,alpha,pwr), the value of
power2(nrep*noncen,ngrp-1,(nrep-1)*ngrp,alpha) should be approximately
equal to pwr, but no smaller.

power2() is useful for computing the power of a test for a contrast, or
for interaction and related effects where the error degrees of freedom
are complicated functions of n.

                            Cross references
See also power() and samplesize().
```

2.271 precedence

Usage:

```
This topic has information on the precedence and grouping properities of
arithmetic, matrix, logical, comparison and bit operations.
```

Keywords: syntax, operations

```
                              Introduction
Operators in MacAnova such as '+', '/', '<', '&&' and '<-' have rules of
association and precedence which determine the order in which they are
evaluated when used together.  As much as possible these mimic the rules
of ordinary algebra where they apply and for most purposes that is all
you need to know.  This topic summarizes the rules and includes a
precedence table for all operators.

                    Association properties of operators
A binary operator OP such as '+', 'ˆ' or '<=' either associates from
left to right, that is, x OP y OP z means (x OP y) OP z, or from right
to left, that is x OP y OP z means x OP (y OP z), or does not associate
at all, that is x OP y OP z is meaningless.
```

Binary arithmetic operators '+', '-', '*', '/', and '%%' associate from left to right. For example, x - y - z means (x - y) - z and x/y/z means (x/y)/z. See topic 'arithmetic'.

Exponentiation ('^' or '**') associates from right to left, that is x^y^z is x^(y^z), not (x^y)^z. See topic 'arithmetic'.

Binary logical operators '&&' and '||' associate from left to right. For example, u && v && w means ((u && v) && w). See topic 'logic'.

Matrix multiplication operators '%*%', '%c%' and '%C%' associate from left to right. For example, x %*% y %c% z %C% w is interpreted as ((x %*% y) %c% z) %C% w. See topic 'matrices'.

The assignment operator '<-' and arithmetic assignment operators '<-+', '<--', '<-*', '<-/', '<-^' and <-%%' (see topic 'arithmetic') also associate from right to left. That is, x <- y <- z is interpreted as x <- (y <- z) and x <-+ y <-+ z is interpreted as x <-+ (y <-+ z). See topics 'assignment' and 'arithmetic'.

Comparison operators do not associate, that is, for example, x < y < z has no meaning. See topic 'logic'.

Precedence of operators

"Precedence" has to do with the interpretations of expressions involving more than one operator, for example 'x <- 3 + 4/5^2' which involves operators '<-', '+', '/' and '^'. Every operator has a numerical precedence level.

The rule is simple: Operators with higher precedence are evaluated before operators with lower precedence.

Table of precedence levels of operations

	Precedence	Meaning
x %| y	1	Bitwise Or (OR)
x %^ y	2	Bitwise Exclusive Or (XOR)
x %& y	3	Bitwise And (AND)
%!x	4	Bitwise Complement (COMPL)
x || y	5	Logical Or
x && y	6	Logical And
!x	7	Logical Not
x == y	8	Equal or same
x != y	8	Not equal or different
x < y	8	Less than
x <= y	8	Less than or equal
x > y	8	Greater than
x >= y	8	Greater than or equal
x + y	9	Addition (sum of x and y)
x - y	9	Subtraction (difference of x and y)
x * y	10	Multiplication (product of x and y)
x / y	10	Division (x divided by y)
x %% y	10	Modular division (x - y*floor(x/y))

```
    x %*% y             11       x MatMult y
    x %c% y             11       transpose(x) MatMult y
    x %C% y             11       x MatMult transpose(y)
    -x                  12       Unary minus ((-1)*x)
    +x                  12       Unary plus ((+1)*x)
    x ^ y or x ** y     13       Exponentiation (x to the y-th power)
    x <- y           14 or 0     Assign value of y to x
    x <-+ y          14 or 0     x <- x + y
    x <-- y          14 or 0     x <- x - y
    x <-* y          14 or 0     x <- x * y
    x <-/ y          14 or 0     x <- x / y
    x <-^ y          14 or 0     x <- x ^ y
    x <-** y         14 or 0     x <- x ** y
    x <-%% y         14 or 0     x <- x %% y
```

In the above MatMult is ordinary matrix multiplication.

You may use parentheses to group terms and change the order of evaluation. Subexpressions within '(...)' or '{...}' are evaluated before the bracketed terms are combined.

The dual levels 0 and 14 for the assignment operators reflect the fact that they have lower precedence than any operator to their right and higher precedence than any operator to their left. For example, x <- y + z sets x to y + z while x + y <- z assigns z to y and then computes the sum of x and the new value of y. Similarly x <-+ y + z is equivalent to x <-+ (y+z) and x + y <-+ z is equivalent to x + (y <-+ z).

Examples

Examples

Expression	Interpretation	Value	Explanation
30/5/2	(30/5)/2	3	/ associates to left
3^2^4	3^(2^4)	43046721	^ associates to right
4*3-2	(4*3) - 2	12 - 2 = 10	* has higher precedence than -
3*2^4	3*(2^4)	3*16 = 48	^ has higher precedence than *
(3*2)^4		6^4 = 1296	Parentheses change evaluation order
-2^4	-(2^4)	-16	^ has higher precedence than prefix -
(-2)^4		16	Parentheses change evaluation order
3<2+4	3 < (2+4)	3 < 6 = T	+ has higher precedence than <
T||F&&F	T||(F&&F)	T||F = T	&& has higher precedence than ||
!T||T	(!T)||T	F||T = T	! has higher precedence than ||
!(T||T)		!T = F	Parentheses change evaluation order

Cross references

See topic 'bit_ops' for examples of how precedence and parentheses affect the order of evaluation of operations '%&', '%|', '%^' and '%!'.

2.272 predtable()

Usage:

```
predtable([keyword phrases] [,silent:T]) or predtable(Term [,keyword
  phrases] [,silent:T]), Term a CHARACTER scalar of the form "A.B. ...",
  where A, B are factors in current GLM model, or term:k, where k is a
  positive integer, and allowed keyword phrases seest:T, sepred:T,
  estimate:F, wtdmeans:T or x:values as for glmtable().
```

Keywords: glm, anova

```
                                  Usage
predtable() computes a table of fitted values (estimated cell expected
values) based on the computations of the most recent GLM (generalized
linear or linear model) command such as anova() or poisson().  The table
has a dimension for each factor in the model, in the order the variables
appear in the model.  It is an error if there are no factors in the
model. See glmtable() for a somewhat more general function.

predtable(seest:T) does the same, except the result is a structure with
two components, 'estimate' and 'SEest' containing the table of fitted
values and their standard errors.

predtable(sepred:T) does the same, except the structure result has
components 'estimate' and 'SEpred', where SEpred contains standard
errors of prediction (usually sqrt(SEest^2 + MSE)) for each cell.

You can use both 'sepred:T' and 'seest:T' together, and can suppress the
table of estimates with 'estimate:F'.

predtable(Term) returns an estimated table of marginal means where the
margins are specified by Term.

Term must be a quoted string or CHARACTER scalar of the form
"Name1.Name2.Name3....", where Name1, Name2, ... are names of factors in
the current GLM model.

When there are k factor names in Term, the value of predtable() is an
array with k dimensions (vector if k = 1, matrix if k = 2), with the
dimensions ordered in the same order as in Term, not the order they
appear in the model if that is different.

predtable(Term, seest:T) does the same, but the result is a structure
with components 'estimate' and 'SEest', where SEest contains the
standard errors of the estimated marginal means.

You cannot use 'sepred:T' with Term when Term specifies a marginal
table, that is, Term does not include all factors in the model.

predtable(term:k [,seest:T]) is essentially equivalent to
predtable(TERMNAMES[k] [,seest:T]), computing the marginal table
matching term k in the model.

You cannot use predtable(Term [,...]) after anova() with a balanced
```

```
design unless Term includes all the factors in the model.

                                Examples
Examples:
  Cmd> anova("y=a+b") # two-way ANOVA
  Model used is y=a+b
  WARNING: summaries are sequential
                  DF          SS          MS
  CONSTANT         1    0.021986    0.021986
  a                2      12.082       6.041
  b                3      12.419      4.1397
  ERROR1          24      39.977      1.6657

  Cmd> predtable() # estimates of cell means
  (1,1)      1.0316     0.23603    -0.43016     -1.3688
  (2,1)     0.98354     0.18795    -0.47824     -1.4169
  (3,1)      2.1081      1.3125     0.64631    -0.29238

  Cmd> predtable(seest:T,sepred:T) #cell mean estimates and SE's
  component: estimate
  (1,1)      1.0316     0.23603    -0.43016     -1.3688
  (2,1)     0.98354     0.18795    -0.47824     -1.4169
  (3,1)      2.1081      1.3125     0.64631    -0.29238
  component: SEest           [SE of estimated cell mean]
  (1,1)      1.2906     0.48563     0.58437     0.57713
  (2,1)      1.4245     0.57943      0.4888     0.57981
  (3,1)      1.4247      0.4894     0.56226     0.66824
  component: SEpred          [SE of prediction for cell]
  (1,1)      1.8252       1.379      1.4168      1.4138
  (2,1)      1.9222      1.4147      1.3801      1.4149
  (3,1)      1.9223      1.3803      1.4078      1.4534

  Cmd> predtable(a,seest:T) # marginal mean estimate and SE's
  component: estimate
  (1)    -0.13284    -0.18092     0.94363
  component: SEest
  (1)     0.44971      0.5415     0.56229

  Cmd> predtable(term:2) #second term is a
  (1)    -0.13284    -0.18092     0.94363

                           Keyword 'silent'
predtable(silent:T) and predtable(Term, silent:T) do the same, except
that certain advisory messages are suppressed.  'silent:T' can be used
with any other keywords.  The default value of 'silent' is False unless
the value of option' 'warnings' is False.

             Behavior when there are variates in the model
The fitted values are by default computed with each variate set to its
unweighted mean value and thus are what are sometimes called the
covariate adjusted cell means.

predtable(wtdmeans:T [,...]) does the same except it adjusts cell fitted
```

values to the weighted means of the variates. You can use wtdmeans:T only when there are variates and when the previous GLM command used weighted OLS (anova() or manova()). This option would be probably appropriate when the weights were proportional to sample sizes.

predtable(x:Vals [,...]), where Vals is a REAL vector with length = the number of variates (non-factors) in the model, does the same computation, except it uses the elements of Vals instead of unweighted or weighted variate means. This option allows you to estimate cell means that are adjusted to any level of the covariates. Use of x:Vals is an error if there are no variates in the current GLM model.

Relationship to glmtable()

predtable() and predtable(Term) are equivalent to glmtable(seest:F) and glmtable(Term, seest:F). Usage of keywords 'seest' and 'sepred' is the same as for glmtable().

Binomial responses

For GLM functions involving a binomial response variable (logistic(), probit(), glmfit() with dist:"binomial"), the values computed are the estimated probabilities p of "success" associated with each cell.

In this case, you can also use keyword phrase n:N, where N is a REAL variable, to specify the number of trials for each cell. N can be a scalar, a vector whose length matches the size of the table, or a matrix or array whose dimensions match those of the table. The resulting table is a table of N*p.

Example:

```
  Cmd> logistic("y=a+b",n:100); predtable(n:100)
```

Caveat about empty cells

Caution: When the marginal table for any term in the model contains empty cells, especially when a factor is nested in another with different numbers of levels, the estimated means may not be what you want.

Behavior with non linear model

After fitting a non-linear model by logistic(), probit(), poisson(), or glmfit(), when Term doesn't contain all the factors in the model, predtable(Term) first computes the estimated marginal table in the linear scale (logit, probit, or log) and then transforms it back into the scale of the response. This means that the computed marginal table is not the marginal means of the fitted table. For example, if b is a factor with 3 levels, after logistic("y=a*b", n:40), sum(predtable("a.b"))/3 is not the same as predtable("b").

Limitation

When keyword phrase coefs:F was an argument on the most recent GLM command, predtable() is not available.

Cross references

See also topics anova(), anovapred(), glmpred(), glmtable(), regpred(),

```
modelinfo(), popmodel(), pushmodel(), 'glm'.
```

2.273 primefactors()

Usage:

```
primefactors(n [, max:T]), n an integer scalar or vector
```

Keywords: general

```
                                  Usage
primefactors(n), where n is a positive integer < 2^52 =
4503599627370496, returns a vector containing the prime factors of n,
possibly with repetitions.

When n is a vector of positive integers <= 999999999999 = 10^12 - 1,
primefactors(n) returns a structure with length(n) components.
Component i is a vector containing the prime factors of n[i] and having
the numeric value of n[i] as its name.  The reason for the smaller range
of permissible values is because a component name can have at most 12
characters.  Note:  Because the component names are numeric and not
alphabetic, you must use a subscript to extract a component.

primefactors(n,max:T), where n is a positive integer scalar or vector
with n[i] < 2^52 = 4503599627370496, returns an integer vector of the
same length as n containing the maximum prime factors of each element of
n.

                                 Examples
Examples:
  Cmd> primefactors(64094231)
  (1)          641        99991

  Cmd> stuff <- primefactors(vector(3,15,64094231)); stuff
  component: 3
  (1)            3
  component: 15
  (1)            3            5
  component: 64094231
  (1)          641        99991

  Cmd> stuff[3] # stuff$64094231 is illegal
  (1)          641        99991

  Cmd> stuff[compnames(stuff) == "64094231"]
  (1)          641        99991

  Cmd> primefactors(64094231, max:T)
  (1)        99991

                             Cross references
```

```
See also goodfactors().
```

2.274 print()

Usage:

```
   print(a, b, ...[,format:Fmt or nsig:m, header:F, labels:F, notes:T,\
     width:w, height:h, macroname:T, missing:missStr, name:setName]\
     [, file:fileName [,new:T]]]), Fmt, missStr, fileName, setName
     CHARACTER scalars, m > 0, w >= 30, h >= 12 integers
```

Keywords: output, missing values

```
                                    Usage
print(a,b, ...) prints objects (variables, expressions, macros) a, b,
....  By default the names 'a', 'b', ... are printed.

By default, print() formats REAL items using the format identified by
'format' on getoptions() output.  This normally is floating point with 5
significant digits.  Type help(options:"format") for details.

  Cmd> print(PI,sqrt(2)*run(5))
  PI:
  (1)      3.1416
  VECTOR:
  (1)      1.4142      2.8284      4.2426      5.6569      7.0711

print() encloses macros and the elements of CHARACTER variables in
quotes ("").  Any internal quotes are escaped with '\' (for example
"\"Hello\"") and non-printable characters are printed as escaped octal
integers (for example, "\033" or "\177").

  Cmd> a <- vector("Charlie","Dog"); print(a)
  a:
  (1) "Charlie"
  (2) "Dog"

print(Message), where Message is a single quoted string or CHARACTER
scalar, prints Message just as it is, without enclosing quotes and
without any internal quotes and non-printable characters escaped.

  Cmd> print("Charlie is a good dog!")
  Charlie is a good dog!

print(Message, macroname:T) does the same, except that when executed in
a macro, " in macro XXXX" is appended to Message.  This is useful for
printing messages in a macro.

You can modify the behavior of print using keywords 'format', 'nsig',
'missing', 'zero', 'labels', 'header', and 'notes'.  See below.

When either of keywords 'format' or 'nsig' are used, print() is
```

```
identical to write() with the same arguments.

print() is particularly useful in macros and inside {...}.

                          Printing to a file
print(a,b,...,file:FileName [,new:T]) where FileName is a quoted string
or CHARACTER variable, writes the output to the specified file rather
than to the screen.  With 'new:T', any information in the file is
discarded before writing.  Without 'new:T', output is appended to the
end of the file.

When FileName is the variable CONSOLE or a CHARACTER variable whose
value is "CONSOLE", the output is written to the screen or output window
rather than to a file.

                           Use of keywords
Keywords 'nsig', 'format', 'name', 'header', 'labels', 'notes',
'missing' and 'zero' are all recognized and can appear more than once.
They affect the printing of objects that follow them, until they are
changed except that a value for 'name' is used only once.  Any of them
that follow all items to be printed are treated as coming before all
items.  For example,
  Cmd> print(x,nsig:5,y,nsig:10)
and
  Cmd> print(nsig:10,x, nsig:5,y)
are equivalent.

Keywords 'file' and 'new' can appear only once, anywhere in the argument
list.

                     Keywords 'nsig' and 'format'
print(nsig:d,a,b,...) or print(a,b,...,nsig:d) prints numbers with d
significant digits in floating point format with width d+7.

  Cmd> print(PI,nsig:7)
  PI:
  (1)       3.141593

print(format:Fmt,a,b,...) or print(a,b,...,format:Fmt), where Fmt is a
quoted string or CHARACTER variable, prints numbers according to
specifications given in Fmt.

Fmt must be of the form "w.df" or "fw.d" (fixed point) or "w.dg" "gw.d"
(floating point) where w (field width) and d (decimals or significant
digits) are integers, for example "6.3f" or "g15.7".  See below for
details.

  Cmd> print(1000*PI,format:"12.6f") # fixed with 6 decimals
  NUMBER:
  (1)  3141.592654

                          Name header keywords
print(name:Name, a, b,... ) prints a with the name specified by
```

```
quoted string or CHARACTER scalar Name on the header.

Name can be of unlimited length aiding the creation of informative
output.

  Cmd> print(3*log(640320)/sqrt(163), nsig:17,\
     name:"3*log(640320)/sqrt(163) is a good approximation to pi")
  3*log(640320)/sqrt(163) is a good approximation to pi:
  (1)       3.1415926535897931

The value of 'name' is used for only one output variable; however, you
can have several instances of name:Name, each affecting the next
variable output.

Alternatively, if the name is a legal MacAnova variable name no more
than 10 characters long you can use a keyword to specify the name.  For
example print(name:"Residuals", r) and print(Residuals:r) are
equivalent.

print(header:F,x,header:T,y,...,) prints x without and y with an
identifying name.

  Cmd> print(PI,header:F)
  (1)      3.1416

                      Keywords 'width' and 'height'
print(a,b,...,width:w) temporarily sets option 'width' to w, an integer
>= 30. This affects how many items are printed per line.

print(a,b,...,height:h) temporarily set option 'height' to h, an integer
>= 12.  This affects the number of lines in any graphs being printed as
"dumb" plots and how often output will be paused in non-windowed
versions.

                            Keyword 'labels'
By default, print() prints coordinate labels, if they exist, or the
index or indices of the first element in each line otherwise.  See topic
'labels'.  Use of 'labels:F' suppresses printing of labels or indices.
A later 'labels:T' re-enables such printing.

  Cmd> print(PI,labels:F) # leading index (1) suppressed
  PI:
        3.1416

                             Keyword 'notes'
Keyword phrase 'notes:T' directs that any notes attached to variables
are printed above the values.

  Cmd> Pi <- vector(PI,labels:"pi",notes:"Copy of variable PI")

  Cmd> print(Pi,notes:T)
  Pi:
  Copy of variable PI
```

```
            pi
                  3.1416
```

See topic 'notes' for details on attached notes.

```
                              Keyword 'missing'
```

print(missing:MissStr1,a,b,...), where MissStr is a quoted string or CHARACTER variable such as "?" or "NA", specifies that all missing values are to be printed using MissStr. If 'missing' is not used, missing values are printed as "MISSING" (or using a different default if you changed it by setoptions(); type help(options:"missing")). Note that this differs from the use of 'missing' on matprint() and matwrite() for which the value must be a REAL scalar.

```
  Cmd> print(vector(PI,?,run(3)),missing:"NA")
  VECTOR:
  (1)      3.1416          NA           1           2           3
```

print(x,file:FileName,new:T,header:F,labels:F,missing:"?") writes x to the file in a form that can be read by vecread().

```
                               Keyword 'zero'
```

print(zero:ZeroStr,a,b,...), where ZeroStr is a quoted string or CHARACTER variable such as " ", "0" or "ZERO" specifies that zero values are to be printed using ZeroStr. If 'zero' is not used, zero values are printed using the same format as other numbers.

```
  Cmd> print(PI*run(-2,2),zero:"Zero")
  VECTOR:
  (1)     -6.2832     -3.1416        Zero      3.1416      6.2832
```

```
                    Details on value of 'format' keyword
```

If Fmt is "w.df" or "fw.d" (fixed point), or "w.dg" or "gw.d" (floating point), integer w specifies a field width of at least w characters. For fixed point format, integer d is the number of digits that will follow the decimal point. For floating point format, d is the number of significant digits printed. If w is omitted (".3f" / "f.3" or ".7g" / "g.7"), it is implicitly set to d+7 ("10.3f"/ "f10.3" or "14.7g" / "g14.7"). If w > 27, width = 27 is assumed and if d > 20, digits = 20 is assumed.

With fixed point output, trailing zeros are kept; for floating point output they are trimmed off. For example, 10.30 is printed as '10.30000' with "8.5f" or "f8.5" format and as '10.3' with "8.5g" or "g8.5" format.

For floating point output, exponential form, 9.3e+07 for example, is used if required to represent the number.

You can change the default format for print() and matprint() by setoptions() using keywords 'nsig' or 'format'. See topics 'setoptions', 'options'.

```
                                Examples
Examples:
  print(nsig:5,a), print(format:"12.5g",a), and print(a,nsig:5),
are equivalent.
  print(nsig:5,file:"myfile",a,new:T)
writes a to file "myfile", starting fresh.

  Cmd> print("Quoted because > 1 argument",vector("a","Escaped\1\2"))
  STRING:
  (1) "Quoted because > 1 argument"
  VECTOR:
  (1) "a"
  (2) "Escaped\001\002"

  Cmd> print("Not quoted because only 1 argument")
  Not quoted because only 1 argument

Keywords that are not otherwised recognized are used to label output.
Keywords may have no more than 10 characters.

  Cmd> print(run(5)) # no labeling keyword
  VECTOR:
  (1)           1           2           3           4           5

  Cmd> print(one2five:run(5)) 3 with labelling keyword
  one2five:
  (1)           1           2           3           4           5

                            Cross references
See also topics 'options', write(), matprint(), matwrite(), paste(),
error().
```

2.275 printoptions()

Usage:

```
   printoptions()
   printoptions(option1:T [,option2:T ...]), where option1,
     option2, ... are option names
```

Keywords: control, general

```
                                  Usage
printoptions(), with no arguments, prints the values of all options in
alphabetical order and in a more concise format than is used by
print(getoptions()).

printoptions(option1:T [,option2:T ...]) prints the values of the named
options in a concise format.  option1, option2, ... must be legal options.

                                Examples
Examples:
```

```
  Cmd> printoptions(format:T,minpvalue:T,seeds:T)
  format = "11.5g"
  minpvalue = 1e-08
  seeds = 518084227 1327950740

  Cmd> printoptions()
  angles = "cycles"
  batchecho = T
  dumbplot = F
  . . . . .
  . . . . .
  warnings = T
  wformat = "16.9g"
  width = 72

                            Cross references
See also getoptions(), setoptions(), 'options'.
```

2.276 probit()

Usage:

```
   probit([Model], n:Denom [, print:F or silent:T, incr:T, offsets:vec,\
     pvals:T, maxiter:m, epsilon:eps, coefs:F]), Denom REAL scalar or
     vector > 0, vec a REAL vector, m an integer > 0, eps REAL > 0
```

Keywords: glm, regression, categorical data

```
                                    Usage
probit(Model,n:Denom) computes a probit regression fit of the model
specified in the CHARACTER variable Model.  If y is the response
variable in the model it must be a REAL vector with y[i] >= 0.  Denom
must either be an REAL scalar >= max(y) or a REAL vector of the same
length as y with Denom[i] >= y[i].  Estimation is by maximum likelihood
on the assumption that y[i] is binomial with Denom[i] trials (Denom
trials for scalar DENOM).  If any y[i] or n[i] is not an integer a
warning message is printed.

To get the coefficients for a classic probit analysis, you should
increase the estimated constant by 5.

probit(Model,n:Denom,...) is equivalent to glmfit(Model,n:Denom,
dist:"binomial", link:"probit",...).

See topic 'models' for information on specifying Model.

                        Side effect variables created
probit() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS, DF,
HII, DEPVNAME, TERMNAMES, and STRMODEL.  See topic 'glm'.  Without
keyword phrase 'inc:T' (see below), TERMNAMES has value vector("","",
...,"Overall model","ERROR1"), DF has value vector(0,0,...,ModelDF,
ErrorDF) and SS has value vector(0,0,...,ModelDeviance,ErrorDeviance).
```

Analysis of deviance

If, say, Model is "y=x1+x2", an iterative algorithm is used to predict invnor(E[y/Denom]) = probit(E[y/Denom]) - 5 as a linear function of x1 and x2. A two line Analysis of Deviance table is printed. Line 1 is the difference 2*L(1) - 2*L(0), where L(0) is the log likelihood for a model with all coefficients 0 (all probabilities = 0.5) and L(1) is the maximized log likelihood for the model fit. Line 2 is 2*L(2) - 2*L(1) where L(2) is the maximized log likelihood under a model fitting one parameter for every y[i]. Under appropriate assumptions, the latter can be used to test the goodness of fit of the model using a chi-squared test.

Incremental fitting

probit(Model,n:Denom,inc:T) computes the full probit model and all partial models -- only a constant term, the constant and the first term, and so on. It prints an Analysis of Deviance table, with one line for each term, representing a difference 2*L(i) - 2*L(i-1) where L(i) is the maximumized log likely for a model including terms 1 through i, plus the deviance of the complete model labeled as "ERROR1". Each line except the last can be used in a chi-squared test to test the significance of the term on the assumption that the true model includes no later terms.

Omitting model

If you omit Model (probit(,n:Denom ...)), the model from the most recent GLM command such as poisson() or anova(), or the model in CHARACTER variable STRMODEL is assumed.

Algorithm

Computations are carried out using iteratively reweighted least squares.

Problimit warning

If you get a warning message similar to the following

```
   WARNING: problimit = 1e-08 was hit by probit() at least once
```

it usually indicates either the presence of an extreme outlier or a best fitting model in which many of the probabilities are almost exactly 0 or 1. The latter case may not represent any problem, since the fitted probabilities at these points will be 1e-8 or 1 - e-8. You can try reducing the threshold using keyword 'problimit' (see below), but you will probably just get the message again.

Other keyword phrases

Keyword phrase	Default	Meaning
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting
epsilon:eps	1e-6	Small positive REAL specifying relative error in objective function (2*log likelihood) required to end iteration
problimit:small	1e-8	Iteration is restricted so that no fitted probabilities are < small or > 1 - small. Value of small must be between 1e-15 and 0.0001.

```
  offsets:OffVec  none   Causes model to be fit to probit(p) to be
                         1*Offvec + Model, where OffVec is a REAL vector
                         the same length as response y.  Note OffVec is
                         in probit units.

See topic 'glm_keys' for information on keyword phrases print:F,
silent:T, coefs:F

The default value for pvals can be changed by setoptions(pvals:T).  See
topics setoptions(), 'options'.

                      Examples on use of 'offsets'
  Cmd> probit("y=x", n:15, offsets:3*x, inc:T, pvals:T)
The P value associated with x can be used to test the hypothesis H0:
beta1 = 3 in the model invnor(p) = beta0 + beta1*x.

  Cmd> probit("y=1", n:20, offsets:rep(invnor(.25),length(y)),\
       inc:T, pvals:T)
The P value associated with the CONSTANT term can be used to test H0:
p = .25, assuming y contains a random sample from a binomial
distribution with n = 20.
```

2.277 prod()

Usage:

```
   prod(x [,squeeze:T] [,silent:T,undefval:U]), x REAL or LOGICAL or a
     structure with REAL or LOGICAL components, U a REAL scalar
   prod(x, dimensions:J [,squeeze:T] [,silent:T,undefval:U]), vector of
     positive integers J
   prod(x, margins:K [,squeeze:F] [,silent:T,undefval:U]), vector of
     positive integers K
   prod(x1,x2,... [,silent:T,undefval:U]), x1, x2, ... REAL or LOGICAL
     vectors, all the same type.
```

Keywords: summary statistics

```
                                   Usage
prod(x) computes the product of the elements of a REAL or LOGICAL vector
x.

If x is LOGICAL, True is interpreted as 1 and False as 0 and hence
prod(x) has value 1 if all elements of x are True and 0 if any is False.

If x is a m by n matrix, prod(x) computes a row vector (1 by n matrix)
consisting of the product of the elements in each column of x.

If x is an array with dimensions n1, n2, n3, ..., y <- prod(x) computes
an array with dimensions 1, n2, n3, ... such that y[1,j,k,...] =
prod(x[i,j,k,...], i=1,...,n1).  This is consistent with what happens
when x is a matrix.  Note: MacAnova3.35 and earlier produced a result
```

with dimensions n2, n3, ...

prod(x, squeeze:T) does the same, except the first dimension of the result (of length 1) is squeezed out unless the result is a scalar. In particular, if x is a matrix, prod(x,squeeze:T) will be identical to vector(prod(x)), and if x is an array, prod(x,squeeze:T) will be identical to array(prod(x),dim(x)[-1]).

prod(NULL) is NULL.

prod(a,b,c,...) is equivalent to prod(vector(a,b,c,...)) if a, b, c, ... are all vectors. They must all have the same type, REAL or LOGICAL or be NULL. prod(NULL, NULL, ..., NULL) is NULL.

prod(x, silent:T) or prod(a,b,c,...,silent:T) does the same but suppresses warning messages about MISSING values or overflows.

If all the elements of a vector x are MISSING, prod(x) is 1.0.

prod(x, undefval:U), where U is a REAL scalar does the same, except the returned value is U when all the elements of x are MISSING.

Structure argument

If x is a structure, prod(x) computes a structure, each of whose components is prod() applied to that component of x.

Keyword 'dimensions'

prod(x, dimensions:J [,squeeze:T] [,silent:T] [undefval:U]) computes products over the dimensions in J = vector(j1,j2,...,jn) where j1, ..., jn are distinct positive integers <= ndims(x). Without 'squeeze:T', the result has the same number of dimensions as x, with dimensions j1, j2, ..., jn of length 1. With 'squeeze:T', these dimensions are removed from the result. The order of j1, j2, ... is ignored.

It is an error if max(J) > ndims(x) or if there are duplicate elements in J.

For example, if x is a matrix, prod(x, dimensions:2) computes the row products as a nrows(x) by 1 matrix and prod(x, dimensions:2,squeeze:T) computes them as a one dimensional vector.

Keyword 'margins'

prod(x, margins:K [,squeeze:F] [,silent:T] [undefval:U]) computes products over the dimensions not in K = vector(k1, k2, ..., km), where k1, ..., km are distinct positive integers <= ndims(x). This computes marginal products for the margins specified in K.

Without 'squeeze:F', only the dimensions in K are retained in the result. Otherwise the other dimensions are retained but have length 1. This is opposite from the default with 'dimensions:J'.

It is an error if max(K) > ndims(x) or if there are duplicate elements in K.

```
                                Example
Examples:
  Cmd> x # matrix with labels
              B1          B2
  A1          18          15
  A2          17          26
  A3          18          19

  Cmd> prod(x) # products down columns
               B1          B2
  (1)        5508        7410

  Cmd> prod(x)/x # elements in row are products of other rows of x
              B1          B2
  A1         306         494
  A2         324         285
  A3         306         390

  Cmd> prod(x,dimensions:2) # products accross rows
             (1)
  A1         270
  A2         442
  A3         342

  Cmd> prod(x,margins:1) # same as a vector
            A1          A2          A3
          270         442         342

                           Cross references
See also topics sum(), 'NULL'.
```

2.278 propinterval()

Usage:

```
propinterval(x[,y],cover:fraction,[options])
propinterval(s1,n1[,s2,n2],cover:fraction,[options])
options can include upperb:T or lowerb:T, plus4:T|F.
```

Keywords: probabilities, descriptive statistics, comparisons

```
propinterval() computes a z-confidence interval for a population
proportion or difference of population proportions, depending on
whether data for one or two variables are given as arguments.  By
default, propinterval() computes a two-sided interval, but you may
choose one-sided alternatives by using one of lowerb:T or upperb:T.
Arguments may be either vectors of 0/1 data, or may be counts of
successes and numbers of trials.

You specify the coverage rate via cover:value.
```

```
You may specify the Agresti "add 4" procedure via plus4:T.  This
will add 2 successes and two failures (split among the two samples
for two-sample tests).

The output is a vector containing the estimate and interval bounds.
```

2.279 proptest()

Usage:

```
proptest(x[,y],null:val,[upper:T or lower:T,contcor:T|F,pool:T|F])
proptest(s1,n1[,s2,n2],null:val,[upper:T or lower:T,contcor:T|F,pool:T|F])
```

Keywords: probabilities, descriptive statistics, comparisons

```
proptest() performs a one- or two-sample z-test for proportions,
depending on whether data for one or two variables are given as
arguments.  Arguments may be either vectors of 0/1 data, or may be
counts of successes and numbers of trials.  By default there is
a two-tailed alternative, but you may choose one-sided alternatives
by using one of lowertail:T or uppertail:T.

You specify the null value via null:value.  For a two-sample test,
only null:0 is allowed.

Use of contcor:T will cause the p-value to be calculated with a
continuity correction.

For a two-sample test, you may specify that the variances be pooled
or unpooled via pool:T or pool:F.

The output is a vector containing the z-statistic and the p-value.
```

2.280 pushmodel()

Usage:

```
pushmodel()
pushmodel(canpush:T)
```

Keywords: anova, glm, multivariate analysis, regression, residuals

```
                              Introduction
All GLM commands except screen(), for example regress() and anova(),
retain information (GLM information) internally for use by certain
functions such as coefs() and modelinfo() that provide results from the
most recent GLM command.  When a GLM command is run, information from
the previous GLM command, if any, is replaced.  When the GLM command is
in a macro, this may confuse the user who may expect that coefs(), say,
gives the same answer after the macro is used as before.
```

Commands pushmodel() and popmodel() allow a macro to save and restore the current GLM information.

pushmodel() saves the current GLM information so that it can subsequently be restored by popmodel(). Until a new GLM command has been run or GLM information has been restored by popmodel(), there is no GLM information available to commands such as secoefs() and modelinfo().

After the next prompt following the use of pushmodel(), the active model before the first use of pushmodel() is restored (equivalent to an automatic execution of popmodel(all:T)).

pushmodel() and popmodel() are intended to be used in macros. A macro can run pushmodel() before a GLM command and then run popmodel() before finishing to ensure that the current GLM information is not changed. Of course, if the purpose of the macro is to change the information, it should not use pushmodel() and popmodel().

The default maximum number of sets of GLM information that can be saved is 2 (0 in limited memory DOS version). On versions allowing command line arguments you can change the default by '-savemodels N' where N >= 0 is an integer. See 'launching'.

Usage

pushmodel() saves the current GLM information from the most recent GLM command. Until a subsequent GLM command or popmodel() has been run, or a new prompt is printed, no model information is available for retrieval by functions like secoefs() and modelinfo(). You can use popmodel() to restore the saved GLM information.

It is an error if there is no current model to save or you have already saved the maximum number of models (default is 2).

pushmodel(canpush:T) returns True if there is current GLM information to save and a place to put it and False otherwise. No GLM information is saved and the current GLM information, if any, is not changed.

Example

Here is the text of a macro aov() which returns the SS and DF of an analysis of variance without changing the current GLM information or side effect variables, if any.

```
  if (pushmodel(canpush:T)){pushmodel()}
  anova($1, silent:T)
  @result <- structure(SS,DF)
  if (popmodel(canpop:T)){popmodel()}
  @result #will be returned as value of the macro
```

This might be used as follows"

```
  Cmd> aov("y=a") # this won't change GLM info or side effect variables
  component: SS
```

```
      CONSTANT           a      ERROR1
        3.8646    0.021371      4.8536
  component: DF
      CONSTANT           a      ERROR1
             1           2           7

                          Cross references
See also popmodel(), modelinfo(), coefs(), secoefs(), contrast(),
'macros'.
```

2.281 putascii()

Usage:

```
putascii(vec [,keep:T]), vec a vector of integers > 0 and <= 255
putascii(vec, file:fileName [, new:T])
```

Keywords: output, character variables

```
                                  Usage
putascii(Vec) prints the characters corresponding to the elements of the
vector Vec, considered as ASCII codes.  All the codes must be integers
between 1 and 255, inclusive.  For example, putascii(run(64,126)) prints
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~

putascii(Vec1,Vec2,...) is equivalent to putascii(vector(Vec1,Vec2,
...)), if Vec1, ..., are REAL vectors.

The primary use of putascii() is to send control sequences to a
terminal.  See macro vt() for an example of its use.  On all machines,
any leading 7's are explicitly translated to beeps or bells.  For
example
  Cmd> putascii(vector(7,7,7,69,82,82,79,82))
rings the bell three times and prints ERROR.

getascii() is almost an inverse to putascii(), translating CHARACTER
variables to a vector of integers.

                           Writing to a file
putascii(Vec,file:FileName [,new:T]) or putascii(Vec1, Vec2, ...,
file:FileName [,new:T]), where FileName is a CHARACTER variable or
quoted string, writes the ASCII codes on file FileName.  If new:T is an
argument, any information in the file will be destroyed; otherwise the
codes are added at the end of the file.

                           Keyword 'keep'
putascii(Vec,keep:T) or putascii(Vec1,Vec2,...,keep:T) returns a
CHARACTER variable whose characters have the ASCII codes.  For example,
putascii(run(4,8), keep:T) has value "\004\005\006\007\010".  Use of
new:T with file:FileName is illegal.

                     Difference from makesymbols()
```

```
The usage putascii(Vec, keep:T) is somewhat similar to makesymbols(Vec,
keep:T) since both translate ASCII codes to characters.  The difference
is that putascii() returns a CHARACTER scalar with length(Vec)
characters while makesymbols() returns a CHARACTER vector of
length(Vec), with each element a single character.  See makesymbols()
for details.

                            Cross references
See also print(), write().
```

2.282 qr()

Usage:

```
   qr(x [,pivot:T, ronly:T]), x a REAL matrix
```

Keywords: matrix algebra

```
                                  Usage
qr(x) computes the elements of a QR decomposition of matrix x

The value returned is structure(qr:Qr, qraux:Qraux), where Qr is a REAL
vector of length P and Qraux is a REAL n by p matrix, as computed by
Linpack subroutine dqrdc.  The elements of Qr on and above the diagonal
constitute the upper triangular matrix R of the QR decomposition, and
the remaining elements, together with the elements of Qraux contain
enough information to compute Q.  No pivoting is performed with this
usage of qr().

qr(x,pivot:T) or simply qr(x,T) does the same computation, with possible
reordering of columns.  The result is a structure(qr:Qr,qraux:Qraux,
pivot:Pivot), where Pivot is a vector of length p containing the column
numbers in x corresponding to the successive columns of q and r.

qr(x,ronly:T) returns a p by p upper triangular matrix consisting of the
R matrix in the QR decomposition, computed without pivoting.  For
example, both parts of the QR decomposition can be computed by
  Cmd> R <- qr(x,ronly:T) ; Q <- x %/% R # x %*% solve(R)

The columns of Q are orthogonal and x - Q %*% R will be zero except for
rounding error.

                             Macro qrdcomp()
An alternative way to get the full QR decomposition is by macro
qrdcomp() in macro file math.mac which uses the information in Qraux.
Type help(qrdcomp) for details.

                            Cross references
See also cholesky().
```

2.283 quitting

Usage:

```
quit or bye or end or stop or exit
quit(F) or bye(F) or end(F) or stop(F) or exit(F)
```

Keywords: general

```
                                Usage
To terminate a MacAnova run, type 'quit'.  On a windowed version,
you will be asked if you want to save the workspace and any
command/output windows.

quit() or quit(T) has the same effect as 'quit'.

quit(F) means you want to quit unconditionally with no opportunity to
save the workspace or windows.  On a version without Windows, it is no
different from quit(T).

It is an error for quit to be part of a compound command (enclosed in
{...}) or for there to be anything other than a comment starting with
'#' after it on the command line.

'stop', 'end', 'bye' and 'exit' mean the same as 'quit'.

                                Example
Example:
  Cmd> quit  # or bye or stop() or end(T), etc.

                      Quitting in windowed versions
In a windowed version, you can also select Quit on the File Menu.
You will be asked whether you wish to save the workspace or the
command/output window(s).  On Mac OS 9, if you hold down the Option
key while selecting Quit or hitting Return after typing 'quit', you
quit unconditionally without a chance to save files.

                            Cross references
See also topic 'launching'.
```

2.284 rank()

Usage:

```
rank(x [ ,down:T, ties:"ignore" or "average" or "minimum"]), x REAL or
  CHARACTER or a structure with all REAL or all CHARACTER components.
```

Keywords: ordering

```
                                Usage
rank(x) computes the ranks of the data in variable x, the minimum value
being assigned rank 1.  In case of ties, the rank corresponding to the
average of the ranks for the tied cases is computed.  x can be REAL or
```

CHARACTER.

When x is CHARACTER, x[i] is considered greater than x[j] if x[i] follows x[j] in alphabetical order using the ASCII collating sequence. See sort() for the explicit ordering of characters.

rank(x,down:T) does the same except that the data are ranked in decreasing order, with rank 1 assigned to the maximum element.

rank(x,ties:Method) or rank(x,ties:Method,down:T), where Method is one of "average", "minimum" or "ignore" (or simply "a", "m" or "i"), treats ties as described below. With REAL x, rank(x,ties:"average" [,down:T]) gives the same ranks as rank(x [,down:T]). When x is CHARACTER, keyword 'ties' is ignored and is assumed to have value "i".

rank(x [keywords]) has the same labels as x, if any.

Matrix, array, or structure argument

When x is a matrix, the result is a matrix each of whose columns contains the ranks of the elements of the corresponding column of x.

When x is an array with dimensions n1, n2, ..., the result is an array of the same size and shape with all the elements with fixed values of subscripts 2, 3, ... defining a "column" whose ranks are computed. An array with dimension > 2 is always treated as an array and not as a matrix, even if there are at most two dimensions greater than 1.

It is also acceptable for x to be a structure, whose non-structure components are all REAL or all CHARACTER. In that case, rank() returns a structure of the same form, each of whose non-structure components is the result of applying rank() to the corresponding component of x.

MISSING values

When x is REAL and there are MISSING values in a column, their rank is also MISSING and the maximum rank of the non-missing values is the number of non-missing values.

Treatment of ties

Treatment of Ties with REAL Data

Suppose k elements in a vector (column) are tied, that is they all have the same value and no other element has this value, and suppose the ranks these elements would have if their values were very slightly changed so as to break the ties while preserving other ordering would be r, r+1, r+2, ..., r+k-1. The following describes the ranks computed for the tied values for each of the three possible methods.

Value for 'ties'	Computed ranks
"average" or "a" (default)	All ranks = (r+(r+1)+...+(r+k-1))/k = r+(k-1)/2
"minimum" or "m"	All ranks = r
"ignore" or "i"	r, r+1, r+2, ..., r+k-1; which ranks go in which position is unpredictable

```
                                Examples
Examples:
  Cmd> x <- vector(27,22,25,26,22,21,?,24) # 1 tie, 1 MISSING

  Cmd> rank(x) # or rank(x,ties:"a")
  WARNING:  MISSING values in argument to rank
  (1)           7         2.5           5           6         2.5
  (6)           1     MISSING           4

  Cmd> rank(x,down:T)
  (1)           1         5.5           3           2         5.5
  (6)           7     MISSING           4

  Cmd> rank(x,ties:"m")
  (1)           7           2           5           6           2
  (6)           1     MISSING           4

  Cmd> rank(x,ties:"m",down:T)
  (1)           1           5           3           2           5
  (6)           7     MISSING           4

  Cmd>  rank(x,ties:"i")
  (1)           7           2           5           6           3
  (6)           1     MISSING           4

In each example except the first, a warning message about MISSING values
has been deleted.

                            Cross references
See also grade(), sort().
```

2.285 rankits()

Usage:

```
   rankits(x [,ties:"ignore" or "average" or "minimum"]), x REAL or a
     structure with REAL components.
   rankits(n:N), integer N > 0
```

Keywords: transformations, descriptive statistics, ordering

```
                                  Usage
rankits(x) computes the vector of rankits (normal scores) for data in
REAL vector x.

rankits(n:N), N a positive integer, is equivalent to rankits(run(N)).

An important use is plot(rankits(x),x) which produces a rankit or normal
scores plot of the values in x.  What is computed is equivalent to
            invnor((rank(x,ties:"ignore") - .375)/(n + .25))
where n is the number of non-MISSING values.  The value corresponding to
a MISSING value is MISSING.
```

```
rankits(x [keywords]) has the same labels as x, if any.

                            Handling of ties
rankits(x,ties:method), where method is "ignore", "average", or
"minimum" (or "i", "a", "m") computes
             invnor((rank(x,ties:method) - .375)/(n + .25))
See rank() for a detailed discussion of the three methods.  It is hard
to think of a situation when you would want to use "minimum" with
rankits().

                   Matrix array or structure argument
When x is a matrix, the result is a matrix each of whose columns
contains the rankits for the corresponding column of x.

When x is an array, rankits(x) is an array of the same size and shape
with all the elements with fixed values of subscripts 2, 3, ... defining
a "column" whose rankits are computed.  An array with dimension > 2 is
always treated as an array and not as a matrix, even if there are at
most two dimensions greater than 1.

It is also acceptable for x to be a structure, whose non-structure
components are all REAL.  In that case, rankits() returns a structure of
the same form, each of whose non-structure components is the result of
applying rankits() to the corresponding component of x.

                                 Example
  Cmd> x <- vector(10.59,18.82,19.46,13.34,13.49)#ranks are 1,4,5,2,3

  Cmd> rankits(x)
  (1)     -1.1798      0.4972      1.1798     -0.4972           0

                            Cross references
See also halfnorm().
```

2.286 rational()

Usage:

```
   rational(x, a, b) or rational(x, a) or rational(x,,b), x REAL or a
     structure with REAL components, a and b REAL vectors
```

Keywords: transformations

```
                                  Usage
rational(x,a,b) computes a rational function of the REAL vector, matrix,
or array x, with coefficients for the numerator and denominator
polynomials in REAL vectors a and b.  The result is REAL with the same
size and shape as x.

The rational function computed is
   (a[1] + a[2]*x + a[3]*x^2+ ... )/(b[1] + b[2]*x + b[3]*x^2+ ... ) .
```

If a or b is omitted, it is construed to represent the constant 1. For example, rational(x,a), rational(x,a,), and rational(x,a,1) are equivalent and compute a polynomial in x, and rational(x,,b), and rational(x,1,b) are equivalent and compute the reciprocal of a polynomial in x.

x can also be a structure, all of whose non-structure components are REAL, in which case the result is a similar structure.

Example

An important use of rational() is in writing macros to compute mathematical functions that are not directly available in MacAnova but which can be approximated by rational functions. For example

```
  Cmd> cumnor1 <- macro("@x <- $1; @posx <- @x > 0
     @posx - (@posx - .5)*rational(abs(@x),1,\\
       vector(1,.196854,.115194,.000355,.019527))^4", dollar:T)
```

creates macro cumnor1() which uses an approximation due to Hastings to cumpute cumulative normal probabilities. For more elaborate use, see macros i0() and i1() in file Math.mac distributed with MacAnova.

Cross references

See topics macro() and 'macros'.

2.287 rbin()

Usage:

```
rbin(N, n, p), N positive integer, n scalar or vector of positive
  integers, p scalar or vector of probabilities.
```

Keywords: random numbers

Usage

rbin(N, n, p) returns a vector of N independent binomial pseudo-random variables with sample size n and probability p. N must be a positive integer.

n must be a positive integer scalar or a vector of N positive integers. p must be a REAL scalar between 0 and 1 or a vector of N values between 0 and 1. If n or p is a scalar, it is used for every element of the result. Otherwise, n[i] and/or p[i] are used for the i-th element of the result.

If the random number generator has not been initialized by setseeds(), setoptions() or previous use of rbin(), rnorm(), rpoi() or runi(), the generator's "seeds" will be initialized automatically using the current time and date, and their values will be printed out.

Use in generating other distributions

```
rbin() can be used to generate other random variables by using a random
vector as p and/or n.  For example,

  Cmd> y <- rbin(N, n, invbeta(runi(N),a, b)) # a, b > 0

will generate a pseudo-random beta-binomial sample variables with
parameters n, a and b.  See the User's Guide for details.

                                Algorithm
The generation algorithm is adapted from Voratas Kachitvichyanukul and
Bruce Schmeiser, "Binomial random variate generation", Commun.ACM, 31
(1988) 216-222, published as Algorithm 678, Trans. Math. Software 15,
394-397.

                             Cross references
See also topics setseeds(), getseeds(), setoptions(), rnorm(), rpoi(),
runi(), invbeta(), cumbin(), 'options'.
```

2.288 read()

Usage:

```
x <- read(FileName, setName or macroName [,quiet:T or F, echo:T or F,\
  printname:F,labels:Labels, silent:T, notfoundok:T, nofileok:T,\
  badkeyok:T, prompt:F]), fileName and setName CHARACTER scalars;
  FileName can also be CONSOLE or have the form string:charVal where
  charVal is a CHARACTER scalar or vector.
```

Keywords: macros, input, files, missing values

```
                Relationship to matread() and macroread()
read() can be used instead of either matread() and macroread().  It
recognizes the same keywords.  In fact, the only difference from
matread() and macroread() is that matread() gives a warning message when
reading a macro and macroread() gives a warning message when reading a
data set.  read() makes no such complaint.  See matread() and
macroread() for details.

                             Cross references
See also topics getdata(), getmacros(), vecread(), readcols(),
'matread_file', 'macro_files'.
```

2.289 readcols()

Usage:

```
readcols(FileName,name1,name2,...,namek[,keyword phrases]). FileName a
  quoted string or CHARACTER scalar, name1, ... quoted or unquoted
  variable names.  FileName can also be CONSOLE or have the form
  string:charVal where charVal is CLIPBOARD or other CHARACTER scalar or
  vector.  Keyword phrases may 'realorchar:T' or any vecread() keyword
  phrases except startline:M
readcols(FileName,vector("name1",...,"namek")[,keyword phrases])
readcols(FileName[,keyword phrases]), the first line of the file
  containing names.
```

Keywords: input, files

```
                                  Usage
readcols(FileName,name1,name2,...,namek) uses vecread() to read
numerical data from file FileName and puts the columns in variables
name1, name2, ..., namek which be quoted or unquoted.

readcols(FileName,vector("name1",...,"namek")) is an alternative usage.

The file should consist of k columns of numbers separated by spaces,
commas or tabs, with MISSING values indicated by '?' or '.'.  See topics
'vecread_file' and vecread() for a complete description of the file
format, including 'skip', 'skipthru', 'go', and 'stop' characters.

For all usages, the number of variable names must divide the total
number of data values.

                        Variable names from file
readcols(FileName), with no variable names provided, does the same,
except that the names are taken from the first non-blank line of the
file, with data assumed to start on the next line.  An informative
message about the variables created is printed.

                           Forms for filename
FileName can take two forms:
 A quoted string or CHARACTER scalar whose value is the file name.  In
 a version with windows, when FileName is "", you can select the file
 using a dialog box.  A variant is keyword phrase file:FileName.

 The keyword phrase string:CharVector, where CharVector is a CHARACTER
 scalar or vector which is "read" instead of a file.  When
 length(CharVector) > 1, each element starts a new line.  Any newline
 characters '\n' terminate lines. In windowed versions, string:CLIPBOARD
 can be useful.  See vecread(), 'CLIPBOARD'.

                         Reading CHARACTER data
readcols(FileName,name1,name2,...,namek, bywords:T) and
readcols(FileName, bywords:T) do the same, except that CHARACTER vectors
are created.

readcols(FileName,name1,name2,...,namek, realorchar:T) and
```

```
readcols(FileName, realorchar:T) do the same, except both REAL and
CHARACTER vectors may be created.  If the first value in a column is
readable as a number, the corresponding variable will be REAL; otherwise
the variable will be CHARACTER.

                              Other keywords
readcols() recognizes the same keyword phrases as vecread(), except
'startline:M' and 'bylines:T'.  These include skip:skipChar,
stop:stopChar, go:goChar, skipthru:skipthruChar, bypass:m, quiet:T or F,
and echo:T or F.  You can't use 'bywords:T', 'bychar:T' or 'byfields:T'
with 'realorchar:T'.  See vecread().

                                 Examples
Examples:
  Cmd> readcols("hald.txt",x1,x2,x3,x4,y)
  Cmd> readcols(file:"hald.txt",vector("x1","x2","x3","x4","y"))

both create variables, x1, x2, x3, x4 and y from the five columns of
data in hald.txt.

  Cmd> readcols(string:"x y\n 1 2\n 7 9\n ? 4")
  Cmd> readcols(string:vector("x y","1 2","7 9","? 4"))

both of which have the same effect as
  Cmd> x <- vector(1,7,?); y <- vector(2,9,4)

                             Cross references
See also topics readdata(), vecread(), 'macros'.
```

2.290 readdata()

Usage:

```
readdata(filename,name1,...,namek [,factors:F  or sort:F]
  [,keyword phrases]), filename a CHARACTER scalar, name1,... names,
  quoted or unquoted variable names
readdata(filename,vector("name1",...,"namek") [,factors:F or sort:F]
  [,keyword phrases])
readdata(filename [,factors:F or sort:F] [,keyword phrases])
```

Keywords: input, files

```
                                   Usage
readdata(FileName,name1,name2,...,namek) uses vecread() to read
one or more columns of data from file FileName, creating variables.
name1, name2, ..., namek.  The variable names can be either quoted
("weight") or unquoted (weight) but must be legal MacAnova names.

When the value of a variable in the first line of the data is not a
number, the values for that variable are read as CHARACTER data and then
turned into a factor, with factor levels maintaining the alphabetical
order of the CHARACTER values.
```

readdata(FileName,name1,...,sort:F) does the same except if any factors are created, the factor levels are assigned in the order CHARACTER values are encountered.

readdata(FileName,name1,...,factors:F) does the same except a column starting with a non-numerical word is not translated into a factor but is read as a CHARACTER vector.

readdata(FileName [,factors:F or sort:F]) does the same except the names for the variables are expected to be in the first line of the file.

For all usages, the number of variable names, whether given as arguments or taken from the first line of the file, must divide the total number of data values.

Printed output

A line giving the name and type (REAL, factor or CHARACTER) is printed for each variable read. If the variable is REAL and has MISSING values, the number of MISSING values is also printed. Argument 'quiet:T' suppresses this output.

Any line in the file starting with skip character '#' is automatically skipped and is echoed to output by default.

Relation to readcols()

readdata() is intended as a replacement for readcols(). It can handle files in which some data columns are non-numerical and can get variable names from the first line of the file.

File format

The file should consist of k columns of "words" separated by spaces, commas or tabs. A word is any set of consecutive characters not including a comma, space or tab.

A column whose first word is a number or MISSING ('?', '.', '*' or 'NA') is read as a REAL vector. Any word in the column after the first which is not readable as a number (for example 5a3) is read as MISSING.

When the first word in a column is not a number or a code for MISSING, the corresponding variable is a factor, by default, with a level for each distinct word in the column. The factor has the original words in the file as row labels. With 'factor:F' as an argument, the column is read as a CHARACTER variable instead of a factor.

Forms for filename

FileName can take two forms:

A quoted string or CHARACTER scalar whose value is the file name. In a version with windows, when FileName is "", you can select the file using a dialog box. A variant is keyword phrase file:FileName.

The keyword phrase string:CharVector, where CharVector is a CHARACTER scalar or vector which is "read" instead of a file. When

```
 length(CharVector) > 1, each element starts a new line.  Any newline
 characters '\n' terminate lines. In windowed versions, string:CLIPBOARD
 can be useful.  See vecread(), 'CLIPBOARD'.

                            vecread() keywords
You can use most vecread() keywords 'quiet', 'silent', 'stop', 'skip',
'skipthru', 'go', 'quiet', 'echo', and 'n', but not 'bypass', 'bywords',
'bylines', 'bychars', 'byfields' and 'realorchar'.  See topic
'vecread_keys'.

                             Cross references
See also readcols(), vecread(), 'vecread_files'.
```

2.291 redo()

Usage:

```
redo() or redo(charVar) where charVar is CHARACTER scalar
REDO()
```

Keywords: control

```
                                   Usage
redo() re-executes the previous line.  It can be used in an expression
3*redo() or as an argument to a function (sqrt(redo())).

What redo() actually does is the following:
 1. redo() creates a macro REDO from the entire preceding command line
    which is automatically saved as variable LASTLINE
 2. redo() then executes REDO, thus re-running the preceding command
    line.  Re-execution may not be exact.  If the preceding line
    consisted of several commands separated by semi-colons, they will
    all be executed, but values that are not assigned will not be
    printed, except for the final command.  And even if the final
    command in the preceding line is an assignment, its value may be
    printed.

In later lines, just typing REDO() will re-execute this line (until a
subsequent use of redo()).

Caution: do not attempt to use redo() immediately following a line
containing redo() or REDO(), as this leads to uncontrolled recursion.

redo(charVar) also creates macro REDO and executes it, but the contents
of REDO come from CHARACTER scalar charVAR rather than LASTLINE.

redo() is implemented as a pre-defined macro.

                                  Examples
Examples:
  Cmd> print(paste("Pi =",PI))
  Pi = 3.1416
```

```
  Cmd> redo() # previous command repeated
  Pi = 3.1416

  Cmd> REDO() # redone command repeated
  Pi = 3.1416

  Cmd> pi <- PI # line ending in assignment

  Cmd> redo() # previous line executed and value of assignment printed
  (1)      3.1416

  Cmd> pi <- PI ;; # note trailing ;; so last command is null

  Cmd> redo()  # previous line executed and nothing printed

  Cmd>

                            Cross references
See also topics edit(), 'macros', 'syntax'.
```

2.292 regcoefs()

Usage:

```
   regcoefs(Model [,pvals:T] [,byvar:F]) or regcoefs([pvals:T] [,byvar:F]),
     where Model is a CHARACTER scalar
```

Keywords: glm, anova, regression, confidence intervals

```
                                  Usage
regcoefs(Model) returns a matrix with appropriately labeled rows and
columns of the regression coefficients, their standard errors and
t-statistics from a least squares fit to the regression model specified
by Model.  There can be no factors in Model.  If Model is omitted, the
most recent GLM model is used.

regcoefs(Model,pvals:T) or regcoefs(pvals:T) also computes two-tail P
values corresponding to the t-statistics on the basis of Student's
t-distribution with degrees of freedom from the last element of side
effect variable DF.

Because of the presence of row and column labels, after any GLM command
with a model without factors, typing regcoefs([pvals:T]) produces a
table similar to that produced by regress().  After non-linear fits such
as logistic() or poisson(), the P-values will not necessarily be
appropriate.

                           Multivarate response
If the response variable is multivariate, the result is a structure,
each of whose components is a labeled matrix of coefficients, standard
errors and t-statistics.  regcoefs(Model,byvar:F) or regcoefs(byvar:F)
```

```
returns a single labeled matrix, with separate columns for the
coefficients, standard errors, ... for each variable.

                            Cross references
See also topics 'glm', regress(), secoefs().
```

2.293 regpred()

Usage:

```
regpred(vals [, silent:T]), vals a REAL vector or matrix.
```

Keywords: glm, regression

```
                                  Usage
regpred(vals) computes the fitted (predicted) value, the standard error
of estimation, and the standard error of prediction for the current
regression model when the X variables have values given by the REAL
vector or matrix vals.

When there is only 1 variate in the model, vals is a scalar or a vector
and the estimates and standard errors are computed for each element of
vals.

When the number of variates in the model is nvars > 1, vals must either
be a vector of length nvars containing data for a single case, or a m by
nvars matrix containing data for m cases.  In the latter case, each
component of the result is a vector of length m.  For example, after
regress("y=x1+x2+x3"), regpred(hconcat(x1,x2,x3)) computes the predicted
values and standard errors for all cases in the data set.

The result is a structure with components 'estimate', 'SEest', and
'SEpred'.

When the error degrees of freedom are 0, all standard errors are set to
MISSING.

Caution: After anova(), manova() and regress(), standard errors are
computed using the final error mean square in the model.  This may not
be appropriate with mixed models, including split plot designs.

                                 Keywords
regpred(vals, silent:T) does the same except certain advisory messages
are suppressed.  The default value of 'silent' is False unless the value
of option' 'warnings' is False.

You can also use keyword phrases estimate:F, seest:F, sepred:F and n:N.
See glmpred() for details.

                                 When used
You can use regpred() after any GLM command as long as there are no
```

```
factors in the model.  The output has no SEpred component except after
regress(), anova() or manova() or their weighted versions.

After anova(), manova() and regress(), regpred(vals) is equivalent to
glmpred(vals,sepred:T).  After other GLM commands, regpred(vals) is
equivalent to glmpred(vals).

                            Cross references
See also topics regress(), anova(), glmpred(), predtable(), glmtable(),
modelinfo(), popmodel(), pushmodel(), 'glm', yhat().
```

2.294 regress()

Usage:

```
regress([Model] [,print:F or silent:T,pvals:T,coefs:F,marginal:T])
```

Keywords: glm, regression

```
                                   Usage
regress(Model) performs a least squares fit of the regression model
given in the quoted string or CHARACTER variable Model.  It prints out
the regression coefficients, their standard errors, and t statistics,
plus other summary statistics (see below).  If option 'pvals' is True,
regress() also prints P values for each coefficient based on Student's t
distribution.  See subtopic 'options:"pvals"'.

No ANOVA table is printed by regress().  To see one, type 'anova()' as
the next GLM command after regress().

                                 Examples
Examples (y, x, x1, x2, and x3 all REAL vectors of length 10):
   regress("y = x")                 Simple linear regression of y on x
   regress("y = x - 1")             Linear regression through origin of
                                    y on x
   regress("y = {run(10)}")         Simple linear regression of y on
                                    vector(1,2,3,4,5,6,7,8,9,10)
   regress("y = x1 + x2 + x3")      3 variable multiple regression of y
                                    on x1, x2 and x3
   regress("{sqrt(y)} = x + {x^2}") Quadratic polynomial regression of
                                    sqrt(y) on x
   regress("{sqrt(y)} = P2(x)")     Same as preceding.

                            Keyword 'weights'
regress(Model,weights:Wts) performs a weighted least squares fit, using
REAL vector Wts as case weights. The elements of Wts must not be
negative.  The results are what you would get by multiplying by
sqrt(Wts) the response vector and all independent variables, including
the contant vector, and then doing a least squares fit.  You can
abbreviate 'weights:Wts' to 'wts:Wts'.
```

Model for regress()

Model is of the form "Response = Var1 + Var2 + ... + Vark", where Response, Var1, ..., Vark are either variable names or have the form {expr}, where expr is a MacAnova expression. All variables or evaluated expressions must be REAL with the same number of rows. The variables to the right of '=' must be vectors or n by 1 matrices. If any right hand side variable is actually a factor, it is treated as a quantitative variate whose values are the levels of the factor. The associated sum of squares has only 1 degree of freedom regardless of the number of levels of the factor.

Regression through origin

You specify regression through the origin by including "-1" in the model. See also topic 'models'.

No model specified

regress() or regress(,weights:Wts) with no model specified computes a least squares regression using the same model as was used by the most recent GLM command such as regress(), anova(), or poisson(). See topic 'glm'.

Printed output

Other printed output from regress() includes multiple R-squared, the overall F-statistic for the model excluding the constant term, the mean squared error and the Durbin Watson statistic.

Side effect variables created

regress() computes side effect variables RESIDUALS, HII, SS, DF, DEPVNAME, TERMNAMES, STRMODEL, COEF, and XTXINV. When weights are used, RESIDUALS = response - fit and WTDRESIDUALS = sqrt(Wts)*RESIDUALS is an additional side effect variable. When an independent variable is of the form {expr}, the corresponding element of TERMNAMES is "{expr}". See topic 'glm'.

Use of coefs() or secoefs()

You can retrieve coefficients and/or their standard errors using coefs() or secoefs().

Other Keywords

Keyword phrase	Default	Meaning
print:F	T	Suppress all output except warning and error messages. Side effect variables are set.
silent:T	F	Suppress all output except error messages. Side effect variables are set.
pvals:T	F	Print P values. Default is T when option pvals is T (see subtopic 'options:"pvals"')
marginal:T	F	Specifies that the elements of the side effect variable SS are computed marginally. When none of the X-variables are aliased, the computed SS are equivalent to SAS Type III SS. See topic

```
                        'glm' for details.  SS will be printed by
                        anova() with no intervening GLM command.

Keyword phrase 'coefs:F' is not legal with regress().

                           Cross references
See also anova().
```

2.295 regresshelp()

Usage:

```
regresshelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
regresshelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
  Subtopics
regresshelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
regresshelp(key:Key), CHARACTER scalar Key
regresshelp(index:T [,scrollback:T])
```

Keywords: general, regression, confidence intervals, glm

```
                                  Usage
regresshelp(Topic1 [, Topic2, ...]) prints help on topics Topic1,
Topic2, ... related to macros in file regress.mac.  The help is taken
from file regress.mac.

regresshelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these macros.

regresshelp(index:T) or simply regresshelp() prints an index of the
topics available using regresshelp().  Alternatively,
help(index:"regress") does the same thing.

regresshelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER
scalar or vector, prints subtopics of topic Topic.  With subtopic:"?", a
list of subtopics is printed.

regresshelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where
Suptopics1 and Subtopics2 are CHARACTER scalars or vectors, prints the
specified subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to regresshelp().  In windowed
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

                               Keyword 'key'
regresshelp(key:key) where key is a quoted string or CHARACTER scalar
lists all topics cross referenced under Key.  regresshelp(key:"?")
prints a list of available cross reference keys for topics in the file.

regresshelp() is implemented as a predefined macro.
```

```
                          Cross references
See help() for information on direct use of help() to retrieve
information from regress.mac.
```

2.296 releigen()

Usage:

```
   releigen(h,e [,maxit:N, nonconvok:T]), h and e symmetric REAL matrices
     with no MISSING values, e positive definite, integer N > 0
```

Keywords: matrix algebra

```
                                   Usage
releigen(H,E) computes an eigenvector/eigenvalue decomposition of the
symmetric matrix H relative to the symmetric positive definite matrix E.
Arguments H and E must both be p by p symmetric matrices with no MISSING
values.

The value returned is structure(values:Vals,vectors:Vecs), where Vals is
the length p vector of relative eigenvalues in decreasing order and Vecs
is a p by p matrix whose columns are the relative eigenvectors.

If H and E are MANOVA hypothesis and error matrices, respectively, you
can use the relative eigenvalues to compute several standard
multivariate hypothesis tests, and the elements of the relative
eigenvectors are the coefficients of the MANOVA canonical variables
associated with the hypothesis.

                          Properties of result
After
  Cmd> releigs <- releigen(H,E) # H and E p by p

  Cmd> v <- releigs$values; u <- releigs$vectors

u is a p by p matrix and v is a vector of length p with elements v[1] >=
v[2] >= ... >= v[p], such that
     H %*% u        = E %*% u %*% dmat(v)
     u' %*% H %*% u = dmat(v)
     u' %*% E %*% u = Ip = p by p identity matrix

On the vector level, this means
     H %*% u[,j] = v[j] * E %*% u[,j], j = 1,...,p
     u[,j]' %*% H %*% u[,j] = v[j], j = 1,..,p
     u[,j]' %*% H %*% u[,k] = 0, j != k
     u[,j]' %*% E %*% u[,j] = 1, j = 1,..,p
     u[,j]' %*% E %*% u[,k] = 0, j != k

dmat(v) is the diagonal matrix with elements of v down the diagonal.

                    Keywords 'maxit' and 'nonconvok'
```

```
See eigen() for information on keyword phrases 'maxit:N' and
'nonconvok:T'.

                            Cross references
See also det(), eigen(), eigenvals(), trideigen() and releigenvals().
```

2.297 releigenvals()

Usage:

```
   releigenvals(h,e [,maxit:N, nonconvok:T]), h and e symmetric REAL
     matrices with no MISSING values, e positive definite, integer N > 0
```

Keywords: matrix algebra

```
                                  Usage
releigenvals(H,E) computes the eigenvalues of the p by p symmetric
matrix h relative to the symmetric positive definite matrix E of the
same size.

The value returned is vector(v1,v2,...,vp), where vj are the relative
eigenvalues with v1 >= v2 >= ... >= vp.  See releigen() for a summary of
the properties of relative eigenvlues.

                    Keywords 'maxit' and 'nonconvok'
See eigen() for information on keyword phrases 'maxit:N' and
'nonconvok:T'.

                            Cross references
See also det(), eigen(), eigenvals() and trideigen().
```

2.298 rename()

Usage:

```
   rename(var, newName), where newName is an undefined variable
```

Keywords: general, variables

```
                                  Usage
rename(Var, Name) where Name is an undefined variable (no value has been
assigned to it) changes the name of Var to Name. Var may be an existing
variable, a constant such as 3.5, "hello", or T, or an expression such
as 'sqrt(20)' or '3*cos(x) + 4*sin(x)'.  When Var a constant or
expression, rename(Var, Name) is equivalent to Name <- Var.  Unless Name
is a CHARACTER scalar (see below), it must not be a function or an
existing variable or macro.  It is an error if variable Name exists,
unless it is the same variable as Var as in rename(x,x).

rename(Var, nameVar) where nameVar is a quoted string or scalar
```

```
CHARACTER variable is equivalent to rename(Var, <<nameVar>>), that is
the new name is the value of nameVar.  For example, rename(x, y) and
rename(x, "y") are equivalent.  The value of nameVar must not be the
name of any existing variable, macro or function and must be a legal
name (see 'syntax').  "NULL", "T", and "F" are not legal names.

It is an error if Var is a locked variable; see lockvars(),
unlockvars(), islock() and variables:"locked_variables".

Var must not be a function or a "special" variable such as CLIPBOARD,
SELECTION or GRAPHWINDOWS.  See topics 'CLIPBOARD', 'GRAPHWINDOWS' and
'graph_assign'.

                              Alternative
You can achieve the same effect as rename(Var,Name) by
  Cmd> Name <- Var; delete(Var)

However, if Var is not a constant or expression, doing it this way
entails the temporary existence in memory of two copies of Var.  If Var
is large, there may not be enough room in your workspace.  The use of
rename() avoids this problem.  For this reason, in a macro that uses
another macro to compute a value, it may sometimes be helpful to use
rename() instead of assignment.

                               Examples
Examples:
  Cmd> rename(PI, pi) # or rename(PI,"pi"), change name of PI to pi

  Cmd> e <- 10; rename(E, e) # is an error since e exists

  Cmd> rename(sqrt(2),sqrt2) # same as sqrt2 <- sqrt(2)

The last example would be an error if sqrt2 is already a variable.

                            Cross references
See also nameof(), compnames(), varnames()
```

2.299 rep()

Usage:

```
   rep(x, n) or rep(x, repFac), where x is REAL, LOGICAL, or CHARACTER, and
     n is an integer > 0 or repFac is a vector of integers >= 0
```

Keywords: combining variables, variables

```
                                 Usage
rep(X, n), where n is a positive integer, replicates the values in X n
times to form a vector of length n*length(X).  It is equivalent to
vector(X,X,X,...,X), where there are n repetitions of X.  X must be a
REAL, LOGICAL or CHARACTER vector, matrix, or array.
```

```
rep(X, 0) is legal an has value NULL.

rep(X, Repfac), where Repfac is a vector of integers with length(Repfac)
= length(X) and Repfac[i] >= 0, replicates the j-th element of vector(X)
Repfac[j] times.  The result has length length(X)*sum(Repfac).  If every
element of Repfac is 0, the result is NULL.

                              Examples
Examples:
  Cmd> rep(vector(1,3,5),4)  # same as vector(1,3,5,1,3,5,1,3,5,1,3,5)
   (1)           1           3           5           1           3
   (6)           5           1           3           5           1
  (11)           3           5

  Cmd> rep(vector(7,6,5),vector(3,0,2)) # same as vector(7,7,7,5,5)
  (1)           7           7           7           5           5

  Cmd> a <- factor(rep(run(5),4)); b <- factor(rep(run(4),rep(5,4)))

  Cmd> print(a,b) # factors for 4 by 5 factorial design with no reps
  a:
   (1)           1           2           3           4           5
   (6)           1           2           3           4           5
  (11)           1           2           3           4           5
  (16)           1           2           3           4           5
  b:
   (1)           1           1           1           1           1
   (6)           2           2           2           2           2
  (11)           3           3           3           3           3
  (16)           4           4           4           4           4

  Cmd> rep(run(5), vector(1,2,3))# ERROR! (args have different lengths)
  ERROR: in rep(x,m), m must be scalar or a vector the same length as x

                                Warning
WARNING: Because rep() is a function you cannot use the name 'rep' as a
variable name, say for a replication factor in a design.  Use 'Rep' or
'reps' instead.

                            Cross references
See also topics vector(), run(), 'models', 'glm'.
```

2.300 replacestr()

Usage:

```
replacestr(source,old,new) replace first instance of old by new
  in source, where all three are Character scalars
```

Keywords: CHARACTER variables

```
replacestr(source,old,new) works on three CHARACTER scalar
```

```
variables.  It replaces in source the first instance of the string
in old by the string in new.  If old does not appear in source,
then source itself is returned.
```

2.301 restore()

Usage:

```
restore(FileName [ ,delete:F, list:T, history:F, options:F,\
  foreignok:T])
```

Keywords: files, general

```
                                  Usage
restore(FileName) restores the workspace or variables previously put in
file FileName by save() or asciisave(), where FileName is a quoted
string or CHARACTER variable.  Unless option 'restoredel' has been set
to False (see subtopic 'options:"restoredel"'), all current variables
are deleted before restoration.  If a history of commands was saved by
using history:T on save() or asciisave(), it replaces the corrent
history of recent commands.

In a version with windows, if FileName is "" you will be prompted for
the name of the file.  Restore Workspace on the File menu in versions
with windows is equivalent to 'restore("")'.

Restoring a workspace may destroy any existing locked variables.  See
topic 'variables:"locked_variables"'.

                          Variables not restored
Although variables HOME, DATAFILE, DATAPATHS, MACROFILES, HELPFILES and
HELPINDICES are saved by save() and asciisave(), they are not restored
unless they do not currently exist.

These variables are used to find files containing macros, data and help
information.  See topics adddatapath(), addmacrofile(), getdata(),
getmacros(), 'macros' and 'DATAPATHS'.

Any of these variables that are not restored become components of
structure SAVEDNAMES with components HOME, DATAFILE, DATAPATHS, ....

If you want to restore DATAPATHS, say, you must follow restore() by

  Cmd> DATAPATHS <- SAVEDNAMES$DATAPATHS

or you can use macro restorenames():

  Cmd> restorenames(DATAPATHS)

To restore all these variables after restore(), use

  Cmd> restorenames()
```

In addition, variable VERSION is not restored and is not saved in SAVEDNAMES. Whether or not VERSION exists before restore() is run, restore() sets it to a value appropriate to the MacAnova version being run.

These steps help ensure that the values of these variables pertains to the computer and version you are actually using rather than to the computer and version used to create the workspace file. See topics 'files', 'launching'.

Keyword 'delete'

restore(FileName, delete:F) does not delete existing variables before restoring. However, a variable or macro with the same name as a variable or macro in the file is replaced by the one in the file. If option 'restoredel' has been set to False, it may be overridden by delete:T.

Keyword 'history'

restore(FileName, history:F) suppresses restoring of any history of recent commands that may have been saved. Use this if you want to preserve the history of commands immediately preceding the restore() command. This is not available in versions that do not save such a history. See sethistory() and gethistory(). Note: a command history is not saved on a save of selected variables.

Keyword 'options'

restore(FileName, options:F) suppresses restoring the values of options in effect when the workspace file was created. Use this if you want to preserve options currently in effect. See topics 'options', setoptions() and getoptions().

Note: option values are not saved on a save of selected variables.

Keyword 'list'

restore(FileName [,delete:F], list:T) lists information on variables as they are restored.

Restoring foreign binary files

It is normally an error when the file is a binary workspace file from an incompatible computer, for example, restoring a Macintosh save file on Windows.

restore(FileName, foreignok:T ...) makes it permissible to try to restore a binary workspace file from an incompatible computer. When necessary and possible, bytes in the internal representation of numbers are permuted so as to give them correct values. This is impossible if either the number of bytes in either a double precision number or a long integer differ between the machines.

If you are planning to use a save workspace on a different type of computer, it is safer to save it using asciisave().

Restoring GRAPHWINDOWS and options

Unless 'graphwind:F' was used when the workspace file was created, special structure GRAPHWINDOWS is restored. In windowed versions the windows corresponding to its components are redrawn (see topic 'GRAPHWINDOWS'). If the number of components saved is greater than the number allowed in the version you are using, extra components are ignored. If the number is fewer, missing components are assumed to be NULL.

Unless options:F is an argument to restore() or was an argument to save() or asciisave() when the workspace file was created, values for options such as 'nsig', 'format', and 'seeds' will be restored to the previous values at the time the save was done (see 'options').

There are a couple of exceptions.
'prompt' is not restored when input is from a 'batch' file; see batch()
'matchdelay' is not restored when restoring a workspace saved on an incompatible system.

Restoring private GLM information

If all:T was used when the file was created, then private information related to the most recent GLM command will be restored. Without this information certain commands such as secoefs() do not work until a GLM command has been executed. See topic 'glm'.

Restoring old workspace files

Workspace files created by earlier versions of MacAnova for the same computer can normally be restored correctly except that binary files created by MacAnova2.4x on a Macintosh cannot be restored.

Cross references

See also topics asciisave(), save(), restorenames(), 'options', 'files'.

2.302 restorenames()

Usage:

```
restorenames()
restorenames(name1 [,name2, ...]), name1, name2, ... are any of these
  variable names: HOME, DATAFILE, DATAPATHS, MACROFILES, HELPFILES,
  HELPINDICES
```

Keywords: files, general, structures

Usage

restore() does not overwrite certain variables containing information on the location of data, macro and help files. These variables are HOME, DATAFILE, DATAPATHS, MACROFILES, HELPFILES and HELPINDICES. Instead, it creates a structure SAVEDNAMES containing the values of these variables in the workspace file being restored. restorenames() is provided to make it easy to overwrite any or all of these variables

```
after restore().

restorenames() restores all the variables in SAVEDNAMES.

restorenames(name1 [, name2, ...]) restores just the named variables.
name1, name2, ... must be names of components of SAVEDNAMES, normally
HOME, DATAFILE, DATAPATHS, MACROFILES, HELPFILES and HELPINDICES.

                                Examples
Example:
  Cmd> restore("workspace.030320")

  Cmd> restorenames(MACROFILES, HELPFILES)

                             Cross references
See also restore(), save(), getdata(), getmacros(), addmacrofile(),
adddatapath(), addhelpfile(), 'workspace'.
```

2.303 return

Usage:

```
return(x), x a variable or constant, only in a macro
return and return() are equivalent to return(NULL)
```

Keywords: control, syntax, macros

```
                          Description and usage
Syntax element 'return' is used to leave a macro immediately, skipping
any following expressions or commands.  It may be used only in the text
of a macro.

In a macro, when 'return(Value)' is executed, the macro is immediately
terminated and Value is returned as the macro value.  Value can be any
constant, variable or expression.  'return(Value)' must be followed by
';' or '}' or be at the end of a line.

In a macro, 'return()' and 'return', with no parentheses, are equivalent
to 'return(NULL)'.

                         Where 'return' is used
'return(Value), 'return()' and 'return' are typically the last command
in the compound command (see topic 'syntax') that follows 'if(...)',
'elseif(...)' or 'else' (see topic 'if'), or as the last command in a
macro.

'return' may not be used in an evaluated string unless it is in a macro
invoked in that evaluated string.  This is true even if the string being
evaluated is within a macro.  See evaluate().

                                Example
Example:
```

```
  Cmd> checkit <- macro("if($1==1){return(\"one\")}
       if($1 == 2){return(\"two\")};return(\"not one or two\")")

  Cmd> reply <- checkit(1); reply
  (1) "one"

  Cmd> reply <- checkit(3); reply
  (1) "not one or two"
```

The final 'return(...)' could be replaced simply by "not one or two", since the default return value is the last expression or variable in the macro.

Cross references

See also macro(), 'macros', 'macro_syntax', 'break', 'next'

2.304 reverse()

Usage:

```
reverse(x), x a vector or matrix.
```

Keywords: time series

Usage and examples

reverse(x) reverses the order of the rows of x, where x is a REAL, LOGICAL or CHARACTER vector or matrix.

The result has the same size and type as x. When isvector(x) is True, the result is a pure vector (ndims(x) = 1). Otherwise, the result has 2 dimensions.

Examples:

```
  Cmd> reverse(vector(1,3,2,4,7))
  (1)          7          4          2          3          1

  Cmd> reverse(matrix(run(6),3))
  (1,1)          3          6
  (2,1)          2          5
  (3,1)          1          4

  Cmd> reverse(matrix(vector("A","B","C","D")))
  (1) "D"
  (2) "C"
  (3) "B"
  (4) "A"
```

2.305 rft()

Usage:
```
rft(rx [,divbyT:T]), rx a REAL matrix
```

Keywords: time series, complex arithmetic

```
                                Usage
rft(rx) where rx is a REAL vector or matrix, computes the Hermitian form
of the complex discrete Fourier transform of each column of rx,
considered as a real series.

Any MISSING values in rx are replaced by 0 in computing the result and a
warning message is printed.

rft(rx,divbyt:T) does the same except the result is divided by the
number of rows of rx.

                            Inverse transform
hft(hconj(hx),divbyt:T) is the inverse of rft() in the sense that rx and
hft(hconj(rft(rx)),divbyt:T) are equal except for rounding error.

                         Limitation on length
The largest prime factor of nrows(rx) must not exceed 29.  You can use
primefactors() to find the maximum factor of nrows(rx) and goodfactors()
to find a length >= nrows(rx) which has no prime factors > 29.  In
addition, the product of all the "unpaired" prime factors can't be too
large.  For example N = 3*5*7*11*13*17*M^2 = 255255*M^2, where M is an
integer, breaks the algorithm and hence is not allowed.

                            Cross references
See also cft(), hft(), hconj(), primefactors(), goodfactors().

See topic 'complex' for discussion of complex matrices in MacAnova.

See subtopic 'matrices:"complex_matrices" for a list of macros for
working with complex matrices.
```

2.306 rnorm()

Usage:
```
rnorm(n), n a positive integer
```

Keywords: random numbers

```
                                Usage
rnorm(N) generates a vector of N pseudo-random normals with mean 0 and
variance 1.  N must be a positive integer.

If the random number generator has not been initialized by setseeds(),
```

```
setoptions() or previous use of rbin(), rnorm(), rpoi() or runi(), the
generator's "seeds" will be initialized automatically using the current
time and date, and their values will be printed out.

To generate a pseudo random sample from the normal distribution with
mean mu and standard deviation sigma do the following.
  Cmd> y <- mu + sigma*rnorm(N) # sigma > 0 and mu REAL scalars

                            Cross references
See also topics setseeds(), getseeds(), setoptions(), rbin(), rpoi(),
runi(), cumnor() and invnor(), subtopic 'options:"seeds"'.
```

2.307 robust()

Usage:

```
   robust([Model] [,print:F or silent:T, trunc:c, maxiter:m, epsilon:eps, \
     fstats:T, pvals:T, marginal:T]), Model a CHARACTER scalar,
     c and eps positive REAL scalars, m a positive integer.
```

Keywords: glm, anova, regression

```
                                    Usage
robust(Model) computes a robust linear fit of the model specified in the
quoted string or CHARACTER variable Model.  An approximate analysis of
variance (ANOVA) table is printed, as well as a robust estimate of
sigma.  In the case of normal errors with possible outliers arising from
contamination, the square of the estimated sigma is approximately
unbiased for the variance of the uncontaminated errors.

See topic 'models' for information on specifying Model.

When Model is omitted (robust() or robust(,...), the most recent GLM
model is assumed, or, if there have not been previous GLM commands, the
model specified in variable STRMODEL, if any.

                        Side effect variables created
Side effect variables created by robust() are DF, SS, HII, RESIDUALS,
and WTDRESIDUALS.  WTDRESIDUALS is not really weighted residuals;
instead it is set to the vector of modified residuals
sigmahat*psi(residuals/sigmahat).  See below for details.

Inference based on the ANOVA table should be used with caution.
Especially when there are cases with high leverage (large value of HII),
or when the ratio of model degrees of freedom to error degrees of
freedom is too large, the approximation can be misleading.  Interpret
with similar caution the results of secoefs() and contrast() after
robust().

                Keyword 'trunc', 'maxiter' and 'epsilon'
Keyword phrase  Default  Meaning
  trunc:c         .75    Positive REAL c is used as a "truncation point"
```

		in the fitting algorithm. See discussion below. The larger the value, the less "robustifying."
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting
epsilon:eps	1e-6	Small positive REAL specifying relative error in objective function required to end iteration

Other keywords

See topic 'glm_keys' for information on keyword phrases 'print:F', 'silent:T', 'fstats:T', 'pvals:T' and 'marginal:T'. Any P values printed by fstats:T and pvals:T are only approximate. Keyword phrase 'coefs:F' is not legal.

The default values for fstats and pvals can be changed by setoptions(fstats:T) and/or setoptions(pvals:T). See setoptions(), options.

Algorithm used

The algorithm used is based on Sec. 7.8 of "Robust Statistics" (Wiley 1981) and Sec. 14 of "Robust Statistical Procedures" (SIAM 1977), both by Peter J. Huber. Coefficients and scale sigma are estimated by minimizing a certain function of sigma and the residuals r = y - fit (Huber 1981, eq. 7.7.9 and 7.7.14).

Effectively the algorithm deemphasizes cases whose apparent residual r[i] satisfies abs(r[i]) > c * sigma, where c is a constant "truncation point". The default value of c is .75 but it can be set by keyword 'trunc'.

Contents of WTDRESIDUALS

After running robust(), WTDRESIDUALS contains the modified residuals rmod = sigmahat*psi(rhat/sigmahat), where sigmahat is the estimated scale, rhat is the vector of estimated residuals, and psi(z) = z when abs(z) < c, psi(z) = c for z >=c, psi(z) = -c, z <= -c.

Contents of ANOVA table

Following a suggestion of Huber, the ANOVA table is computed from pseudo-data obtained as fithat + K*rmod, where fithat = y - rhat is the vector of estimated predicted values and K is a constant (see p. 40 of Huber 1977). K = (1 + (k/n)(1-mu)/mu)/mu, where n is the sample size, k is the number of model degrees of freedom, and mu = m/n where m is the number of non-truncated rhat[i], that is, which satisfy abs(rhat[i]/sigmathat) < c. The robust estimate of sigma is sqrt(mse/beta)/K, where mse is the error mean square from the ANOVA table and beta is a constant computable by beta = cumchi(c^2,3) + 2*c^2*(1 - cumnor(c)) = E[psi(z^2)] for N(0,1) z.

If you want ANOVA results with a different order of terms, you do not need to redo robust(); you can use anova() with the pseudo-data in the preceding paragraph as dependent variable. You can determine m needed

```
to compute K as the number of cases for which WTDRESIDUALS and RESIDUALS
are equal except for rounding error.
```

2.308 rotate()

Usage:

```
   rotate(x, k), x a vector or matrix, k an integer.
```

Keywords: time series, combining variables

```
                                  Usage
rotate(x, k) "rotates" by k rows each column of the REAL, LOGICAL or
CHARACTER vector or matrix x.  When k > 0, rows pushed down off the end
are shifted to the start and when k < 0, rows pushed up before the start
are moved to the end.  k must be a single integer and is interpreted
modulo nrows(x).  For example, rotate(x,k) and rotate(x,k %% nrows(x))
are equivalent.

More explicitly, for -m < k < m, where m = nrows(x), rotate() moves rows
as follows:
  0 <= k <= m-1 (down column shift)   -(m-1) <= k <= 0 (up column shift)
 Row j -> Row j+k   for j=1,...,m-k   Row j -> Row j+k+m for j=1,...,-k
 Row j -> Row j+k-m for j=m-k+1,...,m Row j -> Row j+k for j=-k+1,...,m

The result has the same size and type as x.  When isvector(x) is True,
the result is a pure vector (ndims(x) = 1).  Otherwise, the result has 2
dimensions.

                                Examples
Examples:
  Cmd> rotate(vector(1,3,2,6,5,4),2)
  (1)           5           4           1           3           2
  (6)           6

  Cmd> rotate(matrix(vector(1,3,2,7, 6,5,4,8),4), -2)
  (1,1)           2           4
  (2,1)           7           8
  (3,1)           1           6
  (4,1)           3           5

  Cmd> rotate(vector("A","B","C","D"),-1)
  (1) "B"
  (2) "C"
  (3) "D"
  (4) "A"

                                Caution
Caution: Do not confuse rotate() with rotation() which does rotation of
factor loadings.
```

2.309 rotation()

Usage:

```
rotation(loadings [, method:Method, kaiser:T, reorder:T,
  lambda:lam,verbose:T]), where loadings is a REAL matrix, lam >= 0 is a
  real scalar and Method is a quoted string or CHARACTER scalar
```

Keywords: multivariate analysis

```
                                  Usage
L <- rotation(Loadings) or L <- rotation(Loadings, method:"varimax")
sets L to a matrix derived by varimax "rotation" from Loadings.  That
is, an orthogonal matrix R is found so that L = Loadings %*% R maximizes
a certain criterion.

Loadings must be a REAL matrix with no MISSING values and with
nrows(Loadings) >= ncols(Loadings).  The result L is a REAL matrix with
the same dimensions as Loadings.

rotation(Loadings, kaiser:T [,method:"varimax"]) does the same except
Kaiser normalization is used.  That is the rows of loadings are rescaled
to have norm 1 before rotation and then unscaled after rotation.

rotation(Loadings, method:"quartimax" [, kaiser:T]) does the same except
the quartimax criterion is maximized.

rotation(Loadings, method:"equimax" [, kaiser:T]) does the same except
the equimax criterion is maximized.

rotation(Loadings, method:"orthomax", lambda:lam [, kaiser:T]) where lam
>= is a REAL scalar does the same except the orthomax crition with
parameter lam is maximized.  lam = 1, lam = 0 and lam =
ncols(Loadings)/2 correspond to varimax, quartimax and equimax rotation,
respectively.

rotation(Loadings, verbose:T [,method:Method ...]) prints the value of
the criterion before and after rotation.

rotation(Loadings, reorder:T [,method:Method ...]) does the same except
that after rotation each column is multiplied by +1 or -1 so that the
column sum is positive, and the columns are reordered in decreasing
order of the column sums of squares.

                        Use with factor analysis
rotation() is designed to be used to rotate a factor analysis matrix of
loadings so as to achieve "simple structure."

It is usual to rotate not the loading matrix itself, but the loading
matrix scaled so that the row sums of squared loadings are constant.
The rotated matrix is then rescaled to restore the original row sums of
squares.  This is sometimes called Kaiser normalization and is
accomplished automaticall by using 'kaiser:T' as an argument.

The rotated matrix has the form rotated_L = L %*% A, where A is m by m.
```

```
A can be recovered as
      A <- solve(L' %*% L, L' %*% rotated_L)

                    Keywords 'epsilon' and 'maxiter'
The algorithm used is iterative, using a default convergence criterion
of epsilon = .00001, and performing a maximum of 100 iterations.  These
values can be modified by including keyword phrases 'epsilon:value'
and/or 'maxiter:n', where value is a small positive number and n is a
positive integer.

                                  Caution
Caution: Do not confuse rotation() with rotate() which shifts the rows
of its first argument up or down, wrapping around the end.
```

2.310 round()

Usage:

```
round(x [, ndec]), x REAL or a structure with REAL components, ndec an
  integer
```

Keywords: transformations

```
                                  Usage
round(x) rounds the elements of the REAL variable x to the nearest
integer, producing a vector, matrix, or array with the same shape as x.

round(x,n) where n is an integer is equivalent to 10^(-n)*round(x*10^n).
If n > 0, this rounds to n decimal places.  If n < 0, this rounds to the
nearest multiple of 10^abs(n).  round(x,0) is equivalent to round(x).

                           Structure argument
If x is a structure, so is round(x) or round(x,n).  If xi is the i-th
component of x, the i-th component of round(x) or round(x,n) is
round(xi) or round(xi,n).

                                  Example
Example: round(3141.593,2) is 3141.59 and round(3141.593,-2) is 3100,
the nearest multiple of 100 = 10^2.

                           CHARACTER argument
round(x, p) can also be used when x is a CHARACTER variable and p, if
present, is a quoted string or CHARACTER scalar or REAL scalar.  The
result is a CHARACTER variable of the same shape as x describing the
transformation.  For example, both round(vector("X1","X2"),3) and
round(vector("X1","X2"),"3") return vector("round(X1,3)","round(X2,3)").
Any element of x that is "" or starts with '@', '(', '[', '{', '<', '/'
or '\' is not modified.  This can be useful for creating labels for a
transformed variable.

                           Cross references
See also topics floor(), ceiling(), 'structures', 'labels'.
```

2.311 rowplot()

Usage:
```
rowplot(x [, graphics keyword phrases]), x a REAL matrix
```

Keywords: plotting

```
                              Usage and example
rowplot(x) makes an "interaction" plot of the data in the matrix x.  The
plotting positions are the column numbers and the values in x.  Points
within each row are joined by lines.  Any keywords useable in chplot()
may follow x.  rowplot() is implemented as a macro.

If option 'dumbplot' has been set False (see subtopic
'options:"dumbplot"'), the plot will be a low resolution plot unless
'dumb:F' is an argument.

You can use all the usual graphics keywords, including 'title', 'xlab',
'ylab', and 'file'.  See topics 'graphs', 'graph_keys', 'graph_border'
and 'graph_ticks'.

Example:
  Cmd> rowplot(run(20)^(.2*run(5)'),\
        title:"X^vector(.2, X^.4, X^.6, X^.8, X)'")

                              Cross references
See also topic colplot().
```

2.312 rpoi()

Usage:
```
rpoi(n,lambda), n positive integer, lambda scalar >= 0 or non-negative
  vector of length n.
```

Keywords: random numbers

```
                                    Usage
rpoi(N,lambda) generates a vector of N independent Poisson pseudo-random
variables with mean lambda.  N must be a positive integer.

lambda must be a REAL scalar >= 0 or a REAL vector of length N with
lambda[i] >= 0.  If lambda is a scalar, it is used for every element of
the result.  Otherwise, element of the result will be Poisson with mean
lambda[i].

                         Generating negative binomial
When used together with invgamma(), rpoi() can be used to generate
```

```
pseudo-random negative binomial random variables.

  Cmd> y <- rpoi(N, m*invgamma(runi(N),alpha)) # m > 0, alpha > 0

will generate negative binomial random variables with shape or index
alpha and mean alpha*m, with probabilities
 p[y=x]={(1-m)^alpha}*alpha*(alpha+1)...*(alpha+x-1)*m^x/x!, x=0,1,... .

                              Initializing
If the random number generator has not been initialized by setseeds(),
setoptions() or previous use of rbin(), rnorm(), rpoi() or runi(), the
generator's "seeds" will be initialized automatically using the current
time and date, and their values will be printed out.

                                Reference
The algorithm is based on C. D. Kemp and A. W. Kemp, Appl Statist 40
(1991) 143-158.

                            Cross references
See also setseeds(), getseeds(), setoptions(), runi(), rnorm(), rbin(),
invgamma(), cumpoi(), subtopic 'options:"seeds"'.
```

2.313 rsample()

Usage:

```
   y <- rsample(x,n) or y <- rsample(x,n,T), REAL vector or matrix x,
     positive integer n
   y <- rsample(x,n,F) requires n <= nrows(x)
```

Keywords: random numbers

```
                                 Purpose
rsample() is a macro you can use to select a random sample with or
without replacement from the rows of a REAL matrix or vector.

                                  Usage
y <- rsample(x, n) and rsample(x, n, T) both select a random sample of
size n from the rows of REAL matrix x.  n is a positive integer.
Sampling is *with* replacement.

y <- rsample(x, n, F) does the same, except that sampling is without
replacement.  It is an error if n > nrows(x).

In both usages, y will be a REAL vector of length n or a REAL n by
ncols(x) matrix.

                                 Example
The following selects a random sample of size 5 drawn without
replacement from {1,2,...,20}:

  Cmd> y <- rsample(run(20),5,F)
```

```
  Cmd> y
  (1)          11          10           8           3          18

                              Cross references
See also runi(), rnorm().
```

2.314 rsolve()

Usage:

```
   rsolve(A, B [,quiet:T]), A a square REAL matrix, B a REAL matrix,
     nrows(A) = ncols(B).
```

Keywords: linear algebra

```
                                    Usage
rsolve(a, b) computes the solution x to the system of linear equations
x a = b, where a is a REAL square matrix and b is a REAL matrix with
ncols(b) = nrows(a).  rsolve(a, b) produces the same result as
solve(a',b')', and is mathematically but not computationally equivalent
to b %*% solve(a).

If a is singular, an informative message is printed and the operation
aborts.

MISSING values are not allowed in a or b.

When a has labels, the row and column labels of rsolve(a,b) are the row
labels of b and respectively.  If b does not have labels, the rowlabels
of solve(a,b) are rep("a", nrows(b).  See topic 'labels'.

Expression b %/% a is equivalent to rsolve(a, b).

                               Keyword 'quiet'
Occasionally, "overflow" occurs during the computation.  Any values in
the result that are too large to be represented are replaced by MISSING
and a WARNING message is printed.  You can suppress the message by
including 'quiet:T' as an argument.  If you do, you should use
anymissing() to check the result for the presence of MISSING elements.

                              Cross references
See also solve(), swp().
```

2.315 run()

Usage:

```
   run(first,last,incr) or run(first,last) or run(last)
```

Keywords: combining variables, variables

```
                                  Usage
run(First,Last,Incr) will generate a sequence of numbers from First to
Last with step size given in Incr.  The i-th number is computed as
First+(i-1)*Incr (unless it is very close to 0 or Last; see below).  It
is an error if Incr is 0, unless First = Last.

run(First,Last) is the same as run(First,Last,1) if Last > First, and to
run(First,Last,-1) if Last < First.

run(n) does the same as run(1,n) if n is an integer.  This is the most
common usage.

run(vector(First,Last,Incr)) is the same as run(First,Last,Incr).  A
vector argument must be of length 3, so run(vector(First,Last)) is
illegal.

                          Values computed exactly
When Incr is (Last-First)/n, where n is an integer, there will be n+1
values, with the (n+1)-th being exactly Last, even if First+n*Incr is
slightly less or greater than Last because of rounding error.

Similarly, if 0 is between First and Last and Incr is -First/n, the
(n+1)-th value will be exactly 0 even if First+n*Incr is not exactly 0
because of rounding error.  In both situations, the value is rounded to
the target value (Last or 0) if abs((First+n*Incr-target)) <
1e-15*abs(Last-First).
```

2.316 runi()

Usage:

```
runi(n), n a positive integer
```

Keywords: random numbers

```
                                  Usage
runi(count) generates a vector of count pseudo-random uniforms on the
interval 0 to 1.

If the random number generator has not been initialized by setseeds(),
setoptions() or previous use of rbin(), rnorm(), rpoi() or runi(), the
generator's "seeds" will be initialized automatically using the current
time and date, and their values will be printed out.

You can generate uniform random variables on the interval (a,b), a < b
by
  Cmd> x <- a + (b - a)*runi(n)

                       Discrete uniform generation
You can generate the discrete uniform distribution on the integers 1, 2,
```

```
..., m by
  Cmd> x <- ceiling(m*runi(n))

This is helpful when sampling with replacement from the rows of a data
vector of matrix.

                  Nonuniform random variable generation
When Q() is a macro or function computing the quantile function (inverse
cumulative distribution function) of a continuous random variable
(invnor() or invchi()) for example), Q(runi(n) [,parameters]) generates
a random sample from that distribution.

  Cmd> invstu(runi(5),3) # Student's t on 3 d.f.
  (1)     0.43734     0.34297    0.054439  -0.0017229    -0.32894

  Cmd> invF(runi(5),5,30) # F on 5 and 30 d.f.
  (1)     0.45207      2.2247     0.52716     0.29218       1.506

The functions that you can use directly this way are invbeta(),
invchi(), invF(), invgamma(), invnor(), and invstu().  In principle you
could also use invdunnett() and invstudrng(), but that is not advisable
because they are so computationally intensive.

                            Cross references
See also topics setseeds(), getseeds(), setoptions(), rnorm(), rbin(),
rpoi(), subtopic 'options:"seeds"'
```

2.317 samplesize()

Usage:

```
samplesize(noncen,ngrp,alpha,Pwr[,design:"rbd"][,maxn:N]), noncen > 0,
  0 < alpha < 1, 0 < Pwr < 1, integers ngrp > 0, N > 0
```

Keywords: probabilities, glm, anova

```
                                    Usage
samplesize(noncen,ngrp,alpha,Pwr) computes the group sample size (number
of replicates) required in a balanced one-way analysis of variance
(completely randomized design with equal group sizes) of ngrp groups at
significance level alpha to achieve power Pwr.

Argument noncen is the noncentrality parameter and is interpreted as
sum(alphaˆ2)/sigmaˆ2, where alpha is a vector of treatment effects with
sum(alpha) = 0 and sigma is the error standard deviation.

If the required sample size exceeds 256, 256 is returned.

samplesize(muˆ2/sigmaˆ2,1,alpha,Pwr) returns the sample size required to
achieve power Pwr in a single-sample two-tail t-test of H0: mu = 0 when
the standard deviation is sigma.
```

```
                    Keywords 'design' and 'maxn'
samplesize(noncen,ngrp,alpha,Pwr,design:"rbd") computes the number of
blocks required to achieve power Pwr in a randomized complete block
design with ngrp >= 2 treatments.

samplesize(noncen,ngrp,alpha,Pwr [,design:"rbd"], maxn:N), N > 0 an
integer, does the same, except if the required samples size exceeds N, N
is returned.

                          Cross references
See also power() and power2().
```

2.318 save()

Usage:

```
save(FileName [,all:T, v335:T, v406:T, nulls:F, options:F,\
  history:T, ascii:T])
save() repeats previous save() or asciisave()
```

Keywords: files, general, variables, null variables

```
                                  Usage
save(FileName) saves the MacAnova "workspace", that is, all the current
variables and option values, in a file with name given in the quoted
string or CHARACTER variable FileName.  The file will be in a binary
format that is specific to the type of computer.  The workspace can be
recovered later by restore().  See topics 'files', 'workspace',
restore().

save(FileName,ascii:T) is equivalent to asciisave(FileName), that is,
the file written will be an ASCII text file instead of a binary file.
This option can be used together with others described below.

In a version with windows, FileName can be "", in which case you will
be prompted for the file name.

                          Resaving in same file
save(), with no file name, saves the workspace in the same file as
specified in a previous asciisave() or save().  If the previous save was
ASCII, so will be the current save.  Moreover, if the previous save()
specified an obsolete format (see below), the same format will be used,
unless explicitly changed.  If there was no previous save() or
asciisave(), omitting the file name is an error.

                                Partial save
save(FileName, var1, var2, ....) does a partial save, saving only
variables or macros var1, var2, ... on the file.  Any variable to be
saved that is specified in keyword form (example: save(FileName,
residuals:RESIDUALS)), will be restored having the keyword name.  Unless
the variables are saved using an obsolete format, when a partial
save is restored, other variables are not affected.  If an obsolete
```

format is used, other variables are normally deleted when the file is restored, unless keyword phrase delete:F is used on restore().

Saving GRAPHWINDOWS

On all versions, a complete save, but not a partial save, normally saves the special structure GRAPHWINDOWS (see topic 'GRAPHWINDOWS') as part of the workspace. When GRAPHWINDOWS is restored, windows corresponding to any GRAPH components are redrawn.

save(FileName, graphwind:F) saves the workspace without including GRAPHWINDOWS.

Saving history of commands

A complete save (but not a partial save) normally saves the history. When restore() recovers the information in the file, the saved history list replaces the command history unless restore() keyword phrase 'delete:F' is used.

save(FileName,history:F) saves the workspace without saving the command history.

save(FileName,history:T) saves the workspace together with the command history, even when the value of option 'savehistry' is False (see topic 'options':"savehistry"'). 'history:T' is illegal on a partial save. See sethistory() and gethistory().

Saving options

save(FileName,options:F) suppresses saving the current values of options, which normally occurs. When options:F is not used, restore() resets options to the saved values, including random number seeds. See setoptions(), getoptions().

You can use any of history:F, options:F and graphwind:F together.

Save GLM private information

save(FileName,all:T) saves, in addition to the workspace, a variety of information computed by the last GLM command. When this information is recovered by restore(), it becomes possible to use functions such as secoefs() and modelinfo() to get information on the last GLM analysis before the save.

This option is really useful only when you have just completed a GLM (generalized linear or linear model) computation (regress(), anova(), poisson(), etc.) that took a long time to compute since it is usually sufficient just to repeate the GLM command when you restore the workspace. If the model was complex and had many cases, 'all:T' can greatly increase the size of the workspace file.

Keyword 'nulls'

save(FileName, nulls:F [, var ...]) does not save NULL variables. See topic 'NULL'.

Other information saved

save() also saves the current time and date. These are reported by restore() when the workspace or variables are restored.

In addition, save() saves information about the computer MacAnova is running on and the compiler it was compiled with, plus information about the internal representation of linear models. This information is used by restore() to determine if it is safe to restore variables. If you are planning to restore the file on another computer or different version of MacAnova, use asciisave().

Difference from asciisave()

save() differs from asciisave() in that asciisave() saves the information in the form of a "text" file, more or less human readable, that can be transferred between different types of computers. Files created by asciisave() are often bigger than the corresponding file created by save().

Why use save()

save() and asciisave() are useful when a session must be interrupted and you don't want to lose what you have done. On an unstable system or when using MacAnova remotely over a noisy phone line, a useful insurance measure is to save your current variables every few minutes. The following creates a macro that makes this easy:

```
  Cmd> snapshot <- macro("save(\"snapshot.sav\",all:T)")
```

Then simply typing 'snapshot()' saves a copy of your workspace, options, and internal variables related to GLMs on file snapshot.sav. They can be restored by 'restore("snapshot.sav")'.

Examples

Examples:

```
  Cmd> save("chckpnt.bin") # save entire workspace, but not GLM stuff
  Cmd> save("chckpnt.bin",options:F,history:)#no history or options
  Cmd> save("saveFile",all:T) # save everything, including GLM stuff
  Cmd> asciisave("saveFile",x,y,residuals:RESIDUALS, v31:T)
  Cmd> save("saveFile",x,y,residuals:RESIDUALS, v31:T, ascii:T)
```

The last two are identical and save only x, y, and RESIDUALS on ASCII file saveFile readable by MacAnova 3.1. When restored, RESIDUALS will be recreated with name 'residuals'.

Windowed versions

In windowed versions, Save Workspace on the File menu (Command+K or Ctrl+K) is equivalent to 'save()', except that if there hasn't been a previous save() or asciisave(), you are prompted for a file name using the usual file navigation dialog box.

When you Quit from MacAnova, you are normally asked if you want to save the workspace (only in windowed versions).

Compatibility of workspace files of earlier versions

There have been minor and major changes in workspace file format as MacAnova has evolved. Generally when a change is made, older versions

of MacAnova are not able to restore files created using the new format. For instance, workspace files written by version 3.36 or later which contain NULL variables or variables with MISSING values cannot be restored by earlier versions.

If it is important for an earlier version of MacAnova to be able to restore a workspace file created on the latest version, you can specify one of the obsolete formats by using keywords 'v24', 'v31', 'v335' or 'v406'.

save(FileName,v24:T [, var ...]) or save(FileName,old:T [, var ...]) saves in the format recognized by versions 2.4x and earlier of MacAnova. Keywords all and options are ignored and options are not saved.

save(FileName,v31:T [, var ...]) saves in the format recognized by versions 3.0 and 3.1x of MacAnova.

save(FileName,v335:T [, var ...]) saves in the format recognized by version 3.35 of MacAnova. NULL variables are not saved.

save(FileName,v406:T [, var ...]) saves in the format used in versions later than 3.35 but earlier than 4.07. Information on ticks in GRAPH variables is not saved and there are other differences. Use v406:T when creating a workspace file that is to be read by MacAnova 4.06 or earlier.

Note: Prior to version 4.01, names starting with '_' were not permitted. Variables with such names can be restored by earlier versions, but they cannot be used.

Cross references

See also topics asciisave(), restore(), 'options', 'files'

2.319 scalars

Usage:

Create a scalar variable: x <- 3.1415927; c <- "Hi, I'm Frank"; no <- F

Keywords: variables, syntax

Description

A scalar variable is a vector of length 1. It can be REAL, LOGICAL or CHARACTER. It consists of one item of information.

For practically all purposes, a matrix or array all of whose dimensions are 1 (matrix(sqrt(2), 1), for example) is also considered to be a scalar variable.

Examples

You can create scalar variables in many ways. Here are some examples.

```
  Cmd> sqrt2 <- sqrt(2); twopi <- 2*PI # REAL

  Cmd> bananas <- 7; cost <- prices[bananas]*3.1 # REAL

  Cmd> filename <- "babydata.txt"; today <- weekdays[5] # CHARACTER

  Cmd> fitmean <- F; expensive <- prices[bananas] > .49 # LOGICAL
```

In the third example, "babydata.txt" is an example of what is often called a quoted string - character information enclosed between two double quotation marks.

Cross references

See topics 'file_names' for the use of CHARACTER scalars as file names.

See topics 'variables', 'arithmetic' and 'transformations' for more information about the use of scalar variables.

2.320 screen()

Usage:

```
screen([Model] [, method:"cp" or "rsq" or "adjrsq", mbest:m, forced:fn,\
  s2:mse, penalty:pen, keep:items]), m positive integer, fn vector of
  positive integers, mse and pen positive REAL scalars, items CHARACTER
  scalar or vector with elements "p", "cp", "rsq", "adjrsq", "model",
  "intmodel" or "all".
```

Keywords: glm, regression

Usage

screen(Model) screens all the regression models based on one or more of the X variables given in the CHARACTER argument Model. The best of these models are printed, together with the values of Mallow's Cp = RSS/MSE + 2*p - n, multiple R^2 = coefficient of determination, and adjusted R^2 = 1 - (n-1)*(1-R^2)/(n-p), where p is the number of coefficients in the model, including the constant term.

In the definition of Cp, RSS is the residual sum of squares for a model and MSE is either the residual mean square from the model using all the variables or the value of optional keyword 's2'. When optional keyword 'penalty' is used, its value replaces 2 as multiplier of p in the definition of Cp. See below for details on 's2' and 'penalty'.

By default, screen() finds the models with the 5 smallest values of Mallow's Cp statistic. This default can be changed by keywords 'method' and 'mbest'; see below.

Model must not specify a model with no constant term. For example, screen("y=x1+x2+x3-1") is illegal. See topic 'models' for information on the form of Model.

Keywords 'keep' and 'print'

screen(Model, keep:Items) does the same, except that nothing is printed. Instead information specified by CHARACTER scalar or vector Items is returned as the value of screen(). See below for permissible values.

screen(model, keep:Items, print:T) both prints the results and returns those results specified by Items.

Permissible values of elements of Items:

"p"	Number of coefficients fit including constant (intercept)
"cp"	Mallow's Cp statistic
"rsq"	Multiple R^2
"adjrsq"	Adjusted R^2
"model"	Models selected in the CHARACTER form expected by regress()
"intmodel"	Integer Matrix with each column containing the indices of the variables in one of the selected models
"all"	All of the above

The values produced by the first 5 of these are vectors with one element for every model selected; "intmodel" produces a matrix with one column for every model selected and nv rows, where nv is the number of independent variables in Model.

When more than one item is specified, they are returned as components of a structure with names as in this list.

Details on value

vector("y=x1","y=x1+x3+x4","y=x2+x3+x4","y=x1+x2","y=x4") would be an example of a CHARACTER vector that might be produced by keep:"model". For this case, if there are 4 variables in all in the obvious order, the "intmodel" value would be the matrix

```
[ 1   1   2   1   4]
[ 0   3   3   2   0]
[ 0   4   4   0   0]
[ 0   0   0   0   0]
```

Other Keywords

Keyword	Type of value	Default	Meaning
method	CHARACTER variable	"cp"	Criterion ("cp", "rsq", or "adjrsq") for subset selection
mbest	Positive integer	5	Number of subsets to be found
forced	REAL Vector of positive integers or independent variable names	none	List of independent variables to be forced into all subsets
s2	Positive REAL number	MSE	Replacement for full model MSE in computing Cp
penalty	Positive REAL number	2	Multiplier of p in computing Cp

```
  print   T or F                          print:T forces printing when
                                          'keep' is used.

Keyword 'silent', 'fstats', and 'pvals' are illegal for screen(), and
'print:F' is illegal unless keyword 'keep' is used.

                            Keyword 'method'
The value of keyword 'method' determines the criterion to be used to
rank regressions.
    Value       Criterion Used   What is better
     "cp"        Mallow's Cp      Smaller
     "rsq"       Multiple R^2     Larger
     "adjrsq"    Adjusted R^2     Larger

                             Keyword 'mbest'
The number of subset regressions computed is determined by the value of
'mbest'.

With method:"adjrsq" or method:"cp", exactly mbest regressions are
printed or returned.

With method:"rsq", for each possible number m, m = 1, 2, ..., nv, of
variables, screen() selects the mbest models with largest value of R^2,
where nv is the number of variables in the model.  Thus in this case, up
to (nv-1)*mbest + 1 models would be selected.

                            Keyword 'forced'
The value of 'forced' should be a list of names of any variables that
should be forced into the model.  For instance, with forced:vector("x1",
"x2"), all models examined would include x1 and x2.  By default, no
variables are forced.  The value of 'forced' can also be a vector of
integers, say forced:vector(1,2).

                        Keywords 'penalty' and 's2'
The value of 'penalty' is used to compute Cp = RSS/MSE + penalty*p - n.
Larger values increase the "penalty" of including additional variables
and tends to produce models with fewer variables.  The default value is
2.

The value of 's2' is the MSE to be used in computing Cp.  If not
specified, the mean square error from the complete model is used.

                                 Examples
        Examples, all assuming Model is "y=x1+x2+x3+x4+x5+x6+x7"

Screen with defaults mbest = 5,method = "cp",and penalty = 2
  Cmd> screen(Model)

Screen for 10 best models using adjusted R^2 with variable x3 forced
into the model:
  Cmd> screen(Model,mbest:10,forced:"x3",method:"adjrsq") # or forced:3
```

```
Screen using Cp over models with x6 forced in and penalty factor of 3:
  Cmd> screen(Model,forced:"x6",penalty:3)

Screen using defaults, saving p, cp, and the models, and printing
results:
  Cmd> result <- screen(Model,keep:vector("p","cp","model"),print:T)

  Cmd> regress(result$model[1]) # compute regression with best model

                                Reference
screen() uses a branch and bound algorithm due to Furnival and Wilson.
See their paper, Regression by Leaps and Bounds, Technometrics 16 (1974)
499-511.

                             Cross references
See also regress() and anova().
```

2.321 secoefs()

Usage:

```
   secoefs([Term] [, errorTerm:ErrorTerm, byterm:F, se:F or coefs:F,
     silent:T]), Term a CHARACTER scalar, a positive integer, or a factor
     or variate in the current GLM model, ErrorTerm a CHARACTER scalar or
     positive integer.  byterm:F only when Term, se:F and coefs:F omitted
```

Keywords: glm, anova, regression, confidence intervals

```
                                   Usage
secoefs(Term) returns the model effects or regression coefficients and
their standard errors for the term specified in the CHARACTER variable
Term.  The result is a structure with components 'coefs' and 'se'.

The coefficients and standard errors pertain to the results of the most
recent GLM (generalized linear or linear model) command such as
regress(), anova(), or poisson().

When Term is a main effect term, the components are vectors.  When it is
an interation term, the components are matrices or arrays with the
leftmost subscript corresponding to the leftmost factor in Term.

Caution: After anova(), manova() and regress(), standard errors are
computed using the final error mean square in the model.  This may not
be appropriate with mixed models, including split plot designs.

                             Specifying the term
Term is usually a quoted string or CHARACTER variable such as "a.b"
which exactly matches a term in the most recent model, that is, "a.b" is
not the same as "b.a".  An interaction term produces a matrix or array
with the leftmost subscript corresponding to the leftmost factor in
Term.
```

If any variables in Term originally specified in the form {expr}, where expr is a MacAnova expression, you must include the enclosing '{' and '}'.

For a term which consists of a single factor or variate, Term can be its unquoted name.

Alternatively, Term can be a integer between 1 and the number of terms, excluding the final error term. For example, unless the model contained "-1", secoefs(1) gets the estimated intercept or grand mean and its standard error.

No term specified

secoefs() (no Term specified) computes coefficients and standard errors for all terms in the model. The result is a structure with one component for each term in the model, with each component itself a structure with components 'coefs' and 'se'.

The names of the top level components in the result are taken from the names of the terms, truncated if necessary to 12 characters. When such truncation is neccessary, the result is also given labels which contain the full component names. See topic 'labels'.

secoefs(byterm:F) is the same as secoefs() except that the resulting structure has two components, 'coefs' and 'se', each of which is a structure with one component per term (unless there is only one term in the model). In this case, the names of the bottom level components are taken from the possibly truncated names of the terms. When any truncation takes place, the full term names are also attached as labels.

Keyword 'silent'

secoefs(Term, silent:T) and secoefs(silent:T) do the same, but certain warning and advisory messages are suppressed. 'silent:T' can be used with any other keywords. This feature is useful in a macro when warning messages might confuse the user, or in a simulation. The default value of 'silent' is False unless the value of option' 'warnings' is False.

Suppressing coefficients or standard errors

secoefs(Term,coefs:F) and secoefs(coefs:F) (or secoefs(,coefs:F)) suppress the computation of the coefficients, returning a structure or matrix containing only standard errors. secoefs(Term,se:F) and secoefs(se:F) are equivalent to coefs(Term) and coefs(), respectively.

You can compute a structure containint t-statistics for every coefficient by

```
  Cmd> tt <- secoefs(se:F)/secoefs(coefs:F) #or coefs()/secoefs(coefs:F)
```

Alternatively, you could compute such a structure by

```
  Cmd> @tmp <- secoefs(byterm:F); tt <- @tmp$coefs/@tmp$se
```

After manova()

secoefs(Term,Varno) or secoefs(,Varno) computes coefficients and standard errors only for variable number Varno in the case of a

```
multivariate dependent variable.  If present, Varno must be the second
argument and any keywords must follow it.

                          Specifying error term
For all forms, an optional keyword phrase argument errorterm:ErrTerm or
errorterm:ErrTermNo, where ErrTerm is a CHARACTER variable or quoted
string specifying a term in the model and ErrTermNo is a positive
integer, specifies that the MS from the indicated term is to be used in
computing standard errors.

                          Confidence intervals
If tcrit is a critical value, say, invstu(1-alpha/2,errorDF), you can
compute the lower 1-alpha confidence limits for the coefficients
  Cmd> @tmp <- secoefs(byterm:F);@tmp$coefs - tcrit*@tmp$se
and similarly for upper limits (replace - by +).

                                 Example
After anova("y= a + b + a.b")
   secoefs(a), secoefs("a"), or secoefs(1) will compute the main effect
     coefficients and their standard errors for factor a
   secoefs(a,coefs:F), secoefs("a",coefs:F), or secoefs(2,coefs:F) will
     compute the standard errors of main effect coefficients
   secoefs("a.b") or secoefs(4) will produce matrices of the a by b
     interaction coefficients and their standard errors.
   secoefs() will produce all coefficients and their standard errors in
     a structure with components CONSTANT, a, b, and a.b.
   secoefs(byterm:F) will produce all coefficients and their standard
     errors in a structure with components coefs and se

This will produce the a by b interaction effects and their standard
errors.

                               Limitations
Secoefs() does not work after fastanova(), ipf(), or screen(), or if
'coefs:F' was an argument to the most recent GLM command.

                           Cross references
See also coefs(), contrast(), modelinfo(), popmodel(), pushmodel().
```

2.322 select()

Usage:

```
   select(k, x), k vector of positive integers or LOGICAL vector, x a
     matrix.
```

Keywords: combining variables, variables

```
                                 Usage
select(k, x) computes vector(x[1,k[1]],x[2,k[2]],...,x[n,k[n]]), where n
= nrows(k).  For example, select(k,x)[i] is x[i,k[i]], the k[i]-th
element of the i-th row of x. The length of the result is nrows(k).
```

k must be a REAL vector of positive integers or a LOGICAL vector and x must be a matrix with nrows(x) >= nrows(k) and ncols(x) >= max(k),

When k is a LOGICAL vector, False is translated to 1, and True is translated to 2. For example, when x is a matrix with two columns, select() can be used to select column 1 or column 2 of x depending on whether k[i] is False or True. NOTE: This differs from home LOGICAL subscripts are interpreted. See 'subscripts'

If k[i] is MISSING, select(k,x)[i] is MISSING when x is LOGICAL or REAL and is "" when x is a CHARACTER variable.

When x is REAL or LOGICAL and k has no MISSING values, select(k, x) is equivalent to vector(x[hconcat(run(nrows(k)), k)]).

Cross references

See topic 'subscripts'.

2.323 sethistory()

Usage:

sethistory(lines), lines is a CHARACTER vector with length <= value of option 'history'

Keywords: general

Usage

sethistory(Lines), where Lines is a CHARACTER vector, resets the internal list of previous commands to Lines with Lines[1] the earliest command available. Subsequent keyboard or menu retrieval of previous commands will retrieve elements of Lines.

It is an error if length(Lines) > nHist, where nHist is the value of option 'history'. See subtopic 'options:"history"'.

sethistory() is not implemented in the limited memory DOS version or in any version that does not allow keyboard or menu retrieval of previous commands.

Cross references

See gethistory() for an example using gethistory() and sethistory() to maintain a history of commands executed between MacAnova sessions.

2.324 setlabels()

Usage:

```
setlabels(x, labels [,silent:T]), x an existing scalar, vector, matrix,
  array or structure, labels a CHARACTER scalar or vector, a structure
  with CHARACTER scalar or vector components, or NULL
```

Keywords: general, variables

```
                                  Usage
setlabels(x, Labels) "attaches" coordinate or component labels in Labels
to variable x.  If x already has labels, they are replaced.  Labels must
a CHARACTER scalar or vector, a structure whose components are CHARACTER
scalars or vectors, or NULL.

x must be an existing scalar, vector or array of any type, or a
structure.

When Labels is NULL, any existing labels for x are removed.

When x is a vector, Labels normally is a scalar or vector of row labels
for x.  When x is a matrix or array, Labels is normally a structure with
ncomps(Labels)) = ndims(x) with Labels[I] a scalar or vector used to
label dimension I of x.

setlabels(x, Labels, silent:T) does the same, except messages
concerning a mismatch in the number of vectors of labels provided are
suppressed.

                          Wrong number of labels
It is an error when the length of a non-scalar vector of labels for a
coordinate does not match the dimension of the coordinate.  See topic
'labels' for information on how scalar coordinate labels are
interpreted.

When Labels is a vector and ndims(x) > 1 or labels is a structure with
ndims(x) > ncomps(Labels), the missing labels are all assumed to be "@".

Conversely, when Labels is a structure and ncomps(Labels) > ndims(x),
the extra vectors of labels are ignored.

                            Cross references
See also topics 'labels', getlabels(), haslabels().
```

2.325 setodometer()

Usage:

```
odometer <- setodometer([lower:L,] upper:U [,ndigits:M, ] [,place:N]),
  L and U integer scalars or vectors, integer M > 0, N >= 0
odometer <- setodometer(odometer, place:N), odometer a structure with
  integer components 'digits', 'lower', 'upper', 'place'
odometer <- setodometer(odometer [,step:n]), integer n, default 1
```

Keywords: general, macros

```
                        Description of an odometer
setodometer() is used to create or modify an "odometer".  An odometer is
a structure of the form

  structure(digits:D, lower:L, upper:U, place:N)

where D is a vector of M integers and L and U are integer vectors of
length M or scalars.  D, L and U satisfy L[i] <= D[i] <= U[i], where L
and U are interpreted as rep(L, M) and rep(U, M) when they are scalars.

See topics 'scalars', 'vectors' and 'structures' for information on
these types of variables.

See below for how to use setodometer() to create, set and advance an
odometer.

                                   Details
In the following, when L and/or U are scalars, L[i] and U[i] should be
interpreted as L and/or U (rep(L,M)[i] and/or rep(U,M)[i]).

Note: L[i] = U[i] is permitted.

An odometer is modeled on the distance display in an automobile, with
the M elements of D corresponding to the digits in the display.  It
differs in that D[i] runs from L[i] to U[i] instead of from 0 to 9, and
the digits are in reverse order (D[1] is the least significant, D[M] is
the most significant).  N corresponds to the distance traveled.  When N
= 0, D = vector(L[1], L[2], ..., L[M]) (rep(L,M) when L is a scalar).

Let I = D - L and let R = U - L + 1 (rep(U - L + 1, M) when U and L are
scalars) and define Size = prod(R).

When 0 <= N < Size, the elements of I are the 'digits' of N in a mixed
radix representation with radices R[i], in reverse order.  For example,
when M = 3, N = I[1] + I[2]*R[1] + I[3]*R[1]*R[2].  When L = 0 and U =
m-1 are scalars, the elements of I are the base m digits of N, in order
from least to most significant.  When N < 0 or N >= Size, the elements
of I are the digits of N modulo Size (N - Size*floor(N/Size)).

                            Creating an odometer
O <- setodometer(lower:L, upper:U, ndigits:M) creates an odometer O.  M
> 0 must be an integer, and L and U must be integer scalars or vectors
of length M, with a scalar L or U interpreted as rep(U,M) or rep(L,M).
L and U must satisfy L[i] <= U[i], R[i] = U[i] - L[i] + 1 < 2^31, and
Size = prod(R) < 2^52.  These limits may be different on some computer
systems.

Component 'digits' is initialized to L (or rep(L,M) when L is a scalar).
Components 'lower' and 'upper' are initialized to L and U, respectively,
and component 'place' is initialized to 0.
```

```
O <- setodometer(lower:L, upper:U), without 'ndigits:M', is equivalent
to O <- setodometer(lower:L,upper:U,ndigits:max(length(L),length(U))).

O <- setodometer(upper:U [,ndigits:M]) is equivalent to O <-
setodometer(lower:0, upper:U [,ndigits:M]); that is, the default value
for 'lower' is 0.

O <- setodometer([lower:L,] upper:U [,ndigits:M], place:N) does the
same, except O$place is set to N and O$digits is set to L + mixed radix
digits of N.  N must satisfy 0 <= N < Size = prod(U-L+1).

                   Setting and advancing an odometer
O1 <- setodometer(O, place:N) is equivalent to O1 <-
odometer(lower:O$lower,upper:O$upper,ndigits:length(O$digits), place:N);
that is, it creates an odometer O1 with the same L and U as O, but at
place N.

O1 <- setodometer(O, step:n), where O is an odometer and n is an
integer, creates an odometer O1 with O1$place = O$place + n.  O1$digits
is computed by stepping O$digits forward n steps when n >= 0, or
backward by -n steps when n < 0.  n must satisfy abs(n) < Size =
prod(U-L+1).  When abs(n) is large, this may take some time since it is
computed as a sequence of single forward or backward steps.

O1 <- setodometer(O) is equivalent to O1 <- setodometer(O, step:1); that
is the odometer is advanced by 1.

When 0 <= O$place + n < Size, setodometer(O, step:n) is equivalent to
setodometer(O, place:O$place + n), except that setodometer(O, step:n)
can be much slower when abs(n) is large.

                                Examples
Examples:
 Step through the various factor combinations of a 2^k design:
   Cmd> counter <- setodometer(upper:1,ndigits:k) #lower is 0

   Cmd> for(i,run(2^k)){
          levels <- counter$digits
          # do something with levels
          counter <- setodometer(counter);; # step by 1
        }

 Find hexadecimal representation
   Cmd> N <- 9 + 3*16 + 11*16^2 + 15*16^3; N
   (1)       64313

   Cmd> O <- setodometer(upper:15,ndigits:8,place:N); O # lower is 0
   component: digits
   (1)           9           3          11          15           0
   (6)           0           0           0
   component: lower
   (1)           0
```

```
   component: upper
   (1)          15
   component: place
   (1)       64313

   Cmd> letters <- vector("0","1","2","3","4","5","6","7","8","9",\
       "A","B","C","D","E","F")

   Cmd> paste(letters[reverse(O$digits)+1],sep:"")
   (1) "0000FB39"

   Cmd> O <- setodometer(lower:1,upper:16,ndigits:8,place:N); O #lower 1
   component: digits
   (1)          10           4          12          16           1
   (6)           1           1           1
   component: lower
   (1)           1
   component: upper
   (1)          16
   component: place
   (1)       64313

   Cmd> paste(letters[reverse(O$digits)],sep:"") # +1 not needed now
   (1) "0000FB39"

 Macro to find binary bits of integer from most to least significant
   Cmd> bits <- macro("@N <- argvalue($1,\"N\",\"pos int scalar\")
      @ndigits <- ceiling(log(@N+1)/log(2))
      reverse(setodometer(upper:1,ndigits:@ndigits,place:@N)$digits)")

   Cmd> bits(N)
    (1)           1           1           1           1           1
    (6)           0           1           1           0           0
   (11)           1           1           1           0           0
   (16)           1

                              Cross references
See also paste(), prod(), reverse(), max(), macro(), 'macros',
'macro_syntax'.
```

2.326 setoptions()

Usage:

```
   setoptions(option1:value [,option2:value ... ] [,badoptok:T]) option1,
     option2, ... option names
   setoptions(str [,badoptok:T]), where str is of the form
     structure(option1:value, ...)
   Type 'usage(options)' for a succinct list of all options and their
      permissible values.
```

Keywords: control, missing values, output, random numbers

Usage

setoptions(keyword:value [,keyword:value, ...]) changes the values of various items specified by the keywords.

Legal option names are 'angles', 'batchecho', 'dumbplot', 'errors', 'findmacros', 'format', 'fstats', 'height', 'inline', 'labelabove', 'labelstyle', 'maxlinelen', 'maxwhile', 'minpvalue', 'missing', 'nsig', 'prompt', 'pvals', 'quiet', 'restoredel', 'seeds', 'update', 'traceback', 'warnings', 'wformat', and 'width'.

Option name 'lines' is a synonym for 'height' for compatibility with previous versions.

Options 'history' and 'savehistry' (note spelling) are also available (see gethistory() and gethistory()).

On windowed versions, option 'scrollback' is also legal.

In the Windows version, option 'keyboard' is also legal.

On Mac OS 9, options 'font' and 'fontsize' are also legal.

See topic 'options' for details on these options.

Keyword 'badoptok'

setoptions(keyword:value [,keyword:value, ...] , badoptok:T) does the same, except it is not an error is a non-existent or otherwise improper option is specified. This feature is intended to provide backward compatibility to macros making use of new options.

Structure argument

setoptions(Options [,badoptok:T]), where Options is a structure with component names matching any or all of the legal option names, sets the options from the component values. For example, if Options was created by 'Options <- getoptions()', setoptions(Options) resets the options to what they were at the time getoptions() was invoked. See also getoptions().

Restoring default values for all options

setoptions(defaults:T) restores all the options to their default values. It is an error if there is more than one argument. If a prompt was set at start up, it is restored. When setoptions(defaults:T) occurs in a batch file, the current batch prompt is restored to what it was when the batch file was opened, either the file name or the value of keyword 'prompt' on the batch() command. See batch().

Options menu on Mac OS 9

On Mac OS 9, you can use the Options menu to change most of the options. This works by generating and executing an appropriate setoptions() command.

2.327 setseeds()

Usage:

```
setseeds(seed1,seed2) or setseeds(vector(seed1,seed2)), seed1 and seed2
  non-negative integers <= 2147483399
```

Keywords: random numbers

```
                                Usage
setseeds(seed1,seed2) initializes the random number generator used by
runi(), rnorm(), rbin() and rpoi().  The seeds should be non-negative
integers <= 2147483399.

setseeds(seeds), where seeds is a vector of the form vector(seed1,
seed2), is equivalent to setseeds(seed1, seed2).  In particular, any
time after setting seeds by seeds <- getseeds(), setseeds(seeds)
restarts the generator to the state it was when you used getseeds().

If either seed is zero, the seeds are intialized with values generated
from the time of day; the seeds generated will normally be printed.  You
can suppress the printing by setseeds(0,0,quiet:T).

                           Cross references
See also topics getseeds(), runi(), rnorm(), rpoi(), rbin() and
subtopic 'options:"seeds"'.
```

2.328 shapeof()

Usage:

```
shapeNames <- shapeof(arg1 [, arg2 ...] [,strict:T]), arg1, arg2, ...
  arbitrary variables, including macros and built-in functions.
```

Keywords: variables

```
                                Usage
shapename <- shapeof(arg), where arg is REAL, LOGICAL, CHARACTER or
LONG,  sets shapename to one of "SCALAR", "VECTOR", "MATRIX", or
"ARRAY", classifying the shape by the first of isscalar(arg),
isvector(arg), ismatrix(arg) or isarray(arg) to be true.

When arg is not REAL, LOGICAL, CHARACTER, or LONG, shapeof(arg) returns
one of "STRUCTURE", "MACRO", "FUNCTION", "GRAPH", "UNDEFINED" or "NULL"
just as does typeof(arg).

shapename <- shapeof(arg, strict:T) does the same, except "SCALAR" and
"VECTOR" are returned only when ndims(arg) = 1 and "MATRIX" is returned
only when ndims(arg) = 2.

shapenames <- shapeof(arg1, arg2 ... [,strict:T]), where arg1, arg2,
... are any variables, makes shapenames a CHARACTER vector with
length(shapenames) = number of arguments, with shapenames[i] the name of
```

```
the shape of argument i.

                                Examples
Examples:

  Cmd> a <- sqrt(2); shapeof(a)
  (1) "SCALAR"

  Cmd> b <- matrix(run(4),4); shapeof(b)
  (1) "VECTOR"

  Cmd> shapeof(b,strict:T)
  (1) "MATRIX"

  Cmd> shapeof(run(4)) == shapeof(b) # compare shapes
  (1) T

  Cmd> shapeof(run(4),strict:T) == shapeof(b,strict:T) # compare shapes
  (1) F

  Cmd> shapeof(array(PI,1,1,1),help,NULL,T,cos,strict:T)
  (1) "ARRAY"
  (2) "MACRO"
  (3) "NULL"
  (4) "SCALAR"
  (5) "FUNCTION"

                            Cross references
See also topics nameof(), typeof(), isarray(), ischar(), isdefined(),
isfactor(), isfunction(), isgraph(), islogic(), ismacro(), ismatrix(),
isname(), isnull(), isnumber(), isreal(), isscalar(), isvector().
```

2.329 shell()

Usage:

```
   shell(command), shell(command,keep:T) or shell(command,interact:T),
     command a quoted string or CHARACTER scalar.
   !command immediately after the prompt
```

Keywords: control, general

```
                                  Usage
shell(command) submits the CHARACTER vector or string command to the
operating system for execution.  It is implemented in the Unix/Linux,
Motif, and DOS versions but not on the Macintosh.  It does not work in
Windows 3.1 or Windows 95 but may in Windows NT.  Except in the limited
memory DOS version (BCPP), the program run by command must not expect
any input from the keyboard.

For example, in the Unix/Linux or Motif versions,
  Cmd> shell("ls -l *.dat")
```

causes the output from the Unix/Linux command 'ls -l *.dat' to be printed, giving a full listing of all files in the current directory with names ending with '.dat'. Under DOS, the same effect is obtained by

```
  Cmd> shell("dir *.dat").
```

Keyword 'interact'

shell(command, interact:T) does the same as shell(command) except that you can interact with the program that is started up. This option is required if, for example, you are using shell() to run an editor to modify a file. In the limited memory DOS version (BCPP), you can always interact with the command. When in doubt as to whether a program expects keyboard input, use interact:T.

Keyword 'keep'

shell(command, keep:T) runs the command in non-interactive mode and returns its output as a CHARACTER vector, with each line of output an element. The command must not expect any input from the keyboard. This is not implemented in the limited memory DOS version (BCPP).

Example on Unix/Linux

```
  Cmd> datafiles <-\
         shell(paste("cd ",DATAPATHS[1],"; ls *.dat"), keep:T)
```

returns a vector of file names of the form *.dat in the directory whose name is in DATAPATHS[1]. See topic 'DATAPATHS'.

Operating System Escapes

Somewhat simpler, but less powerful because it cannot be included in a macro, is the use of the 'escape' character '!'. In the Unix/Linux, Motif and DOS versions, any line of input starting with '!' immediately after the prompt will be passed on to the operating system for execution (without the '!'). Specifically,

```
  Cmd> !command to be run ...
```

is equivalent to

```
  Cmd> shell("command to be run ...", interact:T).
```

For example,

```
  Cmd> !ls -l *.dat
```

and

```
  Cmd> shell("ls -l *.dat", interact:T)
```

are equivalent.

In a command line starting with '!', double quotes ('"') and curly brackets ('{' and '}') have no special significance. The only "special" character is a backslash ('\') and then only when it occurs at the end of line to indicate that the command is continued on the next line. Unlike the case when an ordinary line is continued with backslash, a trailing backslash is deleted and is not seen by the operating system. Unbalanced quotes or brackets are ignored.

Under DOS, but not Unix/Linux, you can change default directories by, for example,

```
  Cmd> !cd b:\data   # or shell("cd b:\\data")
```

```
Caution: If you want to execute in MacAnova a command that starts with
'!'  (for example !(x < y)), precede it with a space.  For example,
  Cmd> !(x < y)
attempts to execute "(x < y)" as a shell command, probably causing an
error, while
  Cmd>  !(x < y) #note the extra space
is computed in MacAnova.  Conversely,
  Cmd>  !ls -l *.dat
is an error, because a space has been typed before '!'.

                          Comparison of versions
In the DOS extended memory version (DJGPP), shell(command) or
shell(command, keep:T) sometimes hangs.  shell(command,interact:T) or
!command is more reliable.

In the Windows version, shell(command,interact:T) and !command ignores
command and starts up DOS similar to selecting MS-DOS Prompt in the
Program Manager window.

In the Motif version, shell(command,interact:T) and !command can be
confusing: Output does not appear in the MacAnova command window, but in
the "parent" window from which MacAnova was started up, and any input
must be typed in parent window.

On a Macintosh, starting a line with '!' is an error.
```

2.330 showplot()

Usage:

```
showplot([Graph] [,graphics keyword phrases])
```

Keywords: plotting

```
                                    Usage
showplot(Graph) redraws the graph whose information is encapulated in
Graph, which must be a variable of type GRAPH.  If Graph is omitted,
GRAPH variable LASTPLOT is plotted instead.

If option 'dumbplot' has been set False (see subtopic
'options:"dumbplot"'), the plot will be a low resolution plot unless
'dumb:F' is an argument.

                              Graphics keywords
Keywords 'dumb', 'xmin', 'xmax', 'ymin', 'ymax', 'logx', 'logy', 'xlab',
'ylab', 'title', 'xaxis', 'yaxis', 'borders', 'ticks', 'xticks',
'yticks', 'xticklen', 'yticklen', 'xticklabs', 'yticklabs', 'height',
'width', 'pause', 'silent' and 'notes' may be used as for other plotting
commands.  See topics 'graph_keys', 'graph_border' and 'graph_keys'.
This allows you to change the original labeling and limits, as well as
modify tick mark placement and labelling and which borders of the plot
```

```
are drawn.

                          Updating LASTPLOT
As all plotting commands, showplot() updates LASTPLOT to reflect the
graph being plotted.  To suppress the creation or updating of LASTPLOT,
use keyword phrase 'keep:F' as an argument.  Occasionally when memory is
limited, it may be necessary to use keep:F to view the graph.

See topic 'graph_assign' for information on another way to make plots.

See also plot(), chplot(), lineplot().
```

2.331 sin()

Usage:

```
   sin(x [, degrees:T or radians:T or cycles:T]), x REAL or a structure
     with REAL components x in radians (default), cycles, or degrees as set
     by option "angles" or the optional keyword.
```

Keywords: transformations

```
                                  Usage
sin(x) computes the sines of the elements of x, where x is a REAL
scalar, vector, matrix or array.  The result has the same shape as x.

The argument is considered to be in units of radians, degrees or cycles
as determined by the value of option 'angles'.  The default is radians.
See subtopic 'options:"angles"'.

sin(x, radians:T), sin(x, degrees:T), sin(x, cycles:T) interpret x as in
the indicated units, regardless of the value of option 'angles'.

If any element of x is MISSING or is too large (> 5000000*PI radians in
absolute value), the corresponding element of the result is MISSING and
a warning message is printed.

                          Structure argument
When x is a structure, all of whose non-structure components are REAL,
sin(x [,UNITS:T]), where UNITS is one of 'radians', 'degrees' or
'cycles', is a structure of the same shape and with the same component
names as x with each non-structure component transformed by sin().

                          Cross references
See topic 'transformations' for more information on sin(), including its
use with a CHARACTER argument.
```

2.332 sinh()

Usage:

```
sinh(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                  Usage
sinh(x) returns the hyperbolic sine of the elements of x, when x is a
REAL scalar, vector, matrix or array.  The result has the same shape as
x.  In terms of other functions, sinh(x) = (exp(x) - exp(-x))/2.

If any element of x is MISSING or > 710.4758600739439 in absolute value,
the corresponding element of sinh(x) is MISSING and a warning message is
printed.

When x is a structure, all of whose non-structure components are REAL,
sinh(x) is a structure of the same shape and with the same component
names as x, with each non-structure component transformed by sinh().

                              Cross references
See topic 'transformations' for more information on sinh().
```

2.333 solve()

Usage:

```
solve(A [,singok:T] [,quiet:T]), square REAL matrix A
solve(A, B [,singok:T] [,quiet:T]), square REAL matrix A, REAL matrix B,
  with nrows(B) = nrows(A).
```

Keywords: linear algebra

```
                                  Usage
solve(a) computes the inverse of the square matrix a.

solve(a,b) computes the solution x to the linear equation a x = b, where
a is square and b is a REAL vector or matrix with the same number of
rows as a.  solve(a,b) is mathematically, but not computationally, the
same as solve(a) %*% b.

For either usage, if a is singular, an informative message is printed
and the operation aborts.

MISSING values are not allowed in a or b.

Expression a %\% b is equivalent to solve(a, b).

                            Keyword 'singok'
solve(a, singok:T) and solve(a, b, singok:T) does the same, except when
a is singular, no message is printed and NULL is returned.  This allows,
for example, a macro to test whether a matrix is singular and take
```

corrective action.

Keyword 'quiet'

Occasionally, "overflow" occurs during the computation. Any values in the result that are too large to be represented are replaced by MISSING and a WARNING message is printed. You can suppress the message by including 'quiet:T' as an argument. If you do, you should use anymissing() to check the result for the presence of MISSING elements.

Propagation of labels

If a has labels, the row and column labels of solve(a) are the column and row labels of a, respectively. If b is compatible, the row and column labels of solve(a,b) are the column labels of a and b respectively. If b does not have labels, the column labels of solve(a,b) are rep("a", ncols(b). See topic 'labels'.

Cross references

See also topics rsolve(), swp(), qr(), 'matrices'.

2.334 sort()

Usage:

sort(x [,down:T]), x REAL or CHARACTER or a structure with all REAL or all CHARACTER components.

Keywords: ordering

Usage

sort(x) sorts the data in a REAL or CHARACTER vector x into ascending order.

sort(x, down:T) or simply sort(x,T) sorts the data in descending order.

If x is a matrix, sort(x) or sort(x,T) is a matrix each of whose columns contains the ordered elements of the corresponding column of x.

If x is an array, sort(x) or sort(x,T) is an array of the same size and shape with all the elements with fixed values of subscripts 2, 3, ... defining a "column" which is sorted. An array with dimension > 2 is always treated as an array and not as a matrix, even if there are at most two dimensions greater than 1.

With REAL data, any MISSING values in a column are sorted to the end of the column, regardless of the direction of the sort.

Sorting character data

With CHARACTER data, elements are sorted using the ASCII collating sequence in which most punctuation and all numerals sort ahead of upper case letters which sort ahead of lower case letters. A space sorts ahead of all printable characters. Here is the explicit ordering starting with space:

```
    !"#$%&'()*+,-./0123456789:;<=>?
   @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
   `abcdefghijklmnopqrstuvwxyz{|}~
```

Structure argument

It is also acceptable for x to be a structure, whose non-structure components are all REAL or all CHARACTER. In that case, sort() returns a structure of the same form, each of whose non-structure components is the result of applying sort() to the corresponding component of x.

Cross references

See also grade(), rank().

2.335 split()

Usage:

```
split(x,A [,compnames:CharVar ,silent:T]), x REAL, A a factor or vector
  of integers or LOGICAL vector, CharVar a CHARACTER scalar or vector
split(x,bycols:T or byrows:T [,compnames:CharVar, silent:T]), x a REAL
  matrix
```

Keywords: combining variables, structures

Usage

split(Data,Factor) creates a structure by splitting Data along its first subscript into separate components according to the values of Factor all of whose elements must be positive integers.

If N = max(Factor), the result has N components, some of which may be empty. Thus, when the values in Factor are group or treatment numbers, each component of the result consists of the data corresponding to a particular group or treatment. It is an error if N > 32767.

It is also acceptable for Factor to be a LOGICAL vector, in which case False and True correspond to factor levels 1 and 2, respectively. For example, split(y, x <= 0) would create a structure with two components.

Data must be REAL or LOGICAL and the components of the result are the same type. Each component of the result will be a vector, matrix or array, depending on whether Data is a vector, matrix, or array. A warning is printed if any component of the result contains no elements. If Factor[i] is MISSING, all the corresponding data are omitted. It is an error for all the elements of Factor to be MISSING.

split(Data,Factor,silent:T) does the same, except warning messages about missing values in Factor or empty components in the result are suppressed.

Names of components

If Factor is a variable rather than an expression, say groups or @groups, the components will be named 'groups1', 'groups2', etc.

```
Similarly if Factor is specified in a keyword phrase such as
dose:rep(run(4),5), components will be named 'dose1', 'dose2', etc.

                    Keywords 'bycols' and 'byrows'
split(Data,bycols:T) or simply split(Data) splits the data along the
last subscript, creating a structure with one component corresponding to
each value of the last subscript.  The most important case is when Data
is a m by n matrix, in which case the result each of the n components of
the result is a vector containing the data from a column of Data.
Components will be named 'col1', 'col2', ... .  If Data is a vector, the
result is a structure with a single component named 'col1'.

split(Data,byrows:T) splits the data along the first subscript, creating
a structure with one component corresponding to each value of the first
subscript.  For example, when Data is a m by n matrix, the result is a
structure with m components, each a row vector of size n (1 by n
matrix).  Components will be named 'row1', 'row2', ... .

                          Keyword 'compnames'
For all these usages, an additional argument of the form
compnames:CharVec, is recognized, where CharVec is a CHARACTER vector.
The elements of CharVec are used as names for the components of the
result, truncated to 12 characters if necessary, overriding the naming
conventions just described.  If length(CharVec) = 1, say "group", it is
used as a "root" for forming names for the components of the form
"group1", "group2", ... .  Otherwise length(CharVec) must match the
number of components of the result.  It is an error if any element of
CharVec contains '$'.

                          Use with boxplot()
An important use of split is in boxplot(split(y,groups)), where y is a
REAL vector and groups is a factor.  This produces parallel boxplots of
the of the data in y corresponding to each level of groups.  Similarly,
when y is a REAL matrix, boxplot(split(y,bycols:T)) or simply
boxplot(split(y)), produces parallel box plots of the data in each
column of y.

                             Examples
Examples:
  Cmd> split(run(4),variety:vector(1,2,1,2))
  component: variety1
  (1)            1           3
  component: variety2
  (1)            2           4

An equivalent command would be
  Cmd> split(run(4),vector(1,2,1,2),compnames:"variety")

  Cmd> boxplot(split(y),ylab:"Column Number")
where y is a matrix, produces parallel boxplots of the columns of y.

                          Cross references
See also topics boxplot(), 'structures'.
```

2.336 spool()

Usage:

```
spool(FileName [,new:T] [,everything:T or F])
spool() toggles spooling on and off.
spool(,new:T) restarts spooling on the same file
spool(,everything:T or F)
```

Keywords: output, files

```
                                  Usage
spool(FileName) begins printing MacAnova input and output into the file
with name given in the CHARACTER variable FileName.  If the named file
already exists, the spooled output will be added at the end of the file,
thus allowing a cumulative record of several runs.  Comments to annotate
what you have done may be added to input lines by preceding them with
'#'.  See topic 'comments'.  In a version with windows, when FileName
is "", you will be prompted for a name.

Output to the screen or window that is suppressed because option 'quiet'
has been set True, is not spooled to the file either.

spool() suspends spooling if spooling is currently in effect and
restarts it on the same file if spooling was previously in effect but is
not now.

spool(FileName, new:T) restarts spooling at the beginning of the spool
file, destroying any lines previously spooled.

spool(, new:T) restarts or resumes spooling at the beginning of the most
recently used spool file, destroying any lines previously spooled.

                            Keyword 'everything'
spool(FileName, everything:T [,new:T]) does the same, except lines are
spooled even when output to the screen or window is suppressed by option
'quiet'.

spool(, everything:T or F) allows or suppresses spooling of lines
suppressed by setoptions(quiet:T).  It does not turn toggle spooling on
or off.

                            Menu selection
On a Mac OS 9, selecting item Spool Output to File on the File menu is
equivalent to typing 'spool("")'.  If a spool file name has previously
been provided, the item is labeled either Suspend Spooling or Resume
Spooling and is equivalent to 'spool()'.  In both cases it first erases
everything after the prompt.
```

2.337 sqrt()

Usage:
```
sqrt(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                Usage
sqrt(x) returns the square roots of the elements of x, when x is a
REAL scalar, vector, matrix or array.  The result has the same shape as
x.

If any element of x is MISSING, so is the corresponding element of
sqrt(x).  If any element of x < 0, the corresonding element of sqrt(x)
is MISSING.  In both cases a warning message is printed.

When x is a structure, all of whose non-structure components are REAL,
sqrt(x) is a structure of the same shape and with the same component
names as x, with each non-structure component transformed by sqrt().

                            Cross references
See topic 'transformations' for more information on sqrt().
```

2.338 stemleaf()

Usage:
```
stemleaf(x [, nstems, outliers:F, depth:F, stats:T,\
   title:"Your title"]), x a REAL vector
```

Keywords: descriptive statistics, plotting

```
                                Usage
stemleaf(var) prints a stem and leaf display of the data in REAL vector
var.  The number of stems depends on the number of non-missing values
(must be at least 2) and the number of lines on the screen, and the
maximum number of leaves printed is determined by the screen width (see
subtopic 'options:"width"').  An asterisk as the last leaf on a stem
indicates there has been truncation and some leaves have not been
printed.

stemleaf(nvar,nstems), nstems an integer > 1, selects the number of
stems to be as large as possible <= nstems.

The data are scaled by a power of 10 and rounded toward zero so as to be
integers.  Then the final digits are the leaves and the stems are the
leftmost digits or 0.  If two stems go with a value, they are labeled,
for example, as '2*' and '2.'), If 5 stems go on a value, they are
labeled, for example, as '2*', '2t', '2f', '2s', and '2.'.  A final line
specifies the unit of the leaf digit.

By default, outliers more than 1.5 IQR beyond the lower or upper
```

```
quartiles are listed separately and are not put on a stem, where IQR is
the inter-quartile range.

                              Depth column
By default, a "depth" column accumulating counts from each end is
printed.  The depth for the stem that contains the median is the number
of leaves on that stem enclosed in parenthese.

                                Keywords
With both forms additional keyword arguments are as follows:
   outliers:F         This suppresses the special treatment of outliers;
                      all values are put on stems
   depth:F            Suppresses printing the "depth" column.
   stats:T            Print the sample size, extremes and quartiles.
   title:"Your title" Prints the specified CHARACTER variable before
                      display

                           Cross references
See also boxplot().
```

2.339 strconcat()

Usage:

```
   strconcat(var1 [,var2,...,vark] [, KeyPhrases]), where var1, var2, ...
     are arbitrary variables
   KeyPhrases can be compnames:Names, labels:Labels, notes:Notes and
     silent:T, where Names, Notes and Labels are CHARACTER scalars or
     vectors.  Arguments var1, ... can also be keyword phrases with keyword
     names other than 'compnames', 'labels', 'notes' and 'silent'.
```

Keywords: structures, combining variables

```
                                  Usage
strconcat(a,b,c,...) creates a structure from variables or structures a,
b, c, ... .

strconcat() differs from structure() in that, when an argument is a
structure, its top level components become top level components of the
result, while a structure argument to structure() becomes a single
component of the result.  For example, suppose a and d are non-structure
variables.  Then the value of strconcat(a, structure(b,c),d) is a
structure with 4 components, a, b, c, and d, while the value of
structure(a, structure(b,c),d) is a structure with 3 components, the
second of which is itself a structure with 2 components.

The names of components derived from non-structure arguments are
determined similarly to the names of components of structures created by
structure(), except that when varj is a keyword phrase, the keyword name
is ignored when the argument is a structure.

                        Attaching labels or notes
```

```
You can attach a vector of labels to a structure Str, one element for
each top level component, by Str <- strconcat(Str, labels:CharVec),
where CharVec is either a CHARACTER scalar or a CHARACTER vector of
length ncomps(Str).  When the structure is printed, the labels are used
instead of the component names.  You remove labels by Str <-
strconcat(Str,labels:NULL).  See topic 'labels' for more information.

str <- strconcat(a,b,...,notes:Notes), where Notes is a CHARACTER scalar
or vector, attaches Notes to str.  See topic 'notes'

When there is just one nonkeyword argument and it is a structure, its
labels or notes, if any, are transferred to the output unless keywords
'labels' and/or 'notes' are arguments.

                              Other keywords
See structure() for information about keywords 'compnames', 'notes' and
'silent'.

                                  Example
Example: Build structure in a loop
  Cmd> nterms <- dim(SS)[1];f <- structure(SS[1]/DF[1])
  Cmd> for(i,2,nterms-1){f <- strconcat(f,SS[i]/DF[i]);;}
  Cmd> f <- strconcat(f/(SS[nterms]/DF[nterms]),\
        compnames:TERMNAMES[-nterms])
This produces a structure consisting of F-statistics with components
having the names of the terms.

                             Cross references
See also topics 'structures', structure(), compnames(), changestr(),
'keywords'.
```

2.340 stringplot()

Usage:

```
stringplot([Graph,] x,y, strings:charVec,[, add:T, graphics keyword
  phrases])
stringplot([Graph,] [x,y, strings:charVec],keys:str), str a structure
  whose component names are graphics keywords
```

Keywords: plotting

```
                                  Usage
stringplot(x,y,strings:charVec) draws a graph, drawing charVec[i] at
position (x[i], y[i]).  x and y must be REAL vectors and charVec a
CHARACTER vector, all of the same length.  This contrasts with the
behavior of other plotting functions (except addstrings()) which allow y
to be a matrix, and permit x to be a scalar or vector of length 2 coding
equally spaced values.

It is not an error when x or y is NULL; a warning message is printed and
no plotting occurs.
```

For backward compatibility with earlier versions, keyword 'strings' can be omitted (stringplot(x,y,charVec)).

If option 'dumbplot' has been set False (see subtopic 'options:"dumbplot"'), the plot will be a low resolution plot unless 'dumb:F' is an argument.

Justifying strings

By default, each string is written centered at (x[i], y[i]). However, if 'justify:"l"' or 'justify:"r"' is an argument following charVec, each string will be left or right justified.

Graphics keywords

Most of the usual graphics keywords may be used, including 'xmin', 'xmax', 'ymin', 'ymax', 'logx', 'logy', 'xlab', 'ylab', 'title', 'border' and keywords related to ticks, but not 'symbols', 'lines', 'linetype', 'thickness' or 'impulses'. See topic 'graph_keys'.

In particular, stringplot() is most commonly used with add:T, in which case the strings being drawn are added to the graph encapsulated in GRAPH variable LASTPLOT. With 'add:T', stringplot() is equivalent to addstrings().

Graph variable argument

stringplot(Graph,x,y,strings:charVec), displays GRAPH variable Graph, adding the string or strings in charVec, and saves the modified plot in LASTPLOT. Graph is not changed (unless it is LASTPLOT).

Examples

The most usual use is when both x and y are REAL scalars and charVec is a quoted string or CHARACTER scalar to be added to a plot at coordinates (x,y). A typical usage would be

```
  Cmd> stringplot(110,20,strings:"Frequency 1 cycle/week",\
    justify:"l",add:T)
```

Alternatively you can use addstrings():

```
  Cmd> addstrings(110,20,strings:"Frequency 1 cycle/week", justify:"l")
```

Keywords 'keep' and 'show'

stringplot(x,y,strings:charVec,keep:F) suppresses any change to LASTPLOT.

stringplot(x,y,strings:charVec,show:F,add:T) suppresses immediate display of the modified graph but updates LASTPLOT. This is useful when you are building a complex graph in stages using addlines(), addchars(), addpoints(), or stringplot(). When you are done, simply type showplot(). You can't use both show:F and keep:F.

Keyword 'keys'

stringplot([Graph,] keys:structure(x:x,y:y,strings:charVec [other keyword phrases])) is equivalent to stringplot([Graph,] x:x,y:y, strings:charVec [other keyword phrases]). See topic 'graph_keys' for

details.

Cross references

See topic 'graph_assign' for information on another way to make plots.

See topic 'graphs' for general information on plots and on variable LASTPLOT. See topic 'graph_keys' for information on keywords. See topic 'graph_files' for information on writing a graph to a file.

2.341 structure()

Usage:

structure(var1 [,var2,...,vark] [, KeyPhrases]), where var1, var2, ... are arbitrary variables

KeyPhrases can be compnames:Name, labels:Labels, notes:Notes and silent:T, where Names, Notes, and Labels are CHARACTER scalars or vectors. Arguments var1, ... can also be keyword phrases with keyword names other than 'compnames', 'labels', 'notes' and 'silent'.

Keywords: structures, combining variables

Usage

structure(var1,var2,....,vark) creates a structure with components named var1, var2, ..., vark. The values of the components are equal to var1, etc.

structure() differs from strconcat() in that the any argument that is a structure becomes a single component in the result, whereas strconcat() splits it into its top level components, each of which become top level components of the result.

Names of components

When varj is a keyword phrase of the form name:value, the component is named from the keyword. For example, when a, b, and c are variables, structure(a,b,c) and structure(a:a,b:b,c:c) are equivalent. The names should differ from the keyword names 'labels', 'notes', 'silent' and 'compnames' recognized by structure(). See below for a work around.

When varj is a temporary variable, that is, a variable whose name starts with '@', then the '@' is stripped off to create the name of the component. For example, when @x is defined, structure(@x,...) is the same as structure(x:@x,...).

If varj is an expression or function result, for example "3+4" or "x'", the component name will be NUMBER, LOGICAL, STRING, VECTOR, MATRIX, ARRAY, STRUCTURE, MACRO, or GRAPH depending on the type and shape of the component.

structure(var1, var2, ..., compnames:Names), where Names is a CHARACTER vector or scalar, forms component names from Names, whose elements are truncated to 12 characters, if necessary. These take precedence over

other names as described above. If Names is a scalar, say "Comp", it is used as a root to create names of the form "Comp1", "Comp2", ... Otherwise the number of elements in Names must match the number of components. This usage is particularly useful in assigning names of 11 or 12 characters such as 'covariances" or 'factorloding'.

Legal component names

It is an error if any element of Names contains '$', a space or other "invisible" character.

Attaching labels or notes

structure(var1, var2, ..., labels:Labels), where Labels is a CHARACTER vector or scalar, labels the components using Labels. If Labels is a vector, then its length must match the number of components. The labels are printed instead of the component names on all output. See topic 'labels'. See strconcat() for information on how to add or remove labels from an existing structure.

structure(var1, var2, ..., notes:Notes), where Notes is a CHARACTER scalar or vector, attaches Notes as descriptive notes to the result. See topic 'notes'.

Keyword 'silent'

You can suppress all warning messages by silent:T as in structure(x, y, labels:vector("Height","Strength"), silent:T).

Order of arguments

Keywords compnames, labels, notes, and silent must follow all arguments that are to be included in the output structure. If you want a component with one of these names, either name it using keyword 'compnames' or make sure it is not the last component in the result. For example, to have a component 'labels', use, say, structure(1,"A", compnames:vector("x","labels")) or structure(labels:"A", x:1).

Examples

Example: You can create a structure with components min and max by

```
  Cmd> extremes <- structure(min:min(x), max:max(x))
```

by

```
  Cmd> @min <- min(x);@max <- max(x); extremes <- structure(@min,@max)
```

or by

```
  Cmd> extremes <- structure(min(x),max(x),\
        compnames:vector("min","max"))
```

In all three cases, to give a more complete labeling of the components you can include an argument like labels:vector("Minimum of x", "Maximum of x") .

Cross references

See also topics strconcat() and 'structures'.

2.342 structures

Usage:

```
Create a structure:
  Str <- structure(Name1:x,Name2:y,...[, compnames:Name])
  Str <- strconcat(Name1:x,Name2:y,...[, compnames:Name])
Extract components of a structure:
  Str$name  or  Str[J]  or  Str[[J]]
Number of components of a structure:
  ncomps(Str)
Component names of a structure:
  compnames(Str)
Test whether variable is a structure:
  isstruc(x)
Modify a structure:
  Str[J] <- x or Str[[J]] <- x, legal subscript vector J
  Str[[i]][[J]] <- x, integer scalar i, legal subscript vector J
  Str$a <- x, Str$a$b <- x, ...
  Str <- changestr(Str,"compname",x), equiv. to Str$compname <- x
  Str <- changestr(Str,compname:x), same as preceding
  Str <- changestr(Str,n,x), positive integer n <= ncomps(Str)+1
  Str <- changestr(Str,-n), equivalent to Str <- Str[-n]
```

Keywords: structures, syntax, variables

```
                        Description of structure
A 'structure' is made up of one or more named components, each of which
may itself be a variable or another structure.

A single component can be extracted by name using a '$'.  For example,
if Stats is structure(mean:xbar, var:s_sq), then xbar <- Stats$mean and
s_sq <- Stats$var set xbar and s_sq to the values of components 'mean'
and 'var', respectively.

When a component of a structure is itself a structure, you can use a
chain of component names.  For example, if structure Str has component
'a' which is a structure with component 'b', Str$a$b accesses that
component.  If Str$a$b is itself a structure with component 'c',
Str$a$b$c accesses that component

                          Creating a structure
Some functions and macros, for example secoefs(), describe() and
eigen(), return structures as their values.

You can use structure() and strconcat() to create a structure with
specified components.

You can use strconcat() to combine the components in two or more
structures into one larger structure, optionally including additional
variables as comnponents.

You can use split(y, f), where y is a vector or matrix and f is a
factor, to create a structure whose j-th component consists of the rows
of y associated with level j of f.  split(y) returns a structure, each
```

of whose components is a column of y. split(y,byrows:T) returns a structure, each of whose components is a row of y. See split().

You can use changestr() to delete components from, replace components in, or add new components to an existing structure. See structure(), strconcat() and changestr(). You can also modify an existing structure by assigning a variable to one or more components selected by a subscript. See below.

vector(Str) "unpacks" a structure Str, returning all the elements in its non-structure components as a vector. All the non-structure components must be of the same type, REAL, CHARACTER or LOGICAL.

Functions used with structures

ncomps(Str) returns the number of (top level) components in structure Str. See also ncomps().

compnames(Str) returns a CHARACTER vector of the names of the (top level) components in structure Str. See also compnames().

isstruc(x) returns True when x is a structure. More generally, isstruc(x1, x2, ...) returns a LOGICAL vector whose elements are True or False depending on whether the corresponding argument is a structure.

equal(str1, str2) compares two structures, returning True only when all their components and subcomponents are equal. See equal().

Assignment to components

Str[J] <- x, where J is a legal vector of subscripts and x is a variable, often a structure with ncomps(x) = number of components selected by J, replaces elements of an existing structure Str. See topics 'subscripts' and 'assignment'.

Structure arguments to functions

Many functions accept structures as arguments, including describe(), sum(), prod(), max(), and min(), and all the transformations. Each component of the result is obtained as the function of the corresponding component argument. For example, sum(structure(x,y)) is the same as structure(x:sum(x),y:sum(y)).

Binary and unary operators with structures

Both binary (for example, +, *, %*%) and unary (-, +, !) operators accept structures as operands. Most of these produce a structure of the same shape as the argument(s) or operands(s). For example, structure(a1,a2) %*% structure(b1,b2) = structure(a1 %*% b1,a2 %*% b2).

In addition, if one operand of a binary operation is a structure and the other is not, the result is a structure, each of whose components is computed by combining a component of the structure argument with the other argument. For example, 3*structure(a,b,c) is the same as structure(a:3*a,b:3*b, c:3*c), and structure(a,b) %C% c, is the same as structure(a:a %C% c,b:b %C% c).

Note: You cannot use a structure as an operand to %/% or %\%. See topic 'matrices'.

Extracting components

You can extract components of structure Str using subscripts instead of a name. For example, if Stats is structure(mean:xbar, var:s_sq), 'Stats[1]' and 'Stats[2]' are equivalent to 'Stats$mean' and 'Stats$var', respectively.

If there are more than one component with the same name or if the structure name is not a legal variable name, this is the only way to extract the component. For example Str <- structure(dim(x)[1], dim(x)[2], sqrt(x)) creates a structure with three components, named NUMBER, NUMBER, and MATRIX, but Str$NUMBER retrieves only the first component and Str[2] must be used to retrieve the second.

Usually it is preferable to name components using keywords. For example, Str <- structure(m:dim(x)[1],n:dim(x)[2],data:sqrt(x)) creates a structure with components named 'm', 'n', and 'data'. You can also name components using keyword 'compnames' on structure(), strconcat() and split().

Str[NULL] is NULL. See topic 'NULL'.

More generally, in an expression of the form Str[J], J may be a vector of positive integers, a vector of distinct negative integers, or a LOGICAL vector of length ncomps(Str). Str[J] is a structure whose components are the components of Str selected by J in the same way elements of a vector vec would be extracted by vec[J]. If J selects only one component, Str[J] is that component. If J selects no commponents (all F's or a full set of negative values), STR[J] is NULL. See topic 'subscripts' for how such a vector of subscripts is interpreted.

Alternative subscript form

An alternative way to extract components from a structure is Str[[J]], using double square brackets. This is equivalent to Str[J] when Str is a structure. However, when x is not a structure, the value of x[[1]] is x, not x[1], and, if J != 1, x[[J]] is an error. This feature can be useful in a macro when an argument can either be a non-structure variable or the first component of a structure.

Examples: When Stats is structure(mean:xbar, var:s_sq), Stats[vector(2,1)] is equivalent to structure(var:Stats[2], mean:Stats[1]), and Stats[vector(F,T)] is equivalent to Stats[2] or Stats$var.

Replacing components

You can replace one or more components of a structure by assigning to subscripts. For example, if Stats is as above, Stats[2] <- vector(1,3) will replace component 'var' by vector(3,10) without changing its name. Stats[[2]] <- vector(1,3) has the same effect.

```
You can also change a single named component by assignment.  For
example, if Stat is as above, Stats$var <- vector(1,3) is equivalent
to stats[2] <- vector(1,3).

You can change subcomponents at any depth <= 31 by assignment.  For
example, x is structure(a:run(3),b:structure(A:1,B:2,C:3)), all of x$b$C
<- PI, x[[2]]$C <- PI, x$b[[3]] <- PI or x[[2]][[3]] <- PI are
equivalent.

See topic 'assignment' for details on assignment to structure
components.

You can use changestr() to modify a structure.  See changestr() for
details.
```

2.343 subscripts

Usage:

```
   Ordinary subscripts: x[13], y[2,vector(4,5)]
   Negative subscripts: w[-vector(1,3,5)], z[-run(5),-1]
   Logical subscripts:  a[vector(T,T,F,F)], b[b < 0]
```

Keywords: syntax

```
                          Types of subscripts
Elements of vectors may be selected in several ways using a list of
integer or LOGICAL subscripts inside of square brackets.  The list must
be of one of three forms.  In the examples, z is assumed to be
vector(1,7,4,9,6).

A list of one or more positive integers.
  Examples: z[4] is 9 and z[vector(1,1,2,2,5,4)] is vector(1,1,7,7,6,9).

A list of negative subscripts. These indicate omission of the absolute
values of the subscripts.  If all subscripts are omitted the result is
NULL.  See topic 'NULL'.
  Examples: z[vector(-1, -3, -5)] is vector(7,9).
            z[-run(5)] is NULL

A LOGICAL vector with the same length as the source vector. The value is
a vector with elements corresponding to values of True.  If all the
elements are False, the result is a NULL variable, with no elements.
  Examples: z[z > 4] is z[vector(F,T,F,T,T)] = vector(7,9,6).
            z[z > 9] is z[vector(F,F,F,F,F)] = NULL

                       Erroneous use of subscripts
z[vector(1,2,?)] would be an error because MISSING values are not
allowed as subscripts.

z[vector(-3,-4,-3)] would be an error because there are duplicate
negative subscripts.
```

z[vector(-1,2)] would be an error because there are both positive and negative integers in the same vector of subscripts.

Use of subscripts with matrices and arrays

Elements of matrices may be accessed in an analogous manner to vectors, except that both dimensions must be specified, separated by a comma. An empty row or column specification implies all rows or columns. Suppose

q is the matrix matrix(run(6),3) = [1 4]
[2 5] .
[3 6]

Then q[3,1] is 3, and both q[vector(1,3),2] and q[-2,2] are the column vector 4,6 with dimensions 2 and 1. q[3,] is the row vector 3,6 and q[q[,1]>=2,] selects those rows of q for which column 1 is not less than 2, that is rows 2 and 3.

Elements of arrays may be accessed as for matrices, except that you must specify all subscripts. Again, an empty subscript specifies all legal values for that subscript.

With vectors, matrices or arrays, if any subscript is NULL or is non-selecting (all False or a complete set of negative subscripts), the result is a NULL variable.

Too many subscriupts

It is not an error to use more subscripts than there are dimensions as long as no extra subscript specifies an element greater than the first for that dimension. Extra trailing empty subscripts are ignored, so that for example, run(5)[-1,] and run(5)[-1] are identical. Extra subscripts that are 1 or T have the effect of increasing the number of dimensions.

For example, run(5)[-1,1] and run(5)[-1,T] are equivalent and result in a 4 by 1 matrix instead of a vector of length 4, but run(5)[-1,2] is an error. This is a useful feature when writing macros that may operate on vectors and matrices, or deal with both ANOVA and MANOVA SS. For example, SS[3,,] is equivalent to SS[3] after anova() and is also meaningful after manova() when SS is a 3 dimensional array.

Too few subscripts

If a is a matrix or array, and i is a vector, then a[i] is equivalent to vector(a)[i]. For example, matrix(vector(1,3,2,4,6,5),2)[vector(1,2,6)] yields vector(1,3,5).

Except in this case, it is an error when there are fewer subscripts than dimensions.

Subscripted factor

When a is a factor and I is an appropriate vector of subscripts, a[I] is a factor. For example, anova("{y[-1]} = {a[-1]}") carries out an analysis of variance omitting the first case. See topics factor(), anova() and 'models'. The "official" number of levels of a[I] is the

same as for a, even when max(a[I]) < max(a).

Matrix and Array Subscripts

You can also use a matrix as a single subscript in square brackets. If x is a vector, matrix or array with ndim dimensions, and Sub is a nrows by ndim matrix of positive integers, then x[Sub] is a vector of length nrows(Sub), whose i-th element x[Sub][i] is

```
   x[Sub[i,1],Sub[i,2],...,Sub[i,ndim]]
```

For example, if x is n by n, x[hconcat(run(n),run(n))] is equivalent to diag(x) and x[hconcat(run(n),run(n,1))] extracts the cross diagonal of x. There can be no MISSING values in Sub.

More generally, if Sub is an array of positive integers whose last dimension has length ndim, x[Sub] is an array with ndims(x[Sub]) = ndims(Sub) - 1 with i,j,...,k-th element x[Sub][i,j,...,k] =

```
   x[Sub[i,j,...,k,1],Sub[i,j,...,k,2],...,Sub[i,j,...,k,ndim]].
```

See select() for a way to select a the k[i]-th element in the i-th row of a matrix, when k is a vector of positive integers.

Use with structure

A structure may have a single scalar or vector subscript which selects components of the structure in the same way a scalar or vector subscript selects elements of a vector. See topic 'structures'.

Assignment to Subscripts

You can assign to subscripts as well as extract from them. For example, if y is a matrix with at least 3 rows y[run(3),] <- 0 sets the first 3 rows of y to 0. y[-vector(run(5),run(16,20)),] <- ? sets rows 6 to 15 and beyond 20 to MISSING. A statement like 'y <- x[2,] <- 3' sets row 2 of x to all 3's and y to a row vector of 3's with dim(y)[2] = dim(x)[2].

When y is a structure, y[J] <- x changes components of y. See subtopic 'assignment:"assignment_to_structure_components" for details.

Although you can use matrix subscripts in assignments to subscripts, you cannot use array subscripts where more than 2 dimensions exceed 1.

Cross references

See topic 'assignment' for details on assignment to subscripts, including assignment to subscripted components of a structure.

2.344 sum()

Usage:

```
sum(x [,squeeze:T] [,silent:T,undefval:U]), x REAL or LOGICAL or a
  structure with REAL or LOGICAL components, U a REAL scalar
sum(x, dimensions:J [,squeeze:T] [,silent:T,undefval:U]), vector of
  positive integers J
sum(x, margins:K [,squeeze:F] [,silent:T,undefval:U]), vector of
  positive integers K
sum(x1,x2,... [,silent:T,undefval:U]), x1, x2, ... REAL or LOGICAL
  vectors, all the same type.
```

Keywords: descriptive statistics

```
                                  Usage
sum(x) computes the sum of the elements of a REAL or LOGICAL vector x.

If x is LOGICAL, True is interpreted as 1.0 and False as 0.0 and sum(x)
is the number of elements of x that are True.

If x is a m by n matrix, sum(x) computes a row vector (1 by n matrix)
consisting of the sum of the elements in each column of x.

If x is an array with dimensions n1, n2, n3, ..., y <- sum(x) computes
an array with dimensions 1, n2, n3, ... such that y[1,j,k,...] =
sum(x[i,j,k,...], i=1,...,n1).  This is consistent with what happens
when x is a matrix.  Note: MacAnova3.35 and earlier produced a result
with dimensions n2, n3, ... .

sum(x, squeeze:T) does the same, except the first dimension of the
result (of length 1) is squeezed out unless the result is a scalar.  In
particular, if x is a matrix, sum(x,squeeze:T) will be identical to
vector(sum(x)), and if x is an array, sum(x,squeeze:T) will be identical
to array(sum(x),dim(x)[-1]).

sum(NULL) is NULL.  See topic 'NULL'.

sum(a,b,c,...) is equivalent to sum(vector(a,b,c,...)) if a, b, c,
... are all vectors.  They must all have the same type, REAL or LOGICAL
or be NULL.  sum(NULL, NULL, ..., NULL) is NULL.

sum(x, silent:T) or sum(a,b,c,...,silent:T) does the same but suppresses
warning messages about MISSING values or overflows.

If all the elements of a vector x are MISSING, sum(x) is 0.0.

sum(x, undefval:U), where U is a REAL scalar does the same, except the
returned value is U when all the elements of x are MISSING.

                          Keyword 'dimensions'
sum(x, dimensions:J [,squeeze:T] [,silent:T] [,undefval:U]) sums over
the dimensions in J = vector(j1,j2,...,jn) where j1, ..., jn are
distinct positive integers <= ndims(x).  Without 'squeeze:T', the result
has the same number of dimensions as x, with dimensions j1, j2, ..., jn
```

of length 1. With 'squeeze:T', these dimensions are removed from the result. The order of j1, j2, ... is ignored.

It is an error if max(J) > ndims(x) or if there are duplicate elements in J.

For example, if x is a matrix, sum(x, dimensions:2) computes the row sums as a nrows(x) by 1 matrix and sum(x, dimensions:2,squeeze:T) computes them as a one dimensional vector.

Keyword 'margins'

sum(x, margins:K [,squeeze:F] [,silent:T] [,undefval:U]) sums over the dimensions not in K = vector(k1, k2, ..., km), where k1, ..., km are distinct positive integers <= ndims(x). This computes marginal totals for the margins specified in K.

Without 'squeeze:F', only the dimensions in K are retained in the result. Otherwise the other dimensions are retained but have length 1. This is opposite from the default with 'dimensions:J'.

It is an error if max(K) > ndims(x) or if there are duplicate elements in K.

Structure argument

If x is a structure, sum(x [,dimensions:J or margins:K] [,squeeze:T or F] [,silent:T] [,undefval:U]) computes a structure, each of whose components is sum() applied to that component of x. All the components of x be of the same type, REAL or LOGICAL.

Examples

Examples:
If x is a n by m matrix

```
  Cmd> r <- x - sum(x)/sum(!ismissing(x))
```

computes the matrix of the residuals of x[i,j] from the column means. When there are no MISSING values, divide by nrows(x)

If x is a n by 4 by 5 array,

```
  Cmd r <- x - sum(x)/sum(!ismissing(x))
```

computes an array with r[i,j,k] = the residual of x[i,j,k] from the mean of all x[i,j,k] with the same values for j and k. That is, it treats x analogously to a 4 by 5 array of vectors of length n. See topic 'arithmetic'. When there are no MISSING values, divide by dim(x)[1].

If z is a vector of integers,

```
  Cmd> sum(z == run(min(z),max(z))')
```

computes a row vector giving the frequency distribution of the values in

```
z.

  Cmd> a # 2 by 2 by 3 array with labels
                  C1          C2          C3
  A1 B1            9           5           7
     B2            9          12          11
  A2 B1            4          11          10
     B2           11          15           9

  Cmd> sum(a,dimensions:2) # sum over dimension 2; 2 by 1 by 3 result
                  C1          C2          C3
  A1 (1)          18          17          18
  A2 (1)          15          26          19

  Cmd> sum(a,margins:vector(1,3),squeeze:F) # same as preceding
                  C1          C2          C3
  A1 (1)          18          17          18
  A2 (1)          15          26          19

  Cmd> sum(a,dimensions:2,squeeze:T) # sum over dim 2; 2 by 3 result
              C1          C2          C3
  A1          18          17          18
  A2          15          26          19

  Cmd> sum(a,margins:vector(1,3)) # same as preceding
              C1          C2          C3
  A1          18          17          18
  A2          15          26          19

                            Cross references
See also prod(), tabs()
```

2.345 svd()

Usage:

```
   svd(x [,left:T or F,right:T or F, all:T, maxit:N, nonconvok:T]), x a
     REAL matrix, N > 0 an integer
```

Keywords: matrix algebra

```
                                    Usage
svd() computes some or all of the parts (singular values, left singular
vectors and right singular vectors) of the singular value decomposition
(SVD) of a matrix.

svd(x) computes the vector of singular values in order of decreasing
size, of the m by n REAL matrix x.  When m < n, the last n - m elements
of values are 0.

svd(x,left:T) computes the singular values and the m by n matrix of
orthonormal left singular vectors in a structure with components
```

'values' and 'leftvectors'. When m < n, the last n - m elements of values are 0 as are the last n - m columns of leftvectors.

svd(x,right:T) computes the singular values and the orthogonal n by n matrix of right singular vectors in a structure with components 'values' and 'rightvectors'. When m < n, the last n - m elements of values are 0 and the last n - m columns of rightvectors are orthonormal vectors orthogonal to the first n columns, but are otherwise arbitrary.

svd(x,all:T) and svd(x,left:T,right:T) both compute a structure with components 'values', 'leftvectors', and 'rightvectors'

svd(x,all:T,vals:F) computes a structure with components 'leftvectors' and 'rightvectors' only. Other combinations of all:T and other keywords are possible and do what you would expect.

Propagation of labels

If x has labels, the row labels of the matrices of left and right singular vectors are the row and column labels of x, respectively. The column labels are numerical. The vector of singular values is unlabelled. See topic 'labels'.

If l and r are the matrices of left and right singular vectors and s is the vector of singular values then they satisfy (except for rounding error) l %*% dmat(s) %*% r' = x and l %c% l = r %c% r = I-sub-m (when m < n, only the upper left m by m block of l %c% l is I-sub-m).

Non-convergence

It is possible for the algorithm used by svd() not to converge, although it rarely happens. When it happens, the message

ERROR: singular value algorithm in svd() did not converge

is printed. Keywords 'maxit' and 'nonconvok' may be helpful in this situation.

svd(x [,keywords], maxit:N), where N > 0 is an integer, computes the singular value decomposition, but sets the maximum number of iterations in the algorithm to N. The default value is 30. By using N > 30, this may allow you to compute the SVD.

svd(x [,keywords] ,nonconvok:T) does the same, except failure to converge is not an error. When convergence does not occur, no message printed and NULL is returned. You can use this in a macro to make it possible to recover from failure to converge, perhaps by invoking svd() again using 'maxit' to increase the number of iterations.

Cross references

See also eigen(), eigenvals(), releigen(), releigenvals().

2.346 swp()

Usage:

```
swp(x, n1 [, n2, ...] [, quiet:T, diag:d, tolerance:tol, keepswept:T]),
  x a REAL matrix, n1, n2, ... positive integers, or vectors of positive
  integers, tol > 0 a REAL scalar, d a REAL vector with positive
  elements
```

Keywords: matrix algebra, glm

```
                                  Usage
swp(x,Cols) uses a form of the Beaton SWP operator on the REAL matrix x
to compute a REAL result of the same size.  If x is m by n, Cols should
be a vector whose elements are positive integers <= min(m,n).

A single SWP of x on row and column k produces a matrix y of the same
size as x with

     y[i,j] = x[i,j] - x[i,k]x[k,j]/x[k,k], for i and j not equal to k
     y[i,k] = x[i,k]/x[k,k], for i not equal to k
     y[k,j] = -x[k,j]/x[k,k], for j not equal to k
     y[k,k] = 1/x[k,k]

The element x[k,k] is sometimes called a "pivot".

swp(x,Cols) does successive SWPs of x on Cols[1], Cols[2], ... rows and
columns.  If, attempting to SWP row and column M, abs(pivot) is found to
be too small in comparison with the abs(x[M,M]), the message
  WARNING: tolerance failure pivoting column M; not swept
is printed and that SWP is skipped.

The threshold for finding tolerance failure may be modified by keyword
'tolerance'; see below.

If x is n by n and non-singular, swp(x,run(n)) computes the inverse of x
(it can fail for certain non-positive definite but invertible matrices).

swp(x,Cols1, Cols2, ...), where Cols1, Cols2, ..., are vectors of
positive integers is equivalent to swp(x,vector(Cols1, Cols2, ...)).
For example, swp(cp,1,2,3,4) is equivalent to swp(cp,run(4)).

swp() can be very useful in regression and analysis of variance.

                            Keyword 'tolerance'
swp(x,Cols,...,tolerance:Tol), where Tol is a small REAL scalar does the
same, but Tol is used in the test for what constitutes a small pivot.
Column M will be skipped if abs(pivot) < Tol*abs(x[M,M]), where pivot is
the value of x[M,M] just before a SWP of row and column M is attempted.

The default value of Tol is 10^(-12).

                         Keywords 'quiet' and 'diag'
swp(x,Cols,...,quiet:T) does the same, but no warning message is printed
when a too small pivot is found.
```

```
swp(x,Cols,...,diag:d [,quiet:T]), where d is a REAL vector, does the
same, but the pivot for row and clumn M is compared with abs(d[M]) to
test whether a row and column will be swept.  The length of d must
always be min(nrows(x), ncols(x)).  This feature allows the sequence,
say,
  Cmd> d <- diag(x); x1 <- swp(x,1); x1 <- swp(x1,2,diag:d)
to be completely equivalent to
  Cmd> x1 <- swp(x,1,2)
Without diag:d, the comparison element for swp(x1,2) would be x1[2,2]
instead of x[2,2].

                          Keyword 'keepswept'
swp(x,Cols,..., keepswept:T [,tolerance:Tol, ,diag:d, quiet:F]) does the
same, except that the result is a structure with components 'matrix' and
'sweptcols'.  Component 'matrix' is the swept version of x.  Component
'sweptcols' is a vector of integers, with abs(sweptcols[i]) the number
of the i-th row and column swept.  sweptcols[i] < 0 if and only if row
and column abs(sweptcols[i]) failed the tolerance check.  No warning
message is printed unless 'quiet:F' is an argument.

The use of 'keepswept:T' may allow you to compute a generalized inverse
of a singular square matrix.

  Cmd> tmp <- swp(a, run(nrows(a)), keepswept:T)

  Cmd> b <- tmp$matrix; J <- (-tmp$sweptcols)[tmp$sweptcols < 0]

  Cmd> if(!isnull(J)){b[J,] <- b[,J] <- 0;;}

Now, even if a is singular, a %*% b %*% a should be a within rounding
error and b %*% a %*% b should be b within rounding error.

                          Cross references
See also solve(), qr().
```

2.347 syntax

Keywords: syntax, control, general, character variables, logical variables, variables, missing values, null variables

```
                      Commands and Statements
A MacAnova command or statement is a sequence of characters typed in at
the keyboard followed by ';' or '<cr>' (the RETURN or ENTER key).  The
first character should not be '!' unless it is a "shell escape" (see
shell()).

Typical commands or statements are 'x <- vector(1.2,3.1,5.3,2.4)<cr>'
(assign the vector (1.2,3.1,5.3,2.4) to variable x), 'print(x)<cr>'
(print the value of variable x), 'regress("y=x1+x2+x3")<cr>' (compute a
regression of variable y on variables x1, x2 and x3), or 'y <-
```

3*x^2<cr>' (assign to variable y the value of 3 times x squared).

You may put several commands or statements separated by ';' on a single line terminated by <cr>. An example would be

```
  Cmd> regress("y=x"); plot(x,RESIDUALS)<cr>
```

When you press '<cr>', but not before, all the commands or statements in the line are executed one after the other.

One type of statement consists solely of a number or an algebraic expression involving numbers or variable names. Examples are '17.3', '3*y', 'sqrt(4+cos(-1.32))' and '3*log(640320)/sqrt(163)'. The only effect of this type of statement is to print out the value. This allows MacAnova to be used as a symbolic calculator. Here are some examples:

```
  Cmd> 17.3
  (1)         17.3

  Cmd> 3*log(640320)/sqrt(163)
  (1)       3.1416

  Cmd> 3*y
  (1)       25.584       28.817       30.636

  Cmd> "Hello!"
  (1) "Hello!"
```

Below for brevity both commands and statements are usually just called 'commands'.

Side Effects of Commands

Some commands have "side effects" such as printing tables or creating variables containing the results of computation. For example, although anova() returns only a NULL value (see topic 'NULL'), it has side effects, namely the printing of an analysis of variance table and creation of several named variables such as RESIDUALS, SS and DF.

Variables

Data are stored in permanent or temporary "variables" with names of up to 12 characters. Typical names might be 'x1', 'data', 'weight' or 'time_of_day'.

The names of permanent variables start with a letter or '_', while the names of temporary variables start with '@' followed by a letter or '_'. The remainder of a name consists of letter, digits or '_'. Names are case sensitive (for example, 'residuals' is a different name from 'Residuals'. Some variables are "invisible". See topic 'variables:"invisible"'.

Typically you will select names that are relevant to the problem such as 'weight', 'residuals', or 'depv'.

A name may not be the same as a command name. For instance, you can't use 'rep' as a variable name because there is a command 'rep'.

Typing the name of a variable that is not "invisible" prints out its value.

See topic ’variables’ for more details.

Command Line

The ’command line’ consists of all the commands that are executed by a single <cr>. The command line follows the standard MacAnova prompt "Cmd> ", and may, in fact be continued on several actual lines; see next paragraph.

When a command line becomes longer than one physical line, you can just keep typing as the characters wrap around to the next line. Or, you may continue a line by typing ’\<cr>’ and continuing on the next line. (Nonwindowed versions will issue a continuation prompt "More> ".) If the break is at the end of a command, you need to type ’;’ before ’\<cr>’.

Correcting mistakes

If you make a mistake, you may backspace to erase the error. On windowed versions, you can move the cursor with the mouse at any time to make corrections anywhere in the command line.

Compound Commands

A "compound command" is a sequence of one or more commands inside curly brackets ’{’ and ’}’. For example

```
{i <- i+1; plot(x[,i],y[,i])}
```

is a compound command. Compound commands are used primarily with control constructs such as ’if’, ’for’ and ’while’.

Once you have started typing a compound command by pressing ’{’ the command line is not executed by <cr> before you type a matching ’}’. Below, ’{...}’ represents an arbitrary compound command.

In a compound command, you can type <cr> at any place where a semicolon ’;’ would be appropriate; the compound command will not be executed until you have pressed <cr> after the closing ’}’.

You may nest compound commands ({...{...}...}). None of them is executed until you terminate the outermost compound command with ’}’ and press <cr>.

A common mistake is to fail to terminate a compound command with ’}’. MacAnova appears to be "hung up". Actually it is just waiting for you to finish what you started. Until the compound command is complete, MacAnova has no way to recogonize that you are through typing the command line. You can either type ’}<cr>’ or press the interrupt key and start over.

MacAnova as a Language

The organization of MacAnova commands can be considered as a "functional language," in the sense that the components of commands are functions or operators which take arguments or operands as inputs and may compute

values as outputs.

The arguments of (inputs to) a named command or macro are separated by commas and enclosed in '(' and ')', as in 'print(x,y,z)'. Some named commands or macros require no arguments. In this case you just put '()' after the name, as in 'getoptions()'. In MacAnova documentation, including help() output, a named command or macro is referred to by its name followed by '()', for example, 'log10()' and 'getmacros()'.

Values of commands and compound statements

All commands, including compound commands (see below) have a value. For example, 'sqrt(3)' has the value 1.73205080756888 and '4*atan(1)' has value 3.14159265358979.

Some commands such as print() and regress() have values which are NULL (see topic 'NULL'). In addition, an explicitly empty command, ';;' or '();' has a NULL value. For obvious reasons, you can't use commands with NULL values in algebraic expressions or comparisons or in other contexts require data.

A named command that returns a non-NULL value is often referred to as a "function."

A few functions (for example getseeds()) return an "invisible" value that can be assigned but is not automatically printed when it is not assigned. See topic 'variables:"invisible"'.

The value of a compound command is the value of the last command in the curly brackets. If its last command is empty (';;' or '();'), a compound statement has a NULL value. You can use a compound command which has a non-NULL value in an expression. For example,

```
  Cmd> xbar <- {print(x); sum(x)}/n
```

prints x and computes xbar = sum(x)/n, because the value of '{...}' is the value of 'sum(x)'.

Output from functions may be arithmetically combined (for example, 'cos(x) + 3*sin(y)') and the value of (output from) one function may be an argument to (input for) another, (for example, 'cos(sqrt(x+y))').

See also topics 'arithmetic' and 'transformations'.

Conditional Execution and Looping

There are several syntax elements that you can use to control which commands are executed and in what order. Here is a brief summary:

```
 Conditional execution
   if(Logical){...}
   if(Logical){...}else{...}
   if(Logical){...}elseif(Logical){...}else{...}
 Looping
   for(Var,Vector){...}
   for(Var,start,end[,increment]){...}
   while(Logical){...}
```

```
 Escaping from a loop
   break, break n, breakall, breakif()
 Skipping to the end of a loop
   next, next n
 Leaving a macro, possibly returning a value
   return
```

Logical must be a LOGICAL scalar variable or expression such as 'i < 4'. Vector must be a REAL vector and start, end and increment must be REAL scalars. See below for types of data.

The first '{' following 'if(...)', 'for(...)', and 'while(...)' must be on the same line.

See topics 'if', 'for', 'while', 'break' and breakif() for more details.

Types of Data

There are several types of data, including REAL, LOGICAL, CHARACTER, GRAPH, STRUCTURE and NULL. In certain output, LOGICAL, CHARACTER, and STRUCTURE are abbreviated as LOGIC, CHAR, and STRUC, respectively.

REAL data

REAL data elements are numbers and can be entered from the keyboard, read from a data file, or computed by arithmetic expressions, transformations and other functions or macros. You enter numbers as integers without a decimal point (-321), as decimal numbers (31.4159), or using exponential notation (7.2e+9 = 7.2*1000000000).

You can separate digits by '_' for clarity, but not after 'e'. Thus 123_456.789_1 is equivalent to 123456.7891. See topic 'numbers' for more information.

A missing value is represented by a special internal code named MISSING. When entering data at the keyboard, you enter a missing value as '?'. On output, a missing value usually printed as "MISSING" but you can specify a different output coding, say "?", by setoptions(missing:"?") (see setoptions(), subtopic 'options:"missing"').

Virtually all commands and operations pay at least token attention to MISSING, although at present nothing is done that is more complicated than omitting cases with MISSING or setting to MISSING a result item corresponding to a MISSING input item.

You can do arithmetic on REAL data using the arithmetic operators '+', '-', '*', '^' or '**', and '%%' (for example, a * (b + c)). See topic 'arithmetic'.

You can compare REAL data items using comparison operators '<', '>', '==', '!=', '<=', and '>='. Except for '==' and '!=', a comparison with a MISSING value yields a MISSING LOGICAL value. See topic 'logic'.

LOGICAL data

LOGICAL data have values limited to True, False and MISSING and are

entered and printed as 'T' (True) or 'F' (False). Comparison expressions (for example 'a < 3') generate LOGICAL data as values. See topic 'logic' for detailed information.

You can combine LOGICAL variables and expressions using logical operators '&&', '||' and '!'. For example, '(x > 3) && !(x > 5)' has value True if and only if both (x > 3) and (x <= 5) are True, that is if 3 < x <= 5. Similarly, '(x > 0) || (abs(x) == 3)' has value True if and only if x > 0 or |x| = 3 (or both).

You can use LOGICAL data in arithmetic expressions and comparisons, with True and False being translated as 1 and 0, respectively. For example, F*T is 0, 2*T is 2, and F < T is True . Logical variables can also be used in place of REAL variables as argument to some, but not all functions such as sum(), prod() and max().

CHARACTER data
A CHARACTER data element consists of a sequence of characters, that is letters, numbers, punctuation or anything else that can be typed. It is sometimes called a "string." When entering CHARACTER data you must enclose each string in double quotes as in

```
Cmd> greetings <- "Hello!"
```

The opening and closing double quotes are not part of the string. You include a double quote in a string by prefixing ('escaping') it with '\'. For example,

```
Cmd> a <- "He said, \"Hello\""
```

assigns the string 'He said, "Hello"' to variable a. You can include characters '\', newline and tab by '\\', '\n' and '\t', respectively, using a convention borrowed from Unix/Linux. For example,

```
Cmd> b <- "1\t2\t3"
```

assigns to b the string consisting of '1', '2', and '3' separated by tab characters.

Importance of closing quote

Once you have started typing a CHARACTER string with '"', MacAnova interprets everything, including <cr>, up to the next (non-escaped) '"' as part of the string. It does not recognize the command line to be complete until you have typed the closing '"'.

A common mistake is to forget to terminate a string with '"'; MacAnova appears to "hang", doing nothing. You need to type a closing '"' and terminate the line or press the interrupt key and start again.

Special characters in strings

You can include in a string any character, even one you cannot type directly, using the so called escaped octal representation of its internal (ASCII) code. For example, since 1*8 + 5 = 13, '15' is the octal (base 8) representation of 13 and "\15" or "\015" is the character (usually CR) with code 13. Similarly, because '117' is the octal representation for 1*8*8 + 1*8 + 7 = 79, the ASCII code for 'O', "\117" is equivalent to "O". Also acceptable are escaped hexadecimal representations of the form "\xmn", where m and n are hexadecimal digits (0 - 9, and a - f or A - F). For example, "\117" and "\x4f" are both

equal to "O" (4*16 + 15 = 79).

Comparison of character data

You can compare CHARACTER data items using comparison operators '<', '>', '==', '!=', '<=', and '>='. The ordering of letters is based on their ASCII representation with "A" < "B" < ... < "Z" < "a" < ... < "z". For example, '"A" < "B"', '"a" < "B"' and '"foo" == "bar"' have values True, False and False, respectively. See topic 'variables' for more detail.

Graph variables

A GRAPH variable encapsulate all the information needed to draw a graph or other plot. You can display the plot in GRAPH variable GraphVar by 'showplot(graphVar)'. See topic 'graphs' for more information.

Structure variables

A structure is made up of one or more named data components that may be of any type. See topic 'structures' for details.

Null variables

A NULL variable contains no data. See topic 'NULL' for details.

Assignment of values to variables
Assignment of Values to Variables

You assign values to a variable using the left pointing arrow '<-' made up of the two characters "less than" and "minus". For example, 'foo <- 5' assigns the value 5 to the variable foo. If foo did not previously exist, it is created; otherwise, its previous value is discarded and foo is re-defined. An expression of the form 'x <- 3', say, is always interpreted as 'x <- 3' rather than as 'x < -3'. If you want the latter, be sure to put a space before '-3'.

You can string several assignments together. For example,

```
  Cmd> a <- b <- 1
```

is interpreted from right to left, first assigning 1 to b and then assigning the new value of b to a.

See topic 'assignment' for information about the value of an assignment statement, assignment to subscript-selected elements of a variable, and assignments to components of a structure, and topic 'arithmetic' about arithmetic assignment operators <-+, <--, <-*, <-/, <-%% and <-^.

Organization of Data

REAL, LOGICAL, or CHARACTER data may be organized as scalars, vectors, matrices, or arrays. The transpose of a matrix or array x may be typed as x' or as t(x). Matrix multiplication operators (applicable only to REAL data) are %*%, %c%, and %C%.

See also topics 'vectors', 'matrices', vector(), matrix(), array(), dim(), ndims(), ismatrix(), nrows(), ncols(), 'transpose'.

Data may have vectors of labels for each coordinate. In particular, matrices may have row and column labels. Labels propagate through

operations and functions in a fairly sensible way. Labels are primarily used in output . See topic 'labels' for information.

Subscripts

You refer to an element or set of elements of a vector, matrix, or array by using subscripts -- numbers or variables enclosed in square brackets '[...]' immediately following the variable name.

For example, 'x[3,4]' is the element in row 3 and column 4 of matrix x, 'x[vector(1,3,5),4]' consists of rows 1, 3 and 5 of column 5 of x, 'x[3,]' is row 3 of x and 'x[,4]' is column 4 of x.

You can assign values to subscripted elements, as in 'x[3,4] <- 17' or 'x[3,] <- 5'. See topics 'matrices', matrix(), array(), 'transpose', 'subscripts', 'assignment'.

When x is a structure, you can use x[[J]] in place of x[J]. When x is not a structure, x[[1]] is the same as x and x[[J]] is illegal unless J = 1.

Combining Data Items

There are commands to combine several data items into a larger vector, matrix, or structure. See vector(), hconcat(), vconcat(), structure(), strconcat().

Keyword Phrases

Some command arguments can be "keyword phases" in the form 'keyword:value'. For example, on several commands that write numbers, the argument 'nsig:8' specifies that up to 8 significant digits are to be printed and 'format:"18.13g"' specifies a format with width 18 and 13 significant digits.

LOGICAL keyword phrases like 'keep:T' or 'coefs:F' are particularly common. These enable (T) or suppress (F) alternative actions of a command. See topic 'keywords' for details.

Indirect Reference to Variables and Constants

<<String>>, where String is a quoted string or CHARACTER variable whose value is the name of a variable, refers indirectly to that variable. For example, '<<"cos">>(PI/4)' and '<<"E">>+3' are equivalent to 'cos(PI/4)' and 'E+3', and, after the command 'a <- vector("x1","x2","x3")' creates a CHARACTER vector a, '<<a[2]>> <- 3' is equivalent to 'x2 <- 3'.

The following line creates variables x1, x2, and x3 from the columns of matrix x.

```
  Cmd> for(i,run(ncols(x))){<<paste("x",i,sep:"")>> <- x[,i];;}
```

See topics paste(), 'for' and 'subscripts'.

Indirect reference works even with keywords and structure component names. For example, setoptions(<<"nsig">>:5) is equivalent to setoptions(nsig:5) and, when Str is a structure with a component named

```
'x', Str$<<"x">> is equivalent to Str$x.

If String is "?", "T", "F", or "NULL", the value of <<String>> is
MISSING, True, False, or a NULL variable.  If String represents a
number, <<String>> is the value of the number.  For example
10*<<"-123.456">> has value -1234.56.  If String represents a CHARACTER
scalar of the form "\"string without non-escaped quotes\"", <<String>>
is equivalent to "string without non-escaped quotes".  For example,
"ABCD" == <<"\"ABCD\"">> is True.

More generally, String can contain one or more MacAnova expressions or
commands.  The commands are executed and the value of <<String>> is the
value of the last command in String.  In this case, <<String>>
essentially does the same as evaluate(String).  There are some
restrictions on what commands can appear in String.  See evaluate().
  Cmd> <<"sqrt(2*PI)">>
  (1)       2.5066

On Mac OS 9, you can use Option+\ and Option+| instead of << and >>,
respectively.

                      Re-executing a Command Line
Just before MacAnova accepts a new command line, the immediately
preceding command line is saved as macro LASTLINE.  This allows you to
re-execute the immediately preceding command line by typing LASTLINE()
Alternatively, you can used pre-defined macro redo(): redo() makes a
copy of LASTLINE as macro REDO() and then executes REDO().  You can
subsequently re-execute the same line one or more additional times by
typing REDO().  You cannot use redo() on two successive lines.  See
topics 'macros', redo().

On machines where macro edit() is defined, you can edit the immediately
preceding line and re-execute the modified version by
  Cmd> REDO <- edit(LASTLINE); REDO()
See topic edit().

                            Cross references
See also topics 'comments', 'interrupt', 'quitting'.
```

2.348 t()

Usage:

```
t(x) is same as x', x a vector, matrix, array or structure
t(x,J), x an array or structure, and J a vector containing a permutation
  of run(p)
```

Keywords: matrix algebra, operations

```
                                  Usage
t(x), where x is a vector, matrix, array or a structure, is equivalent
to x'.  See topic 'transpose'.
```

```
y <- t(x, J), where x is an array with p dimensions and J is an integer
vector containing a permutation of run(p), sets y to a copy of x with
the dimensions permuted.  y is an array with dimensions dim(x)[J] and,
when I is a vector of p integers with 1 <= I[j] <= dim(x)[J[j]],
  y[I[1],I[2],...,I[p]] = x[I[J[1]], I[J[2]],..., I[J[p]]].

For example, when x has three dimensions, and J = vector(3,1,2),
dimension 1 of y is dimension 3 of x, dimension 2 of y is dimension 1 of
x, and dimension 3 of y is dimension 2 of x.

t(x,j1,j2,...,jk) is equivalent to t(x,vector(j1,j2,...,jk)), where j1,
..., jk are scalars or vectors.

When p >= 2, t(x,run(p,1)) is equivalent to t(x) and x'.  When p = 1,
t(x) and x' are 2 dimensional with one row, but t(x,run(2,1)) is
illegal, since length(J) = 2 != ndims(x) = 1.

                          Structure argument
When x is a structure, all of whose non-structure components are arrays
with the same number of dimensions, t(x,J) is a structure of the same
shape and with the same component names as x, with each non-structure
component a copy of the corresponding component of x with dimensions
permuted.

                               Examples
Examples:
  Cmd> x <- array(run(24),4,3,2) # x has dimensions 4, 3, 2

  Cmd> y <- t(x,vector(3,1,2)) # y has dimensions 2, 4, 3

  Cmd> i1 <- 2; i2 <- 2; i3 <- 3;vector(y[i1,i2,i3],x[i2,i3,i1])
  (1)          22          22

  Cmd> i1 <- 1; i2 <- 4; i3 <- 2;vector(y[i1,i2,i3],x[i2,i3,i1])
  (1)           8           8

                          Cross references
See also topics array(), 'vectors', 'matrices', 'subscripts'.
```

2.349 t2int()

Usage:

```
   t2int(x1,x2,Coverage [, pooled:F]), x1 and x2 REAL vectors or matrices
     with ncols(x1) = ncols(x2), 0 < Coverage < 1
```

Keywords: probabilities, descriptive statistics, confidence intervals

```
                               Coverage
t2int(x1,x2,Coverage), where x1 and x2 are REAL vectors, computes a (two
```

```
sided) t confidence interval for mu1 - mu2 with confidence coefficient
Coverage, where mu1 is the population mean of the data in x1 and mu2 is
the population mean of the data in x2.  A pooled estimate of the
standard error of the difference is used.  This assumes equal variances.

Coverage must be between zero and one.  The value is a vector of length
2 giving the lower and upper endpoints of the interval.

t2int(x1,x2,Coverage,pooled:F) computes a confidence interval for
mu1-mu2 based on the unpooled estimate sqrt(s1^2/n1+s2^2/n2) of the
standard error and Satterthwaite's approximate degrees of freedom.  It
does not assume equal variances.

If there are any missing values, they are omitted from the computation
and an informative message is printed

                            Matrix arguments
When x1 and x2 are REAL matrices with ncols(x1) = ncols(x2) = M,
t2int(x1,x2,Coverage [,pooled:T]) computes confidence intervals for each
column.  The result is a 2 by M matrix with the lower and upper limits
in rows 1 and 2, respectively.

                            Cross references
See also tint(), tval(), and t2val().
```

2.350 t2val()

Usage:

```
t2val(x1,x2 [,df:T or pooled:F]), x1 and x2 REAL vectors or matrices
  with the same number of columns
```

Keywords: probabilities, descriptive statistics, comparisons

```
                                  Usage
t2val(x1,x2) computes the two-sample Student's t statistic for testing
the null hypothesis that the data in REAL vectors x1 and x2 have the
same population means mu1 and mu2.  The usual pooled estimate of
variance is used in computing the standard error of the difference of
means.  This assumes that the two variances are equal.

t2val(x1,x2,df:T) computes a structure with two components, 't' and
'df', containing the t-statistic and its degrees of freedom,
respectively.

t2val(x1,x2,pooled:F) computes a structure with components 't' and 'df'.
't' contains the t-statistic computed using the unpooled estimate
sqrt(s1^2/n1 + s2^2/n2) of the standard error of the difference of
means.  'df' contains Satterthwaite's approximate degrees of freedom.  A
test of H0: mu1 = mu2 based on t and df does not assume the two
populations have the same variance.
```

```
When there are missing values, they are omitted from the computation and
an informative message is printed.

                      Testing H_0: mu1-mu2 = delta
When the null hypothesis is that mu1 - mu2 is some specific value other
than zero, say delta, then t2val(x1-delta,x2) will produce the correct t
value for that null hypothesis.  For example, if delta is -3, use
t2val(x1-(-3),x2 [,df:T or pooled:T]).

                            Matrix arguments
When x1 and x2 are REAL matrices with ncols(x1) = ncols(x2) = M, t2val()
computes two-sample t-statistics for each column separately.
t2val(x1,x2) returns a vector of length M.  t2val(x1,x2,df:T) and
t2val(x1,x2,pooled:F) return a structure with components 't' and 'df',
with each component a REAL vector of length M.

                                 P values
P values may be computed using cumstu() or twotailt(), for example, by
  Cmd> result <- t2val(x1,x2,df:T);twotailt(result$t,result$df)
or
  Cmd> result <- t2val(x1,x2,pooled:F);twotailt(result$t,result$df)

                            Cross references
See also topics tval(), tint(), t2int(), cumstu(), and twotailt().
```

2.351 tabs()

Usage:

```
tabs(y [,a,b,...] [, mean:T, var:T, covar:T, count:T or n:T, stddev:T,\
  min:T, max:T, sum:T, prod:T]), y REAL vector or matrix, a, b,
  ... factors, LOGICAL or positive integer vectors
tabs(y [,a,b,...], all:T [,mean:F, var:F, ...] [,covar:T])
tabs(,a,b,...[,count:T or n:T])
```

Keywords: categorical data, descriptive statistics

```
                                   Usage
tabs(y,a,b,c,...) computes summary statistics from data in REAL vector
or matrix y for each "cell" defined by grouping variables a, b, c, ... .
The values in a, b, c, ... must be positive integers < 32768, considered
as category levels, or MISSING.  In particular, a, b, c, ... can be
factors.  See factor().

By default, the output is a structure with components 'mean' containing
cell means, 'var' containing cell variances, and 'count' containing cell
sample sizes, but this can be modified by keywords.

The values in each component, including 'count', are based on the
non-MISSING elements of y.  When a cell has no non-MISSING values, the
count is 0 and any statistics computed are MISSING.
```

When y has MISSING elements, a warning message is printed. When a factor has elements that are MISSING, the corresponding value of y is ignored and a warning message is printed.

When there are nFact grouping variables and y is a vector or a matrix with 1 column, each component of the output is an array with nFact dimensions. When y is a matrix with nv = ncols(y) > 1, each component is an array with nFact + 1 dimensions, the last of which is nv.

tabs(y,a,b,c,..., silent:T) does the same except warning messages are suppressed.

LOGICAL and CHARACTER factors

Any grouping variable can be a LOGICAL vector, with False and True corresponding to levels 1 and 2, respectively. For example, tabs(y, x1<=0, x2<=0) cross tabulates y according to the signs of x1 and x2. See topic 'logic'.

When you want to cross tabulate data according to the values of either non-integer REAL or CHARACTER vectors, use pre-defined macro makefactor() to create corresponding factors. See topic makefactor().

Dimensions of output

When grouping variables a, b, c, ... are simple vectors and not factors created by factor(), the length of each dimension is the maximum value of the grouping variable.

When a grouping variable is a factor, the size of its dimension is the "official" number of levels, even if the last level is empty, as can be the case when it has the form f[J], where f is a factor and J is a vector of subscripts. See subtopic 'subscripts:"subscripted_factor"'.

Use without factors

tabs(y), with no factor arguments, computes statistics for y as a whole. Each component of the result is a scalar or a vector of length ncols(y). This usage is similar to describe(y) except fewer statistics are computed and the sample size component is called 'count' rather than 'n'. See describe().

Use with no response argument

tabs(NULL,a,b,...) or tabs(,a,b,...), with no first argument, just computes the counts in each cell, returning a vector, matrix or array. See topic 'NULL'.

Controlling which quantities to compute

When one of the keyword phrases 'mean:T', 'var:T', 'count:T', 'stddev:T', 'min:T', 'max:T', 'sum:T', 'prod:T' or 'covar:T' is an argument, tabs() returns a REAL vector, matrix or array containing the means, variances, counts, standard deviations, minima, maxima, sums, products or covariance matrices of the values in each cell. When more than one of these keyword phrases are arguments, the result is a structure with component names matching the keywords.

```
With 'covar:T', there can be no MISSING values in y and the result or
component 'covar' of the result is an array with nfact + 2 dimensions,
the last two of which are both ncols(y).

                                Examples
Examples:
  tabs(y,a,b,mean:T,var:T,count:T) is equivalent to tabs(y,a,b).
  tabs(y,mean:T,var:T,count:T) is equivalent to tabs(y).
  tabs(y,a,b,mean:T) computes only a matrix of cell means
  tabs(y,a,b,mean:T,stddev:T,min:T,max:T) computes a 4 component
    structure of means, standard deviations and extremes for each cell
  tabs(y,a,b,sum:T,prod:T) computes a 2 component structure whose
    components are matrices of sums and products of all elements in
    each cell
  tabs(y,mean:T,stddev:T,silent:T) computes statistics for y as a whole;
    no warning message is printed when y has MISSING elements
  tabs(y,a,b,mean:T,count:T,covar:T) computes 3 dimensional arrays of
    cell counts and means and a 4 dimensional covariance matrix array
  tabs(y,covar:T) computes the ncols(y) by ncols(y) covariance matrix.

                              Keyword 'all'
Alternatively, you can use keyword phrase 'all:T' to compute all
quantities except the covariance matrix for each cell.  You can suppress
specific computations by, for example, 'sum:F' and 'prod:F', or force
computation of covariance matrices by 'covar:T'.

Example:
  tabs(y,a,b,all:T,stddev:F,min:F,max:F,sum:F,prod:F) is equivalent
  to tabs(y,a,b)

                               Keyword 'n'
You can use 'n:T' and 'n:F' instead of 'count:T' and 'count:F'.
However, with 'n:T', the corresponding component of the result will
still be named 'count'.

When y is NULL or absent, only 'count:T' or 'n:T' are permitted and at
least one factor is required.
```

2.352 tan()

Usage:

```
   tan(x [, degrees:T or radians:T or cycles:T]), x REAL or a structure
     with REAL components x in radians (default), cycles, or degrees as set
     by option "angles" or the optional keyword
```

Keywords: transformations

```
                                 Usage
tan(x) computes the tangents of the elements of x, where x is a REAL
scalar, vector, matrix or array.  The result has the same shape as x.
```

```
The argument is considered to be in units of radians, degrees or cycles
as determined by the value of option 'angles'.  The default is radians.
See subtopic 'options:"angles"'.

tan(x, radians:T), tan(x, degrees:T), tan(x, cycles:T) interpret x as in
the indicated units, regardless of the value of option 'angles'.

When any element of x is MISSING or is too large (> 5000000*PI radians
in absolute value), the corresponding element of tan(x) is MISSING and a
warning message is printed.

When x is a structure, all of whose non-structure components are REAL,
tan(x [,UNITS:T]), where UNITS is one of 'radians', 'degrees' or
'cycles', is a structure of the same shape and with the same component
names as x with each non-structure component transformed by tan().

                            Cross references
See topic 'transformations' for more information on tan(), including its
use with a CHARACTER argument.
```

2.353 tanh()

Usage:

```
tanh(x), x REAL or a structure with REAL components
```

Keywords: transformations

```
                                  Usage
tanh(x) returns the hyperbolic sine of the elements of x, when x is a
REAL scalar, vector, matrix or array.  The result has the same shape as
x.  In terms of other functions, tanh(x) = (exp(x)-exp(-x))/
((exp(x)+exp(-x))

When any element of x is MISSING the corresponding element of tanh(x) is
MISSING and a warning message is printed.

When x is a structure, all of whose non-structure components are REAL,
tanh(x) is a structure of the same shape and with the same component
names as x, with each non-structure component transformed by tanh().

                            Cross references
See topic 'transformations' for more information on tanh().
```

2.354 tek()

Usage:

```
tek()
```

Keywords: plotting

```
                                  Usage
tek() (no argument) puts your terminal in Tektronix 4014 emulation mode
if you are running MacAnova on Unix/Linux through a terminal emulator
with this capability.  The codes emitted are taken from
getoptions(tekset:T)[1].  See subtopic 'options:"tekset"'.

You normally don't need tek() since MacAnova automatically switches into
Tektronix mode when a graph is drawn.

tek() is implemented as a pre-defined macro (Unix/Linux versions only).

                              Cross references
See also topics vt(), vtx(), plot(), chplot(), lineplot(), showplot().
```

2.355 tekx()

Usage:

```
texk()
```

Keywords: plotting

```
                                  Usage
tekx() puts a Unix/Linux work station Xterm terminal emulator in
Tektronix 4014 mode from vt100 mode.  You don't normally need tekx since
MacAnova recognizes when it is running in a Xterm environment (the value
of environmental variable $HOME is "xterm") and automatically switches
to Tektronix 4014 mode to draw a high resolution graph.

tekx is implemented as a pre-defined macro (Unix/Linux versions only)

                              Cross references
See also topics tek(), vt(), vtx(), 'graphs', 'unix'.
```

2.356 time_series

Keywords: time series

```
There are many MacAnova functions that are useful in time series
analysis.  In addition there are two files, tser.mac and arima.mac,
containing macros for doing frequency domain and time domain analyses.

                Functions useful in time series analysis
Fast Fourier transforms (FFT) for Complex, Hermitian, and Real series
  cft(), hft(), rft()
Functions for working with complex and Hermitian series and Fourier
transforms
```

```
  cconj(), creal(), cimag(), cpolar(), crect(), cprdc(), cprdcj(),
  cdivc(), cdivcj(), hconj(), hreal(), himag(), hpolar(), hrect(),
  hprdh(), hprdhj(), hdivh(), hdivhj() cmplx(), ctoh(), htoc(),
  unwind(), reverse(), padto(), rotate()
Autoregressive and moving average operators and their zeros
  autoreg(), movavg(), polyroot()
Other functions
  convolve(), partacf(), yulewalker()

Type, for example, 'usage(cft)' or 'help(cft)', to get a thumbnail
sketch or a complete description of cft().

                    Macros for time series analysis
File tser.mac, distributed with MacAnova, contains the following macros:
  arspectrum()  Estimate spectrum of autoregression by solving the Yule-
                Walker equations
  autocor()     Compute autocorrelation function
  autocov()     Compute autocovariance function
  burg()        Estimate autoregression coefficients using Burg's
                algorithm and optionally compute the spectrum of the
                fitted moddel
  compfa()      Compute smoothed modified periodograms and, optionally,
                cross periodograms, using cosine tapering, with optional
                detrending
  compza()      Compute the Fourier transform of a cosine tapered
                series, optionally detrending
  costaper()    Compute a cosine taper with a specified amount of
                tapering
  crosscor()    Compute auto and cross correlation function
  crosscov()    Compute auto and cross covariance or correlation
                function
  crsspectrum() Compute smoothed periodgrams and cross periodogram with
                no tapering or detrending
  detrend()     Remove a polynomial trend in equally spaced time
  dpss()        Compute discrete prolate spheroidal sequences
  evalpoly()    Evaluate a real polynomial of a complex variable
  ffplot()      Plot a frequency function against frequency
  gettsmacros() Retrieves macros from tser.mac
  multitaper()  Compute multitaper spectrum estimates
  spectrum()    Compute smoothed periodograms with no tapering or
                detrending
  testnfreq()   Test whether its argument has prime factors > 29
  tsplot()      Plot time series against time

These can be retrieved by, for example, getmacros(multitaper) or
multitaper <- macroread("tser.mac","multitaper").

In addition, there are several macros for working with complex matrices
in fully complex form.  See subtopic 'matrices:"complex_matrices"' for a
list.

                       Help for macros in tser.mac
Help for these macros is available in file tser.hlp.  You can get help
```

```
on these macros using help().  If you know a macro is in tser.mac, it
may be faster to use pre-defined macro tserhelp().  For example, to get
help on burg(), type tserhelp(burg).  See topic tserhelp() for details.
tserhelp() also can retrieve the following informational topics from
tser.hlp:

  bandwidth     Comments about the bandwidth and EDF of spectrum
                estimates
  complex_data  Information on representing complex data and series in
                MacAnova
  complex_fun   Summary of MacAnova functions for working with complex
                series
  fourier       Information concerning Fourier transforms
  hermitian     Information on complex series with Hermitian symmetry

See help() for information on direct use of help() to retrieve help
information from tser.hlp and arima.mac.

                        Macros in file arima.mac
File arima.mac, distributed with MacAnova, contains the following
macros:
  acfarma()      Compute autocovariance function of ARMA model
  arima()        Fit ARIMA model or linear regression with ARIMA errors
                 by unconditional least squares or MLE estimation
  arimares()     Compute residuals from ARIMA model; used by macros
                 arima, hannriss, innovest.
  hannriss()     Fit an ARIMA model using the Hannan-Rissannen algorithm
  innovations()  Compute the coefficients for one step prediction in
                 terms of previous one-step prediction errors.  Used by
                 macro innovest
  innovest()     Fit an ARIMA model using the innovations algorithm
  levmar()       Fit a non-linear model by least squares using a form of
                 the Levenberg-Marquart algorithm
  moveoutroots() Fix up coefficients for a MA or AR operator so that all
                 the zeros are outside the unit circle in the complex
                 plane
  neg2logLarma() Compute -2log(L) from for ARIMA model with given
                 coefficients
  nlreg()        Fit a non-linear regression model by least squares.
  rhatcovar()    Compute variances and or covariances of sample
                 autocorrelations or the entire variance matrix of the
                 sample autocorrelation function using Bartlett's
                 formula
  rhatvar()      Compute variances of sample auto correlations using
                 Bartlett's formula
  specarma()     Compute spectrum of ARMA model

                      Help for macros in arima.mac
File arima.mac serves as its own help file from which the help topics
can be retrived either by help() or by pre-defined macro arimahelp().
If you know a topic is in arima.mac, arimahelp() may be faster since it
searches only one file.  To get help on, say, macro arima(), type
arimahelp(arima) or help(arima).  See arimahelp() for details.
```

```
See help() for information on direct use of help() to retrieve help
information from tser.hlp and arima.mac.

                            Cross references
See also topics 'macros', macroread(), getmacros(), usage(),
macrousage().
```

2.357 tint()

Usage:

```
   tint(x,Coverage), x a REAL vector or matrix, 0 < Coverage < 1 a REAL
     scalar
```

Keywords: probabilities, descriptive statistics, confidence intervals

```
                            Usage and example
tint(x,Coverage) computes a (two sided) t confidence interval with
coverage rate Coverage for the population mean of the data in REAL
vector x.  The result is vector(lowerLimit, upperLimit).

Coverage must be a REAL scalar between 0 and 1.

When x is a matrix with M columns, tint(x, Coverage) returns a 2 by M
matrix, with lower and upper limits in rows 1 and 2, respectively.

When there are missing values, an informative message is printed.

Example:
  Cmd> alpha <- .05; tint(x,1-alpha) # computes 95% conf. interval.

                            Cross references
See also tval(), t2val(), and t2int().
```

2.358 toclip()

Usage:

```
   toclip(x [, missing:Code, format:Fmt, sep:C1, linesep:C2]), Code,
     Fmt, C1, C2 quoted strings or CHARACTER scalars
```

Keywords: character variables, output

```
                                  Usage
toclip(x) is equivalent to CLIPBOARD <- x and puts a possibly multi-
lined CHARACTER representation of x in special variable CLIPBOARD.

When x is REAL, you can use keywords 'sep', 'missing', 'linesep', and
'format' that are permissible with paste(x,multiline:T,...).
```

```
In windowed versions, toclip() allows easy export of MacAnova data to
another application such as a spreadsheet.  For example, if x is a
10 by 5 REAL matrix that you want to export to a spreadsheet, after
'toclip(x)', you can select a 10 by 5 rectangle of cells in the
spreadsheet and select Paste in the Edit menu.  When the target
application has special requirements for representing MISSING or field
and line separators, you can use keywords to customize what gets put
on the Clipboard.

                                Examples
Examples:
  toclip(x,missing:"NA") and toclip(x,missing:"-99") put x in CLIPBOARD
    with MISSING coded as NA and -99, respectively
  toclip(x,sep:",",format:".10f") puts x in CLIPBOARD with elements
    separated by commas and with 10 decimal places
  toclip(x,linesep:":",sep:",") puts x in CLIPBOARD with elements
    separated by commas and rows separated by colons.

                             Cross references
See also topics 'CLIPBOARD', fromclip(), paste(), clipreaddata().

toclip() is implemented as a pre-defined macro.
```

2.359 toeplitz()

Usage:

```
toeplitz(x), x a REAL vector.
```

Keywords: matrix algebra

```
                                   Usage
toeplitz(x) computes an n by n Toeplitz matrix from a REAL vector x of
length n.  After a <- toeplitz(x), a is constant on the diagonals with
a[i,j] == x[|i-j| + 1].

The primary use of toeplitz() is to create a covariance matrix from an
auto-correlation function.  For example, if gamma0, gamma1, ... are the
variance and first n-1 autocovariances of a stationary time series
{x[t]}, then toeplitz(vector(gamma0, gamma1, ...))  computes the n by n
covariance matrix of the vector vector(x[1], x[2], ..., x[n]), or indeed
of vector(x[1+k], x[2+k], ..., x[n+k]) for any integer k.

                             Cross references
See also partacf(), yulewalker().
```

2.360 trace()

Usage:

```
trace(x), x a REAL square matrix
```

Keywords: matrix algebra

```
                                Usage
trace(x) computes the so-called trace of REAL square matrix x, that is,
sum(diag(x)) = x[1,1]+x[2,2]+...+x[k,k], when x is k by k..

                            Cross references
See also topics 'matrices', dmat(), diag(), det().
```

2.361 transformations

Usage:

```
List of available transformations.  Argument x is REAL or structure with
REAL components. 't' => behavior depends on option 'angles'. '2' => has
two argument variant.
 Transformation          Result for REAL scalar x
   abs(x)                |x| = absolute value of x
   acos(x)        t      arccosine of x
   asin(x)        t      arcsine of x
   atan(x)        t 2    arctangent of x
   atanh(x)              inverse hyperbolic tangent of x
   ceiling(x)            least integer >= x
   cos(x)         t      cosine of x
   cosh(x)               hyperbolic cosine of x
   digamma(x)            digamma(x) = (d/dx)log(gamma(x))
   exp(x)                exp(x) = exponential function of x
   floor(x)              greatest integer <= x
   lgamma(x)             ln(gamma(x)) = log gamma function of x
   log(x)                ln(x) = base-e log(x) = natural logarithm of x
   log10(x)              base-10 log(x) = common log of x
   log2(x)               base-2 log(x) = log(x)/log(2)
   polygamma(x)   2      digamma(x) and (d/dx)^n digamma(x)
   round(x)       2      nearest integer to x
   sin(x)         t      sine of x
   sinh(x)               hyperbolic sine of x
   sqrt(x)               square root of x
   tan(x)         t      tangent of x
   tanh(x)               hyperbolic tangent of x
```

Keywords: transformations

```
                        Available transformations
Transformation are abs(x), acos(x), asin(x), atan(x) or atan(x,y),
atanh(x), ceiling(x), cos(x), cosh(x), digamma(x), exp(x), floor(x),
lgamma(x), log(x), log10(x), log2(x), polygamma(x) or polygamma(x,n),
round(x) or round(n,ndec), sin(x), sinh(x), sqrt(x), tan(x) and tanh(x)
```

where x (and y) are REAL variables or structures with REAL components. x can also be a CHARACTER variable (see below).

Type usage(transformations) for one line descriptions of what these transformations compute. Type help(tranName), for example help(cos) or help(sqrt), for help on individual transformations.

Trigonometric functions and their inverses are affected by the value of option 'angles'. See below and subtopic 'options:"angles"'.

Vector, matrix, array, or structure argument

When x is a REAL vector, matrix or array, the result is REAL with the same size and shape, with the elements of the result the transformations of the elements of x. For example, if x is a matrix, exp(x)[i,j] is exp(x[i,j]).

When x is a structure, the result is a structure of the same shape and with the same component names as x, whose components are the transformed components of the argument. See topic 'structures'.

MISSING or out of range argument

When any element of x is MISSING or outside the range of validity for the function (for instance, < 0 for sqrt()), the corresponding element of the result is MISSING and a warning message is printed. See below for more details.

Propagation of labels

When x or any components have labels, the result has the same labels. See topic 'labels'.

Trignometric Functions

For sin(), cos() and tan(), the elements of x are interpreted as being in radians, cycles or degrees according to the value of option 'angles' whose default value is "radians". Similarly, the values returned by asin(), acos() and atan() are in the units specified by option 'angles'. You can change the default by setoptions(angles:"degrees") or setoptions(angles:"cycles"). See topic setoptions(), subtopic 'options:"angles"'.

All the trigonometric functions also allow one of 'degrees:T', 'cycles:T' and 'radians:T' as an extra argument. These override the units specified by option 'angles'. For example, sin(30, degrees:T) always returns 0.5 and asin(0.5, degrees:T) always returns 30.

Illegal or too large arguments

When the argument to a function is illegal (for example, sqrt(-1) or atanh(1.2)), a warning message is printed and the result is set to MISSING.

When the value of a function is too large to be represented in the computer (for example, sinh(-3000)), a message is printed and the result is set to MISSING.

```
Because of significant loss of precision in computing trigonmetric
functions of a large argument, the result of sin(x), cos(x) or tan(x) is
MISSING when |x| >= 5000000*PI radians (= 2500000 cycles = 900000000
degrees) and a warning message is printed.

When x > 4503599627370495 or x < -4503599627370495, floor(x) and
ceiling(x) are set to MISSING because of the impossibility of exact
representation of integers beyond these limits.  These limits may be
different on some computers.

                            CHARACTER argument
When the argument x to a transformation is a CHARACTER variable, the
result is a CHARACTER variable of the same size and shape with elements
usually involving the transformation name and the elements of x.

  Cmd> log10(vector("height","weight"))
  (1) "log10(height)"
  (2) "log10(weight)"

Any element of x that is "" or starts with '@', '(', '[', '{', '<', '/'
or '\' is not modified.

  Cmd> log(vector( "", "@[", "(1)"))
  (1) ""
  (2) "@["
  (3) "(1)"

This also works with two argument transformations such as atan(x,y) and
round(x,p), as long as both arguments are CHARACTER (p can be a number
on round()).

  Cmd> round("x","4") # round("x",4) works the same way
  (1) "round(x,4)"

This feature can be useful in creating labels for transformed data.

  Cmd> logx <- matrix(log(x),labels:structure(getlabels(x,1),\
        log(getlabels(x,2))))

This uses the row labels of x and transforms the column labels of x.

                             Cross references
See also topics 'arithmetic', 'syntax', atan(), hypot(), boxcox(),
polygamma(), round(), 'structures', rational(), labels, getlabels(),
matrix().
```

2.362 transpose()

Usage:

```
   x'  or  t(x), where x is a matrix
```

Keywords: matrix algebra, operations

```
                                Usage
x' (x followed by a single quote or "prime") computes the transpose of x
if x is a matrix, that is the matrix y with y[i,j] = x[j,i].

When x is a vector of length n, x' is a 1 by n matrix, that is, a row
vector.

When x is an array with dimensions n1, n2, ..., nk, y <- x' computes an
array y with dimensions nk, ..., n1 such that y[i1,...,ik] is
x[ik,...,i1].  When x is a generalized matrix (see 'matrices'), so is
x', and matrix(x)' = matrix(x').

t(x) is synonymous with x'.

Provided ndims(x) > 1, t(x,run(ndims(x),1)) is equivalent to t(x) and
x'.  See also t().

              Transposition with matrix multiplication
Instead of x' %*% y and x %*% y' you can use x %c% y and x %C% y,
respectively which use less internal memory.  See topic 'matrices' for
more information on these matrix multiplication operators.

                          Structure argument
When x is a structure, each of whose components is REAL, LOGICAL, or
CHARACTER, x' computes a structure with the same shape and with the same
component names as x whose non-structure components are the transposes
of the corresponding components of x.

                           Cross references
See also topics array(), 'matrices', 'subscripts'.
```

2.363 trideigen()

Usage:

```
trideigen(Diag, Subdiag [ [, start] , end], values:F or vectors:F),
   Diag, Subdiag REAL vectors, start and end positive integers
```

Keywords: matrix algebra, time series

```
                                Usage
trideigen(Diag, Subdiag) computes the eigenvalues and eigenvectors of
the symmetric tri-diagonal matrix with diagonal Diag and sub- and
super-diagonal Subdiag. Diag and Subdiag must be REAL vectors with
length(Subdiag) = n - 1 or length(Subdiag) = n, where n = length(Diag).
In the latter case, Subdiag[1] is ignored.  The result is a structure
with components 'values' and 'vectors', similar to eigen.  The
eigenvalues are returned in decreasing order.
```

```
trideigen(Diag, Subdiag, vectors:F) computes only the eigenvalues,
returning a vector of length n.

trideigen(Diag, Subdiag, values:F) computes only the eigenvectors,
returning a n by n matrix.

                 Limiting number of eigenvalues computed
trideigen(Diag, Subdiag, start, end) computes the i-th eigenvalues and
eigenvectors, for i = start, ..., end.  trideigen(Diag, Subdiag) is
equivalent to trideigen(Diag, Subdiag, 1, length(Diag)).

trideigen(Diag, Subdiag, end) is eqivalent to trideigen(Diag, Subdiag,
1, end).

                               Application
Command trideigen() was added specifically to make it straightforward to
compute discrete prolate spheroidal sequences (dpss) used in multi-taper
spectrum analysis.  For these, if W is the desired width, the following
computes the first K dpss
   d <- cos(2*PI*W)*(.5*run(-n+1,n-1,2))^2
   e <- .5*run(0,n-1)*run(n,1)
   dpssvecs <- trideigen(d,e,K,values:F)

                         Comparison with eigen()
The advantages of trideigen() over eigen() are (a) you do not need to
create the n by n tridiagonal matrix, and (b) you can obtain a subset of
the eigenvalues and eigenvectors.

                            Cross references
See also eigen(), releigen().
```

2.364 trilower()

Usage:

```
trilower(A), A a matrix
```

Keywords: matrix algebra, variables, combining variables

```
                                  Usage
trilower(a) returns a matrix d of the same size and shape as a with
d[i,j] = a[i,j] for i >= j (on or below the diagonal) and d[i,j] = 0 for
i < j (above the diagonal).  Variable a must be a matrix but need not be
square.
                        [1   5   9]                    [1   0   0]
For example, when a is  [2   6  10] , trilower(a) is   [2   6   0] .
                        [3   7  11]                    [3   7  11]
                        [4   8  12]                    [4   8  12]

When a has type CHARACTER, elements above the diagonal are sent to empty
strings ("") instead of 0's.
```

```
trilower(a,T) or trilower(a,pack:T) returns the lower triangle (elements
a[i,j] with i >= j) of a in packed form.  Matrix a must be square, that
is, nrows(a) = ncols(b).  For example, when
               [1   4   7]
          a =  [2   5   8]  ,
               [3   6   9]
trilower(a) is vector(1, 2, 5, 3, 6, 9).

Note: keyword 'square' is not valid for trilower().

                            Cross references
See also triupper(), triunpack(), qr().
```

2.365 triunpack()

Usage:

```
   triunpack(vec [, lower:T  or  upper:T]), vec a vector of length p(p+1)/2
```

Keywords: matrix algebra, variables, combining variables

```
                                  Usage
triunpack(v) creates a symmetric square matrix from vector v which
specifies the upper triangular part of the matrix, including the
diagonal.  The length of v be of the form p*(p+1)/2, that is, it is a
'triangular number' and the dimension of the result is p by p.  For
example, triunpack(run(10)) produces the matrix
             [1   2   4   7]
             [2   3   5   8]
             [4   5   6   9]
             [7   8   9  10]

in which the upper triangle is filled column by column from run(10) and
the lower triangle is filled in symmetrically.

triunpack(v,upper:T) does the same, except the elements below the
diagonal are set to 0.

triunpack(v,lower:T) returns triunpack(v,upper:T)', whose elements above
the diagonal are 0.

When v is a CHARACTER vector, empty strings ("") are used instead of 0's
in filling out the other half.

                                 Example
Example:
  Cmd> triunpack(run(10), upper:T)
and
  Cmd> triunpack(run(10), lower:T)
produce
```

```
            [1   2   4   7]          [1   0   0   0]
            [0   3   5   8]          [2   3   0   0]
            [0   0   6   9]   and    [4   5   6   0] , respectively.
            [0   0   0  10]          [7   8   9  10]

                            Cross references
See also triupper(), trilower(), qr().
```

2.366 triupper()

Usage:

```
triupper(A [,pack:T]), A a matrix
```

Keywords: matrix algebra, variables, combining variables

```
                                    Usage
triupper(a) returns a matrix d of the same size and shape as a with
d[i,j] = a[i,j] for i <= j (on or above the diagonal) and d[i,j] = 0 for
i > j (below diagonal).  Variable a must be a matrix but need not be
square.
                         [1   5   9]                   [1   5   9]
For example, when a is   [2   6  10] , triupper(a) is  [0   6  10]
                         [3   7  11]                   [0   0  11]
                         [4   8  12]                   [0   0   0]

triupper(a,square:T) returns the m by m upper left block of triupper(a),
where m = min(nrows(a),ncols(a)).

When a has type CHARACTER, elements below the diagonal are sent to empty
strings ("") instead of 0's.

                                Packed output
triupper(a,T) or triupper(a,pack:T) returns the upper triangle (elements
a[i,j] with i <= j) of a in packed form.  Matrix a must be square, that
is, nrows(a) = ncols(a).  For example, when
                [1   4   7]
           a =  [2   5   8]
                [3   6   9]

triupper(a,pack:T) is vector(1, 4, 5, 7, 8, 9).

                                Cross references
See also trilower(), triunpack(), qr().
```

2.367 tserhelp()

Usage:

```
tserhelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
tserhelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
  Subtopics
tserhelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
tserhelp(key:Key), CHARACTER scalar Key
tserhelp(index:T [,scrollback:T])
```

Keywords: general, time series

```
                                  Usage
tserhelp(Topic1 [, Topic2, ...]) prints help on topics Topic1, Topic2,
... related to macros in file tser.mac.  The help is taken from file
tser.hlp.

tserhelp(Topic1 [, Topic2, ...], usage:T) prints usage information
related to these macros.

tserhelp(index:T) or simply tserhelp() prints an index of the topics
available using tserhelp().  Alternatively, help(index:"tser") does
the same.

tserhelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER scalar
or vector, prints subtopics of topic Topic.  With subtopic:"?", a list
of subtopics is printed.

tserhelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where Suptopics1
and Subtopics2 are CHARACTER scalars or vectors, prints the specified
subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to tserhelp().  In windowed
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

                              Keyword 'key'
tserhelp(key:key) where key is a quoted string or CHARACTER scalar lists
all topics cross referenced under Key.  tserhelp(key:"?") prints a list
of available cross reference keys for topics in the file.

tserhelp() is implemented as a predefined macro.

                            Cross references
See help() for information on direct use of help() to retrieve
information from tser.hlp.
```

2.368 tinterval()

Usage:

```
tinterval(x[,y],cover:fraction,[upper:T or lower:T, pool:T|F])
```

Keywords: probabilities, descriptive statistics, comparisons

```
tinterval() computes a t-confidence interval for a population mean
or difference of population means, depending on whether one or two
variables are given as arguments.  By default, tinterval() computes
a two-sided interval, but you may choose one-sided alternatives
by using one of lowerb:T or upperb:T.

You specify the coverage rate via cover:value. You may choose pooled
or unpooled variances in two-sample tests via pool:T or pool:F.

The output is a vector containing the estimate and interval bounds.

Because the variance of the data must be known, these intervals are
rarely used in practical data analysis.
```

2.369 ttest()

Usage:

```
ttest(x[,y],(upper:T or lower:T), null:val [pooled:T|F])
```

Keywords: probabilities, descriptive statistics, comparisons

```
ttest() performs a one- or two-sample t-test, depending on whether
one or two variables are given as arguments.  By default there is
a two-tailed alternative, but you may choose one-sided alternatives
by using one of twotail:T, lowertail:T, or uppertail:T.
You specify the null value via null:value.  For two-sample tests,
you may choose to pool variances or not pool via pooled:T or pooled:F.

The output is a vector containing the statistic, the degrees of
freedom, and the p-value.
```

2.370 tval()

Usage:

```
tval(x [,df:T]), x a REAL vector or matrix
```

Keywords: probabilities, descriptive statistics, comparisons

```
                                  Usage
tval(x) computes the Student's t-statistic for testing the null
```

```
hypothesis that the data in REAL vector x have mean zero.  When x is a
matrix, the results is a vector with the t-statistics for each column of
x.

tval(x,df:T) produces a structure with two components, 't' containing
the t-statistic and 'df' containing the degrees of freedom.  When x is a
matrix, each component is a vector of length ncols(x).

When x contains MISSING values, an informative message is printed.

                          Testing H_0:mu = delta
When the null hypothesis mean is something other than zero, say delta,
tval(x-delta) will produce the correct t value for that null hypothesis.
For example, to test H0: E[x] = 3, use tval(x-3).

                                 P values
P values may be computed using cumstu() or twotailt(), for example, by
  Cmd> @result <- tval(x,df:T);twotailt(@result$t,@result$df)

                             Cross references
See also topics t2val(), tint(), t2int(), cumstu(), and twotailt().
```

2.371 twotailt()

Usage:

```
   twotailt(tval, df), tval a REAL scalar, df > 0 REAL.
```

Keywords: probabilities, descriptive statistics, comparisons

```
                                    Usage
twotailt(tval,df) computes the two tailed P value for a Student's t
value of tval with df degrees of freedom.  It is implemented as a macro.

In the simplest usage, tval and df are REAL scalars with df > 0.
However, if one argument is a scalar, the other argument can be a REAL
vector, matrix or array and the result has the same size and shape.
When both arguments are not scalars, they both must have the same size
and shape.

                             Cross references
See also cumstu().
```

2.372 typeof()

Usage:

```
   typeNames <- typeof(arg1 [, arg2 ...]), arg1, arg2 ... arbitrary
     variables, including macros and built-in functions.
```

Keywords: variables

```
                                Usage
typename <- typeof(arg), where arg is any variable including macro and
built-in function sets typename to the type of arg, "CHARACTER",
"LOGIC", REAL", "STRUCTURE", "NULL", "MACRO", "FUNCTION", "UNDEFINED" or
"GRAPH".

typenames <- typeof(arg1, arg2 ...), where arg1, arg2, ... are any
variables, makes typenames a CHARACTER vector with length(typenames) =
number of arguments, with typenames[i] the name of the type of argument
i.

                               Examples
Examples:

  Cmd> a <- sqrt(2); typeof(a)
  (1) "REAL"

  Cmd> b <- "pi"; typeof(b)
  (1) "CHARACTER"

  Cmd> typeof(a) == typeof(b) # test equality of types
  (1) F

  Cmd> typeof(PI,T,"type",structure(PI,T),LASTPLOT,notavar,typeof,help)
  (1) "REAL"
  (2) "LOGICAL"
  (3) "CHARACTER"
  (4) "STRUCTURE"
  (5) "GRAPH"
  (6) "UNDEFINED"
  (7) "FUNCTION"
  (8) "MACRO"

                           Cross references
See also topics nameof(), shapeof(), isarray(), ischar(), isdefined(),
isfactor(), isfunction(), isgraph(), islogic(), ismacro(), ismatrix(),
isname(), isnull(), isnumber(), isreal(), isscalar(), isvector().
```

2.373 unique()

Usage:

```
   unique(x [,index:T, fuzz:d, relative:T]), x REAL, LOGICAL or CHARACTER,
     d >= 0 a REAL scalar
```

Keywords: ordering, variables

```
                                Usage
unique(x) computes a vector consisting of all the distinct non-MISSING
values in the REAL, LOGICAL or CHARACTER vector, matrix, or array x.
```

```
When x is REAL or LOGICAL, it is an error when all its elements are
MISSING.  If x1 contains the same values of x in a different order,
unique(x1) will return the same values as unique(x), but possibly in a
different order.

                              Keyword 'index'
unique(x, index:T) computes a vector J of positive integers such that
x[J] is the same as unique(x).  That is it finds the subscripts of the
unique non-missing elements of x.

                             Inexact matching
unique(x, fuzz:d [, index:T]), where x is REAL and d >= 0 is a REAL
scalar does the same, except that, as x is scanned, x[j] is determined
to be different from x[i], 1 <= i < j only if abs(x[j] - x[i]) > d.  The
numbers returned may depend on the ordering of values in x.  That is,
for example, unique(x, fuzz:d) and unique(sort(x),fuzz:d) may return
different sets of numbers.

unique(x, fuzz:d, relative:T [, index:T]) does the same, except x[j] is
determined to be different from x[i] only if abs(x[j] - x[i]) > D, where
D = d*(abs(x[j]) + abs(x[i])).

                                 Examples
Examples:
  Cmd> unique(vector(5,3,1,2,4,2,5,7,2,7))
  (1)          5          3          1          2          4
  (6)          7

  Cmd> unique(vector(5.1,3,2.9,3.5,5,2.6), fuzz:.15)
  (1)        5.1          3        3.5        2.6

  Cmd> unique(sort(vector(5.1,3,2.9,3.5,5,2.6)), fuzz:.15)#order differs
  (1)        2.6        2.9        3.5          5

  Cmd> x <- vector(run(3), run(3)+1e-4)

  Cmd> unique(x,fuzz:4e-5)
  (1)          1          2          3     1.0001     2.0001
  (6)     3.0001

  Cmd> unique(x,fuzz:4e-5,relative:T)
  (1)          1          2          3     1.0001

  Cmd> unique(vector(T,T,T,F,T))
  (1) T       F

  Cmd> paste(unique(vector("B","C","A","B","D","A","A")))
  (1) "B C A D"

  Cmd> unique(vector("B","C","A","B","D","A","A"), index:T)
  (1)           1           2           3           5

If x is a REAL or CHARACTER vector,
```

```
  Cmd> a <- factor(match(x,unique(x)))
```

computes a factor each level of which corresponds to a unique value of vector x and

```
  Cmd> a <- factor(match(x,sort(unique(x))))
```

does the same except the factor levels are in the same numerical or alphabetic order as the elements of x.

Cross references

See also factor(), match()..

2.374 unix

Keywords: general

Unix versions

Two Unix/Linux versions are distributed. One runs in the command or shell window where it was launched (usually an xterm or equivalent). The other (the Carapace version) uses multiple command windows, high resolution graphics windows, the mouse, menus, etc.

Features common to all Unix/Linux versions

Various command line arguments are recognized. These allow automatic restoring of a workspace, suppressing the startup message, etc. See topic 'launching'.

MacAnova recognizes options, file names and path names specified in environmental variable MACANOVA. See topic 'customize'.

The startup file is Macanova.ini.txt in ~/MyMacAnovaFiles unless flag -f has been used on the command line (see 'launching') or in environmental variable MACANOVA. See topic 'customize'.

The size of variables is limited only by the amount of memory and disk space available.

Unless you use command line option -home (see 'launching') or include -home in environmental variable MACANOVA (see 'customize'), MacAnova pre-defines CHARACTER variable HOME to contain the user's home directory. HOME is used to expand file names of the form "~/path" by substituting the value of HOME for '~'. This allows you to refer to files in your home directory such as stuff.data as "~/stuff.data" regardless of the current directory. When you redefine HOME, it changes the expansion of "~/". See topic 'files'.

File names starting with "~name/" are expanded similarly to the shell. That is, name is taken to be a Unix/Linux user name and "~name" is expanded to the path name of the home directory of that user. Redefining

HOME has no effect on this expansion. See topic 'files'.

Pre-defined variables DATAPATHS, and DATAPATH are initialized with path names that are installation dependent (see topic 'DATAPATHS'). You can override these using options -path or -appdir on the command line (see 'launching') or in environmental variable MACANOVA (see 'customize').

Non-Windowed Version

High resolution graphics are implemented by emitting codes appropriate for plotting on a Tektronix 4014 graphics terminal. When this is not appropriate, you are limited to displaying "dumb" plots and should use setoptions(dumbplot:T). See topic setoptions(), subtopic 'options:"dumbplot"'.

When MacAnova is running in an xterm, the graphics commands open a pseudo Tektronix 4014 graphics window and draw in it, switching back to the text window when <RETURN> is hit after the plot. Be aware that certain vendor replacements for xterm (for example, hpterm and dterm) don't support Tektronix emulation. To use high resolution graphics you need to start up an xterm window.

You can execute Unix/Linux commands by prefixing the line with '!' in the first position after the prompt or by using command shell(). Keyword phrase keep:T on shell() is recognized. You must use keyword phrase interact:T if the program being invoked expects any input from you. A line of the form '!command ... ' is equivalent to shell("command ...", interact:F). See shell().

A pre-defined macro edit() is available which allows you to edit macros and data from within MacAnova.

Key bindings

Depending on how it was compiled on a particular system, keyboard line editing and history may be available as implemented using the GNU Readline Library. If so, the default editing mode is based on Emacs. You can customize key bindings by creating a special file named ".inputrc" in your MyMacAnovaFiles directory. This will be read each time MacAnova starts up. To enable editing based on Vi commands instead of Emacs, put the following lines in this file:

```
   set editing-mode vi
   "k": previous-history
   "j": next-history
   "H": beginning-of-history
   "G": end-of-history
```

On some systems a standard environmental variable INPUTRC is defined and keyboard bindings are taken from file $INPUTRC and not from .inputrc. If that is the case and you want to customize key bindings, you will need to define INPUTRC in your .profile or .cshrc file. For csh, tcsh or derivatives use

```
  setenv INPUTRC $HOME/.inputrc
```

For sh, ksh, bash and other similar shells, use

```
  export INPUTRC=$HOME/.inputrc
```

History of previsous commands

When keyboard line editing is implemented, a "history" mechanism is also available. A certain number (default is 100) of previous commands are saved. Using the emacs editing mode, you can scroll backward by pressing Ctrl+P (possibly also an up-arrow key) and scroll forward by pressing Ctrl+N (down-arrow). Under the vi editing mode, the corresponding keys (when in vi command mode) are 'k' amd 'j'. See setoptions() for information on how to change the number of lines saved.

As part of this facility you also may have so-called file name completion. When you have partially typed a file name, press the Tab key or press the Esc key twice and an attempt will be made to complete it. Press Tab twice or Esc four times and you will get a list of possible completions. See Readline documentation for information on modifying key bindings using file .inputrc. If you want to use the Tab key as a regular key, but the following line in the .inputrc file:
"C-i": self-insert

Carapace Version

This version is compiled using the Carapace library which in turn uses the WxWidgets library. It allows multiple command/output and high resolution graphics windows and uses menus, dialogs, the mouse, and so forth in the usual way. Commands are typed into the lower pane of a command window, and output appears in the upper pane. Output and graphics windows can be printed and/or saved to files.

Text in a command window can be copied to the clipboard. Content of a graphics window can be copied to the clipboard as a bitmap. The MacAnova variable CLIPBOARD is connected to text on the clipboard in the sense that accessing CLIPBOARD returns a MacAnova string containing the text content of the clipboard, and assigning to CLIPBOARD writes text to the clipboard.

Cross references

See topic 'carapace' for details on the Carapace version.

2.375 unlockvars()

Usage:

```
unlockvars(a [,b,c...] [,silent:T]), a, b, c, arbitrary variables
```

Keywords: general, variables

Usage

unlockvars(var1, var2, ...) marks variables var1, var2, ... as being unlocked variables so that they can be deleted or assigned to. A warning message may be printed if a variable is not locked.

unlockvars(var1, var2, ..., silent:T) does the same except warning

```
messages are not printed.  It is an error if any argument is an
expression or a function result.

                              Cross references
See also lockvars(), delete(), 'variables:"locked_variables"'
```

2.376 unwind()

Usage:

```
   unwind(angleMat [, crit:Val]) where angleMat is REAL matrix and
     .5 <= Val < 1
```

Keywords: time series, complex arithmetic

```
                                    Usage
unwind(angleMat) attempts to eliminate or reduce discontinuities (jumps
between rows) in each column of matrix angleMat, considered as being a
sequence of angles.

The assumed units of the angles are as specified by option 'angles'
("radians", "degrees", or "cycles").  For example, if angles are
measured in degrees, unwind(vector(350, 352, 359, 2, 1, 355)) is
vector(350, 352, 359, 362, 361, 355).  See subtopic 'options:"angles"'.

The operation of unwind() is controlled by the value Val of a criterion.
Let Jump = abs(angleMat[i,j] - angleMat[i-1,j]).  Then when Jump >
Val*Cycle, where Cycle is 2*pi radians, 360 degrees or 1 cycle,
angleMat[i,j] is increased or decreased by a multiple of Cycle so as to
bring the change less than Val*Cycle .  The default is Val = 0.75.

unwind(angleMat,crit:Val) uses Val for the unwinding criterion.  Val
must be between 0.5 and 1 cycles even when option 'angles' has value
"radians" or "degrees".

                              Cross references
See also topics hpolar(), cpolar(), subtopic 'options:"angles"'.
```

2.377 usage()

Usage:

```
   usage(Topic [,allfiles:T]), topic a quoted or unquoted name of a help
     topic
   usage(Topic1, Topic2, ... [,allfiles:F])
```

Keywords: general

```
                                    Usage
usage(Topic) gives short usage information on Topic, where Topic is a
function or command or one of the general information topics such as
```

```
'options' or 'graph_keys'.

usage() works identically to help(), except it gives only a brief
summary of the topic, instead of all the detail.

usage() returns an "invisible" LOGICAL scalar whose value is True only
when at least one topic requested was found.  The value may be assigned
or tested but will not be printed automatically.  See topic
'variables:"invisible"'.

usage() scans the files NAMED in CHARACTER vector HELPFILES.  If the
current help file is not named in HELPFILES, it is scanned first.

If there is only one topic (the usual case), usage() scans files until
it finds the topic.  You can use 'allfiles:T' as an argument to force it
to search all the files in HELPFILES.

usage(topic1, topic2, ...) does the same except that all the files are
searched for the topics.  When you use 'allfiles:F' as an argument, no
additional files are searched once a topic has been found in a file.

For many commands and macros, help(topic, subtopic:"usage") provides
more complete usage information than does usage().

usage() is implemented as a macro which uses gethelp() and getusage() to
retrieve the information.  Functions getusage() and gethelp() scan only
a single file.

                              History note
Prior to Version 4.12, getusage() was named usage() and there was no
macro named usage().

                            Cross references
See help(), getusage() and gethelp() for full details.  See topic
addhelpfile() for information on adding file names to HELPFILES.
```

2.378 userfunhelp()

Usage:

```
userhelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
userhelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
  Subtopics
userhelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
userhelp(key:Key), CHARACTER scalar Key
userhelp(index:T [,scrollback:T])
```

Keywords: general

```
                                  Usage
userfunhelp(Topic1 [, Topic2, ...]) prints help on topics Topic1,
Topic2, ... related to user functions.  The help is taken from file
```

```
userfun.mac.

userfunhelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these topics.

userfunhelp(index:T) or simply userfunhelp() prints an index of the
topics available using userfunhelp().  Alternatively
help(index:"userfun.hlp") does the same thing.

userfunhelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER
scalar or vector, prints subtopics of topic Topic.  With subtopic:"?", a
list of subtopics is printed.

userfunhelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where
Suptopics1 and Subtopics2 are CHARACTER scalars or vectors, prints the
specified subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to userfunhelp().  In windowed
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

                              Keyword 'key'
userfunhelp(key:key) where key is a quoted string or CHARACTER scalar
lists all topics cross referenced under Key.  userfunhelp(key:"?")
prints a list of available cross reference keys for topics in the file.

userfunhelp() is implemented as a predefined macro.

                            Cross references
See help() for information on direct use of help() to retrieve
information from userfun.hlp.
```

2.379 user_fun

Keywords: general, control, files

```
This topic is now in file userfun.hlp.  Type
  userfunhelp(user_fun)

It provides a brief introduction to the form of a user function (routine
compiled separately from MacAnova) that can be loaded by loadUser() and
executed by User().

Some other useful entries in userfun.hlp are callback_fun and
arginfo_fun.  Type
  userfunhelp()
for a complete list of entries.
```

2.380 variables

Keywords: syntax, variables, character variables, logical variables, null variables

Variables and their names

Data are stored in named "variables" with names up to 12 characters long.

Names are case sensitive (for example, 'residuals' is a different name from 'Residuals'). You should avoid names in all capital letters (for example 'RESIDUALS') because MacAnova uses some such names for special purposes.

You should normally select names that are relevant to the problem such as 'weight', 'residuals', or 'depv'.

Variables can be permanent (the variable remains accessible until you delete it) or temporary (the variable is automatically deleted at the next prompt). Temporary variables are particularly useful in macros. See topics macro() and 'macros'.

Permanent variables

Permanent variables have names that begin with a letter (a-z or A-Z) or the character '_', followed by 0 or more letters, numerals or '_'. For example 'x12a' and 'time_of_day' are names of permanent variables. The use of variable names that start with '_' can lead to confusion since the variable is then "invisible" (see below).

Temporary variables

Temporary variables have names that start with '@' followed by a letter or '_' and 0 or more letters, numerals or '_'. Examples are '@x1a' and '@_Result. A variable whose name starts with '@_' is "invisible". All temporary variables are automatically deleted each time a prompt is printed.

Invisible variables

"Invisible" variables have names that begin with '_' or '@_'. They differ from "visible" variables in two ways. (i) Commands list() and listbrief() ignore invisible variables unless 'invis:T' is an argument (see listbrief(), list()); and (ii) typing an invisible variable's name does not result in the variable's value being printed.

Special variables

A few variables are "special" and have non-standard properties. Currently CLIPBOARD, GRAPHWINDOWS and SELECTION (GTK only) are the only "special" variables. See topics 'CLIPBOARD' and 'GRAPHWINDOWS' for details.

Locked variables

Locked variables are variables that cannot be deleted or assigned to. You can mark variables as being 'locked' using lockvars(). You can unlock locked variables using unlockvars() and delete them using keyword phrase 'lockedok:T' on delete().

When you save your workspace (see save()), the locked status of each variable is retained so that when you restore the workspace (see restore()) a locked variable is still locked. However, locking a variable does not protect it from being destroyed by restore().

One situation when locking a variable might be helpful is when you have computed a valuable result that took a long time and you want to make sure you don't destroy it accidentally.

You cannot lock temporary variables or "special" variables.

Types of Variables

There are several types of variables, including REAL, LOGICAL, CHARACTER, GRAPH, STRUCTURE, MACRO and NULL. In certain output, LOGICAL, CHARACTER, and STRUCTURE are abbreviated as LOGIC, CHAR, and STRUC, respectively.

Variables of type LONG can be created by asLong() but they exist only transiently, being "coerced" to equivalent REAL variables when assigned. See asLong().

A REAL variable contains numerical data or the special value MISSING. A LOGICAL variable contains data with values limited to True, False or MISSING (see 'logic'). A CHARACTER variable contains data consisting of character information. A LONG variable contains integer values between -2147483647 and +2147483647 = 2^31 - 1.

REAL, LOGICAL, CHARACTER or LONG variables may be scalars (consist of a single data item; see 'scalars'), vectors (several data items indexed by a single subscript; see 'vectors'), matrices (data items indexed by two subscripts; see 'matrices'), or arrays (data items indexed by more than two subscripts; see 'arrays'). See also topic 'subscripts'.

A GRAPH variable encapsulates all the information needed to draw a graph. See topic 'graphs'.

A STRUCTURE variable or simply a structure consists of named components of data which may be of any type, including STRUCTURE. See topic 'structures'.

A MACRO variable or simply a macro contains one or more MacAnova commands to be executed together. See topics 'macros', 'macro_syntax', macro().

A NULL variable contains no data of any sort. See topic 'NULL'.

You can attach descriptive notes to variables, including GRAPH variables and macros. See topics 'notes', attachnotes(), appendnotes(), getnotes(), and hasnotes().

Coordinate Labels

REAL, LOGICAL and CHARACTER variables may have vectors of labels for

each coordinate. In particular matrices may have row and column labels. A structure may have labels for each component. Labels propagate through operations and functions in a fairly sensible way. Labels are primarily used in output . See topic 'labels' for details.

2.381 varnames()

Usage:

```
varnames(Model [,stripat:T,stripbrack:T]) where Model is CHARACTER
  scalar containing a GLM model
varnames([stripat:T,stripbrack:T]) is equivalent to varnames(STRMODEL
  [,stripat:T,stripbrack:T])
```

Keywords: variables, glm, character variables

Usage

varnames(Model) returns as a CHARACTER vector the name of the response variable followed by the names of the factors and variates in Model in the order they first appear. Model should be a CHARACTER variable or quoted string.

Model must contain a legal GLM model, except that variables in the model need not be REAL or even be defined. Variables to evaluated "on the fly" are not evaluated and the expression is returned as the name enclosed in {...}. See topic 'models'.

NOTE: This represents a change from earlier versions. Previously varnames(Model) extracted the variable names without checking the validity of the model and did not handle expressions to be evaluated on the fly correctly.

varnames(Model, stripbrack:T) does the same, except any names which are expressions are not enclosed in {...}.

varnames(Model, stripat:T [,stripbrack:T]) does the same, except any non-expression names which are temporary variables have the leading '@' strippped off.

No model specified

varnames([keyword phrases]) without a model is equivalent to varnames(STRMODEL [,keyword phrases]), returning a vector of the names of the variables in STRMODEL. This is usually the model used by the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson().

Examples

```
Cmd> varnames("{log(y)} = @x + {@x^2}")
(1) "{log(y)}"
(2) "@x"
(3) "{@x^2}"
```

```
  Cmd> varnames("{log(y)} = @x + {@x^2}",stripat:T)
  (1) "{log(y)}"
  (2) "x"
  (3) "{@x^2}"

  Cmd> varnames("{log(y)} = @x + {@x^2}",stripbrack:T)
  (1) "log(y)"
  (2) "@x"
  (3) "@x^2"

  Cmd> varnames("{log(y)} = @x + {@x^2}",stripbrack:T,stripat:T)
  (1) "log(y)"
  (2) "x"
  (3) "@x^2"

                            Cross references
See also topics xvariables(), modelvars(), 'models', compnames(),
nameof(), 'glm'.
```

2.382 vboxplot()

Usage:

```
   vboxplot(x1,x2,...,xk [,vs:indv, boxsize:W] [,excludeM:T, boxtype:m,
     symbols:outlierSyms, graphics keyword phrases]), x1,...,xk REAL
     vectors, indv REAL length k vector with no MISSING values, m > 0
     integer, W REAL non-negative vector or scalar, outlierSyms CHARACTER
     scalar or vector of length 2
   vboxplot(Struc, [,vs:indv, boxsize:W] [,excludeM:T, boxtype:m,
     symbols:outlierSyms, graphics keyword phrases]), Struc a structure
     with k REAL vector components
```

Keywords: plotting, descriptive statistics

```
                                    Usage
vboxplot(var1, var2, ... , vark [,graphics keyword phrases]) produces
vertically oriented parallel Tukey boxplots for the vectors var1 through
vark.  It is identical with boxplot(var1, var2, ..., vark, vertical:T
[,graphics keyword phrases]).

vboxplot(Struc [,graphics keyword phrases]) produces vertically oriented
parallel box plots for the components of structure Struc, all of which
must be vectors.  It is identical with boxplot(Struct, vertical:T
[,graphics keyword phrases]).

You can use all the graphics keyword phrases that boxplot() recognizes.
Keyword 'symbols' is interpreted differently from other plotting
commands; see below.

                              Keyword 'symbols'
Keyword 'symbols' has a different meaning from other plotting commands.
You use it to specify symbols for moderate outliers (beyond inner fences
```

```
and inside outer fences) and extreme outliers (beyond outer fences).
The value of 'symbols' must be a CHARACTER scalar or vector of length 2.

  Cmd> vboxplot(x1, x2, x3, symbols:vector("\3", "\5"))

uses "\3" (square) as a symbol for moderate outliers and "\5" (triangle)
as a symbol for extreme outliers.  When the value of 'symbols' is a
scalar, the default symbol is used for extreme outliers.

                            Cross references
For more information including how to use split() to create a structure
argument, see boxplot().
```

2.383 vconcat()

Usage:

```
   vconcat(a,b,c,... [,labels:structure(rowLabs,colLabs), silent:T]) where
     a, b, c, ... matrices and rowLabs and colLabs are CHARACTER scalars or
     vectors
```

Keywords: combining variables, variables, null variables

```
                                    Usage
vconcat(a,b,c,...) combines matrices a, b, c ... vertically by
concatenating their columns.

All arguments must be of the same type, REAL, LOGICAL, or CHARACTER, and
have the same number of columns n.  The result is a matrix of that type
with n columns and ma+mb+mc+... rows, where ma, mb, mc, ... are the
number of rows of a, b, c, ... .

vconcat(a,b,...,labels:structure(rowLabs,colLabs) [,silent:T]) uses
CHARACTER scalars or vectors rowLabs and colLabs as row and column
labels for the result.  With silent:T, no warning is printed if labels
are the wrong size.  See topic 'labels' for details.

                            Non matrix arguments
Any argument that is a vector of length m is considered to be a m by 1
matrix.  In particular, if a is a vector of length m, vconcat(a) is a m
by 1 matrix.

An argument that is an array with only two dimensions not equal to 1 is
considered to be a matrix (see 'matrices').  For example,
   vconcat(array(run(6),1,3,2),array(run(7,14),4,1,2))
is equivalent to
   vconcat(matrix(run(6),3),matrix(run(7,14),4))

Any argument of type NULL is ignored.  When all arguments are NULL, so
is the result.

                            Cross references
```

```
See also topics hconcat(), 'matrices', 'vectors'.
```

2.384 vecread()

Usage:

```
vecread(FileName [keyword phrases]), FileName a CHARACTER scalar, (REAL
  data)
vecread(FileName, bywords:T [keyword phrases]) (CHARACTER data)
vecread(FileName, bylines:T [keyword phrases]) (CHARACTER data)
vecread(FileName, bychars:T [keyword phrases]) (CHARACTER data)
Keyword phrases are: silent:T, quiet:F, echo:F or echo:T, prompt:F,
  printname:F, badkeyok:T, nofileok:T, stop:stopChar or go:goChar,
  skip:skipChar, skipthru:skipthruChar, n:N, startline:M, bypass:P,
  badvalue:val, byfields:T, stopChar, goChar, skipChar CHARACTER scalars
  consisting of one character, N > 0, M > 0, P >= 0 integers, val a REAL
  scalar or MISSING.  See topic 'vecread_keys'.
FileName can also be CONSOLE or have the form string:charVal where
charVal is a CHARACTER scalar or vector.
```

Keywords: input, files

```
                              Introduction
This topic has sections on Reading REAL data, with examples, Reading
CHARACTER data, with examples, Reading a matrix with vecread(),
Controlling the lines to be read, Reading from a CHARACTER variable, and
Reading the console or batch file.

Topics 'vecread_file' and 'vecread_keys' provide additional information
on file format and keyword use for vecread().

vecread() reads data from a text file sequentially, row by row, starting
at the beginning of the file, interpreting items as numerical or
character data depending on keyword phrases.

                              General usage
The general usage of vecread() is

  Cmd> Var <- vecread(FileName [,keyword phrases])

where FileName is a quoted string or a CHARACTER variable.  In windowed
versions, when FileName is "", you are prompted to enter the file name
using a dialog box.  Var becomes a REAL or CHARACTER vector or possibly
(with 'nofileok:T') NULL.

                     Controlling the first line read
vecread(FileName [,keyword phrases], bypass:P), where P > 0 is an
integer, skips by all lines until P lines starting with the "stop
character" (default '!'; see below) have been read.  bypass:P can be
used with any other keyword phrases.  Other keywords affecting which
lines are read have no effect until after the P-th line starting with
the stop character.  This allows you to have several data sets in the
```

```
same file, separated by lines starting with the stop character.

vecread(FileName [,keyword phrases], startline:M), where M > 0 is an
integer, completely ignores the first M-1 lines in the file (or after
the P-th line starting with the stop character with 'bypass:P').
startline:M can be used with any other keyword phrases.

                          Reading REAL data
vecread(FileName) and vecread(FileName, byfields:T) read numbers from
the file with name FileName and return a REAL vector containing the
data.

Data of the form '?', '??', '???', ... as well as an isolated 'NA',
period '.'  or asterisk '*' are read as MISSING.

A number that is too large to be represented in the computer (for
example, -3.1e10000) is read as MISSING.

                          Keyword 'byfields'
Reading REAL data without byfields:T or with byfields:F
 The file should contain a sequence of numbers or missing value codes,
 separated by tabs, spaces or single commas.

 Unreadable items are skipped and an informative message is printed
 once.  Numbers are extracted from "words" like '-1.2a5' which is
 interpreted as if it were '-1.2 a 5'.  Single commas between items are
 ignored; a sequence of m commas is treated as m-1 unreadable items.

Reading REAL data with byfields:T
 The file is interpreted as a sequence of possible empty "fields"
 separated by commas, spaces, tabs and ends of lines, with each field
 becoming an element of the result.  A field that is not a number or
 missing value code is unreadable and is returned as MISSING.  This
 includes fields like '-1.2a5' that contain one or more digits.  Empty
 fields, before a leading comma, after a trailing comma and between two
 commas with no intervening visible characters, are returned as MISSING.

                          Keyword 'badvalue'
vecread(FileName, badvalue:BadVal [,byfields:T]), where BadVal is a REAL
scalar or MISSING (?), returns a REAL vector with BadVal substituted for
every unreadable item.  For example, when reading '-1.2a5 17 ?',
vecread(FileName, badvalue:-99) returns vector(-1.2,-99,5,17,?) and
vecread(FileName, byfields:T, badvalue:-99) returns vector(-99,17,?).
With byfields:T this enables you to distinguish between codes for
MISSING and non-numeric items.

                      Reading REAL data examples
File "data1.txt" looks like the following:
   Henry   Male   67.3,10.5
   Susan   Female 59.2,   ?

File "data2.txt" looks like the following (note the extra comma):
   Henry   Male   67.3,   10.5
```

```
   Susan   Female 59.2,  ,   ?

File "data3.txt" looks like the following (note digits in fields):
   Henry   Season_1 67.3,10.5
   Susan   Season_2 59.2,   ?

  vecread("data1.txt") returns vector(67.3,10.5,59.2,?).

  vecread("data1.txt",byfields:T) returns vector(?,?,67.3,10.5,?,?,59.2,
   ?).

  Both vecread("data1.txt", badvalue:-1) and vecread("data1.txt",
   badvalue:-1,byfields:T) return vector(-1,-1,67.3, 10.5,-1,-1,59.2,?).

  vecread("data2.txt", badvalue:-1) returns vector(-1,-1,67.3,10.5,-1,
   -1,59.2,-1,?).

  vecread("data2.txt", badvalue:-1,byfields:T) returns
   vector(-1,-1,67.3, 10.5,-1,-1, 59.2,?,?).

  vecread("data3.txt") returns vector(1,67.3,10.5,2,59.2,?), extracting
   1 and 2 from Season_1 and Season_2.

  vecread("data3.txt",badvalue:-1) returns vector(-1,-1,1,67.3,10.5,-1,
   -1,2,59.2,?).

  vecread("data3.txt",byfields:T) returns vector(?,?,67.3,10.5,?,?,59.2,
   ?), treating Season_1 and Season_2 as unreadable.

  vecread("data3.txt", byfields:T, badvalue:-1) returns
   vector(-1,-1,67.3, 10.5,-1,-1,59.2,?).

  vecread(string:",1,,2,3,",byfields:T) returns vector(?,1,?,2,3,?)
   (see topic 'vecread_keys' for the use of 'string').

                        Reading CHARACTER data
vecread(fileName, bywords:T), vecread(fileName, bylines:T) and
vecread(fileName, bychars:T) read CHARACTER data from a file.  The
latter two can read data containing commas, spaces or tabs or other
"invisible" characters.

                           Keyword 'bywords'
vecread(FileName, bywords:T) returns a CHARACTER vector, each element of
which is a "word" from the file.  For this usage, a word is a sequence
of printable non-blank characters, excluding commas.  Words are
separated by commas, or spaces, tabs or other "invisible" characters.

Quotation marks (") are not special and are treated as any other visible
character that is not a comma.

An "empty" word, before a leading comma, after a trailing comma, or
between successive commas with no intervening visible characters, is
returned as the null string "".
```

```
                          Keyword 'bylines'
vecread(fileName, bylines:T) returns a CHARACTER vector, each element of
which is an entire line read from file fileName. The lines do not
include an end-of-line character but do include any other "invisible"
or non-printing characters such as TABS.

                          Keyword 'bychars'
vecread(fileName, bychars:T) returns a CHARACTER vector, each element of
which is a single character read from file fileName, including any
end-of-line characters (returned as "\n") or other invisible characters.

                    Reading CHARACTER data examples
Here are more examples of reading the sample files used above to
illustrate reading REAL data:

  vecread("data1.txt", bywords:T) returns vector("Henry","Male",
   "67.3","10.5","Susan","Female","59.2","?").

  vecread("data2.txt", bywords:T) returns vector("Henry","Male",
   "67.3","10.5","Susan","Female","59.2","","?").

  vecread("data1.txt", bylines:T) returns
   vector("Henry   Male   67.3,10.5", "Susan   Female 59.2,   ?").

  vecread("data1.txt",bychars:T) returns vector("H","e","n","r","y"," ",
   " "," ","M","a","l","e"," "," "," ", "6","7",".","3",",","1","0",".",
   "5","\n","S","u","s","a","n"," "," ", " ","F","e","m","a","l","e","
   ", "5","9",".","2",","," "," "," ","?", "\n")

  vecread(string:",,a,b,", bywords:T) returns vector("","","a","b","")
    (see below for the use of 'string').

The following creates macro isnumber that tests whether each "word" of
a CHARACTER scalar or vector represents a valid number:

  Cmd> isnumber <- macro("@tmp <-  paste(vecread(string:$1,bywords:T))
    !ismissing(vecread(string:@tmp,badvalue:?,byfields:T))",\
    dollars:T)

Then
  isnumber("3.45") returns True, isnumber("3b45") returns False, and
  isnumber("3.4 4.5 A") returns vector(T,T,F).

                    Reading a matrix with vecread()
When the file contains a data matrix consisting of n rows of data, each
of k items, you can read the data into a n by k matrix by
  Cmd> x <- matrix(vecread(FileName [,byfields:T]),k)'
or, for CHARACTER data,
  Cmd> x <- matrix(vecread(FileName, bywords:T),k)'

The transpose is needed because vecread() reads row by row, but matrices
are filled column by column.
```

```
If there are several matrices in a file, separated by lines starting
with the stop character (default "!"), you can read the third one, say,
by
  Cmd> x <- matrix(vecread(FileName, bypass:2), k)'

                    Controlling the lines to read
In addition to 'bypass:P' and 'startline:M', keyword phrases 'skip:C',
'skipthru:C', 'stop:C' and 'go:C' control which lines will be scanned
for data.  In each case C is a single character such as "#" or "!" .
These are referred to as the "skip character", the "skipthru character",
the "stop character" and the "go character".  Except for the stop
character, these have no effect until after the lines skipped by
bypass:P and startline:M.

The default stop character is "!", whether reading numerical or
CHARACTER data.  That is, a '!' terminates scanning the file.  There are
no defaults for the skip, skipthru or go characters.

Briefly, lines starting with the skip character are skipped, as are all
lines up to and including the first line starting with the skipthru
character, and reading is terminated by the stop character or a line
that does not start with the go character.  When the skipthru character
is "\n", reading will start after the first completely empty line.  See
topic 'vecread_keys' for details.

If '!' appears in the file as other than a stop character, you should
use 'stop:C', where C is a character that does not occur in the file.
If the file consists solely of standard ASCII characters, 'stop:"\377"'
is a good choice.

With 'bylines:T' and 'bychars:T' and when skipping lines as controlled
by 'bypass:P', the stop character is recognized only as the first
character in a line.  With 'bywords:T' or 'byfields:T' it is recognized
only as the first character in a word or field.  Otherwise it is
recognized at any position in a line.

When you use keyword phrase n:N (see above), reading is terminated when
N items have been read.  When a stop character is found or a line that
does not start with the go character is found before N items have been
read, reading is stopped and a warning message is printed.

                  "Reading" from a CHARACTER variable
vecread(string:CharVar [, keywords]) where CharVar is a CHARACTER scalar
or vector, does not read from a file.  Instead, it "reads" CharVar as if
were a file.  See topic 'vecread_keys' for details.

Here is a particularly useful way to use keyword 'string':

  Cmd> x <- vecread(string:CLIPBOARD [,byfields:T])
and
  Cmd> x <- vecread(string:CLIPBOARD,bywords:T)
```

would read data from the special variable CLIPBOARD. In windowed versions, this would be taken from the Clipboard. Pre-defined macro fromclip() makes use of this feature to read data on the Clipboard. In the GTK version, you can use special variable SELECTION in a similar way to read the current X selection. See topic 'CLIPBOARD'.

Reading from the keyboard or batch file

vecread(CONSOLE [,keywords]) reads what you type rather than a file. If a variable CONSOLE exists, its value is ignored. In windowed versions, a dialog box is displayed in which you enter data; in other versions, you are prompted to type in the data. The prompt can be suppressed by 'prompt:F'.

Data should be typed in one of the formats just described. To stop input, type the stop character (default '!'), followed by RETURN, or, if you provided a go character (see 'vecread_keys'), type a line starting with any other character. In windowed versions, clicking on the "Done" or "Cancel" button in the dialog box also ends input.

In a batch file vecread(CONSOLE [,...]) reads the immediately following lines as the data file. For this usage it is essential that either a stop character terminates the data or keyword phrase n:N limits number of items read. You will probably want to use 'prompt:F' in this case. See also batch().

Other optional keywords phrases that may always be used

You can also specify the file name by 'file:FileName', which need not be the first argument. In addition, you can replace FileName by 'string:CharVar', where CharVar is a CHARACTER scalar or vector which is "read" as if it were a file.

You can use keyword phrases 'echo:T', 'quiet:T', 'quiet:F', 'silent:T' and 'printname:F' to control what vecread() prints.

You can ignore duplicated or unrecognized keywords by 'badkeyok:T'. This is useful in writing macros, since you can have a line like

```
  @x <- vecread(string:@lines, stop:"$", badkeyok:T, $K)
```

without checking to see whether 'string', 'stop' or a non-vecread keyword is a macro argument and thus included when $K is expanded. See topics 'macros' and 'macro_syntax'.

Keyword phrase 'nofileok:T' instructs vecread() to return NULL instead of aborting when it is unable to open a file. This is useful in writing robust macros that use vecread().

See topic 'vecread_keys' for details on these keywords.

Cross references

See also topics readcols(), read(), matread(), macroread(), batch(), 'vecread_file', 'vecread_keys', 'files', console(), 'vectors'.

2.385 vecread_file

Keywords: variables, files, input, output

Introduction

This topic discusses the format of files vecread(), readcols() and readdata() can read. Such files are plain text files which contain only REAL or CHARACTER data in unstructured format.

Macro readcols() uses vecread() to read a file. The only difference in file format is that, when no variable names are provided to readcols() or readdata(), the first line of the file is interpreted as containing variable names.

See topic 'matread_file' for information on the format of files to be read by matread() and read().

See topic 'vecread_keys' for information on vecread() keywords.

Multiple data sets in file

With the help of keyword 'bypass' you can read one of several sets of data in the same file if the data sets are separated with lines beginning with the "stop character" (default "!"; see 'vecread_keys'). You can read the third set, say, by including 'bypass:2' as an argument to vecread() or readcols(). This discussion really describes the lines after the bypassed data.

Keyword 'bypass' cannot be used with readdata().

REAL data in file

REAL data to be read by vecread(), readcols() and readdata() consist of numbers and codes for MISSING, often on several lines. When there are several data items on a line, they are separated by spaces, tabs or commas. Any of '?', '.', '*' and 'NA' code for MISSING. '??', '???', ... are treated as a single missing value.

Interpretation of extra commas and non-numeric "fields" (sequences of characters that are not commas, spaces or tabs) depends on whether 'byfields:T' is an argument to vecread() or readcols(). See topic 'vecread_keys' for details.

readdata() always uses 'byfields:T' in reading REAL data,.

What is done with items that are not numbers or missing value codes depends on whether 'badvalue:badv' is an argument to vecread(), readcols() or readdata(), where badv is a REAL scalar or MISSING. See topic 'vecread_keys'.

CHARACTER data in file

CHARACTER data to be read by vecread() or readcols() have no special format when 'bylines:T' or 'bychars:T' is an argument. When 'bywords:T' is an argument, the file is interpreted as consisting of "words" separated by "white space" or commas. A word is a sequence of visible characters other than commas. An empty word, read as "", is assumed

before a leading comma, after a trailing comma or between two commas enclosing no visible characters.

When all the data lines are at the start of the file and start with the same character, for example '%', you can restrict reading to them by including 'go:"%"' as an argument to vecread() or readcols().

Since keywords 'bywords', 'bychars' and 'bylines' are illegal for readdata(), it cannot be used for reading CHARACTER data;

Stop character

Data can be terminated by a "stop character" (default is '!'). With 'bychars:T' and 'bylines:T' this is recognized only as the first character in a line.

You can change the stopping character by, say, 'stop:"$"'. See topic 'vecread_keys' for details.

With readdata() or with 'byfields:T' and 'bywords:T' on vecread() or readcols(), the stopping character must be at the start of a field or word. In other situations, it can be anywhere in the file.

Skip character

Lines starting with a "skip character" specified by an argument of the form, say, skip:"#" are skipped. See topic 'vecread_keys' for more information.

The default skip character is '#'. If you don't want any skip character, perhaps because there are lines in the file starting with '#' that should be read, use 'skip:""' as an argument (not with readdata()).

Writing a data file to be read by vecread()

You can write a file vecdata.txt of REAL data that vecread(), readcols() and readdata() can read by

```
Cmd> print(x,new:T,file:"vecdata.txt",header:F,labels:F,missing:"?")
```

where x is a REAL vector or matrix. When x is a matrix, it is written row by row as readcols() expects. If you want it written column by column, use x' as an argument. You can specify the format or the number of significant digits by keywords 'format' and 'nsig'. See print().

2.386 vecread_keys

Usage:

```
   List of keywords that can be used on vecread() or readcols().
    Keyword              Value
    badkeyok             T or F*
    badvalue             REAL scalar or MISSING
    bychar               T or F*
    byfields             T or F*
    bylines              T or F*
    bypass               Nonnegative integer
    bywords              T or F*
    echo                 T or F
    file                 CHARACTER scalar file name
    go                   CHARACTER scalar with one character
    n                    Positive integer
    nofileok             T or F*
    printname            T or F*
    prompt               T* or F
    quiet                T or F*
    silent               T or F*
    skip                 CHARACTER scalar with one character, default "#"
    skipthru             CHARACTER scalar with one character
    startline            Positive integer
    stop                 CHARACTER scalar with one character
    string               CHARACTER scalar or vector
   * Default
```

Keywords: files, input, output

```
                              Introduction
This topic summarizes the use of keyword phrases as arguments to
vecread() which reads data items row by row from a text file.  Type
usage(vecread_keys) for a list of vecread() keywords.  See also topics
vecread() and 'vecread_file'.

Some of the examples use keyword 'string'.  See below for information.

     Keyword phrases specifying that CHARACTER data should be read
                           Keyword 'bywords'
bywords:T             Read CHARACTER data by words
  The file is interpreted as a sequence "words" or fields separated by
  commas, or "invisible" characters (spaces, tab characters and ends of
  lines).  The result is a CHARACTER vector, each element of which is a
  word from the file.

  Each word is empty or consists of a sequence of visible characters
  that are not commas.  An empty word occurs before a leading comma,
  after a trailing comma or between two commas with no intervening
  visible characters and is read as "".  For example
  vecread(string:",a ,, b,",bywords:T) returns vector("","a","","b","")

                           Keyword 'bylines'
bylines:T             Read CHARACTER data by lines
```

```
  Each line read becomes a single element of a CHARACTER vector.  The
  line-separating character "\n" is not included.  All other characters,
  including commas and invisible characters are read.  If no line starts
  with the stopping character, the length of the result is the number of
  lines in the file.

                            Keyword 'bychar'
bychar:T                Read CHARACTER data as separate characters
  Each character read, including commas, invisible characters and line
  separating characters, is read as a single element.  If no line starts
  with the stopping character, the length of the result is the number of
  characters in the file.

          Keyword phrase specifying how REAL data will be read
                           Keyword 'byfields'
byfields:T              REAL items read by "fields"
  The file is interpreted as a sequence of "fields" separated by commas
  or "invisible" characters.  The result is a REAL vector, each element
  of which is derived from a field in the file.

  A field is either empty or consists of a sequence of visible
  characters that are not commas.  An empty field occurs before a
  leading comma, after a trailing comma or between two commas with no
  intervening visible characters and is read as MISSING.  A non-empty
  field that is a well formed number (for example, -17, 3.14e73 or
  2.7182818) is read as that number.  If the number is too large to be
  represented in the computer (for example -3.1e3000), it is read as
  MISSING.  When a non-empty field is not a well formed number (for
  example 3.5a7 or Foo), it is read as badV when 'badvalue:badV' is an
  argument, or as MISSING otherwise.

                Keyword phrases specifying what to read
                             Keyword 'file'
file:FileName           Read from file FileName
  This is an alternate way to specify the file to be read.  That is
  vecread(file:FileName [,keyword phrases]) is equivalent to
  vecread(FileName [,keyword phrases]).  Filename must be a quoted
  string or CHARACTER scalar.

                            Keyword 'string'
string:CharVar          "Read" from CHARACTER scalar or vector CharVar
  CharVar is "read" as if it were a file whose contents are specified by
  CharVar.  Any instances of "\n" are treated as terminating a line.
  When CharVar is a vector, each element is assumed to start a new line.

  When CharVar is a CHARACTER vector, each element is read as for a
  CHARACTER scalar, with each element starting with a new line.  With
  bychar:T, "" is inserted between each element of CharVar.

  Examples:  Suppose S1 is "12\n34" and S2 is vector("12","34").

  vecread(string:S1 [,byfields:T]) and vecread(string:S2 [,byfields:T])
  return vector(12,34).
```

```
  vecread(string:S1, bywords:T or bylines:T) and vecread(string:S2,
  bywords:T or bylines:T) return vector("12","34")

  vecread(string:S1, bychars:T) returns vector("1","2","\n","3","4")

  vecread(string:S2, bychars:T) returns vector("1","2","","3","4")

                              Keyword 'n'
       Keyword phrase limiting the number of items to be returned
n:N                  No more than N items to be returned
  If fewer than N items are found before the stop character or the end
  of the file, a warning message is printed.  N must be a positive
  integer.

  Keyword 'n' is illegal as an argument to readdata().

     Keyword phrases specifying which lines should be read as data.
                             Keyword 'stop'
stop:"$"              Set stop character to '$' (default is "!")
  Reading stops when a stop character is encountered.  With 'bychars:T'
  and 'bylines:T', a stop characters is recognized only as the first
  character in a line.  With 'byfields:T' and 'bywords:T' it is
  recognized only as the first character in a field or word.

  For numerical data, the stop character can be any punctuation
  character except '+', '-', ',', '?'  or  '.', that is, any of
  !"#$%&'()*/:;<=>@[\]^_`{|}~.

  For CHARACTER data, the stop character can be any punctuation
  character except ',', including '+', '-', '?' or '.'.

  For both numerical and CHARACTER data, the stop character can also be
  a "non-ASCII" character (code >= 128 = octal 200 = hexadecimal 80)
  such as "\200", "\377", "\x80" or "\xff", except when reading from
  CONSOLE.  This might be called for when you want to ensure the whole
  file is read, and you know the file does not contain non-ASCII
  characters.

                             Keyword 'bypass'
bypass:P              Skips past P lines starting with stop character
  When P > 0, nothing is read or echoed until P lines starting with
  the stop character have been read.  This allows you to read several
  sets of data on the same file provided they are separated by lines
  starting with the stop character.  This takes precedence over all
  other keywords that control which lines are read.  'bypass' is
  illegal with readdata().

                            Keyword 'startline'
startline:M           Sets the first line to be scanned.
  Start reading on line M of the file, that is, the first M-1 lines are
  ignored and never echoed.  This takes precedence over all other
  keywords except bypass that control which lines are read.  Stop,
```

```
  skip, skipthru, and go characters are not recognized if they come
  before line M.  The default value for M is 1.  With bypass:P, reading
  starts on line M after the P-th line starting with the stop character.

                            Keyword 'skipthru'
skipthru:"@"           Set skipthru character to '@' (no default)
  Lines up to and including the first line starting with the skipthru
  character are ignored.  They may be echoed, depending on the values of
  keywords 'echo', 'quiet' and 'silent'.

  The skipthru character can be any character.  When the skipthru
  character is "\n", reading starts after the first completely "white"
  line, that is, a line with no visible characters.

  The skipthru character takes precedence over and can be the same as
  the skip, stop or go characters.  It is ignored until after the lines
  skipped because of keywords 'bypass' and 'startline'.

                               Keyword 'go'
go:"%"                 Set go character to '%' (no default)
  Reading stops with the first line that does not start with the go
  character.  It is ignored until after the lines skipped because of
  keywords 'bypass' and 'startline'.

                              Keyword 'skip'
skip:"#"               Set skip character to '#' (default is "#")
  Lines starting with skip character are ignored (they may be echoed).

  For numerical data, the allowable skip characters are the same as for
  stop characters.

  For CHARACTER data, any visible printing character, including a comma,
  may be a skip character.

  When the skip character is the same as the stop character (for
  example, stop:"#",skip:"#"), scanning will be stopped only if the stop
  character occurs after the first character in a non-skipped line (for
  example, by a line starting " #..." but not by "#...").

              Keyword phrases controlling what is printed
                              Keyword 'quiet'
quiet:F                Prints skipped lines, if any
  The lines printed are those skipped because of the skip or skipthru
  characters, but not those skipped by 'bypass:P' or 'startline:M'.

                              Keyword 'echo'
echo:T or F            Control echoing of lines scanned.
  With echo:T, all lines scanned, including lines skipped because of the
  skip or skipthru character, are echoed to output.  With echo:F, the
  only lines echoed are skipped lines with quiet:F.

  When FileName is CONSOLE and the vecread() command is in a batch file,
  data that is read is echoed to output unless echo:F is an argument.
```

```
  This also happens in windowed versions even when not reading from a
  batch file.

                              Keyword 'silent'
silent:T               Suppress all warning messages and echoing
  This is incompatible with echo:T or quiet:F.

                            Keyword 'printname'
printname:F             Suppress printing the name of the file read
  The default is printname:T except with string:CharVar.

                              Keyword 'prompt'
prompt:F               Suppresses prompt when reading from CONSOLE

                   Keyword phrases related to anomalies
                            Keyword 'badkeyok'
badkeyok:T             Ignore unrecognized or duplicate keywords
  Without badkeyok:T, unrecognized or duplicate keywords are considered
  errors.  This feature is useful in a macro where a line like
  @x <- vecread("data.txt",silent:T,badkeyok:T,$K) should work,
  even if the argument list to the macro includes keyword 'silent' or
  other keywords not recognized by vecread().

                            Keyword 'badvalue'
badvalue:badVal        Replace unreadable REAL items by badVal
  When reading REAL data with badvalue:badVal an argument, any
  unreadable items are replaced by badVal, which must be a REAL scalar
  or MISSING.  Without badvalue:badVal, unreadable items are replaced by
  MISSING when 'byfields:T' is an argument and are skipped otherwise.

                            Keyword 'nofileok'
nofileok:T             Failure to open the file is OK.
  When a file can't be opened, NULL is returned and no error message is
  printed.  Without nofileok:T, this is an error.
```

2.387 vector()

Usage:

```
   vector(x1,x2,...,xk [,KeyPhrases]) where x1, x2, ... all have the same
     type, REAL, LOGICAL, or CHARACTER, or are structures with components
     all of the same type
   KeyPhrases can be labels:lab, notes:Notes and silent:T, where labs and
     Notes are CHARACTER scalars or vectors.
```

Keywords: variables, combining variables, character variables, null variables

```
                                    Usage
vector(x1, x2, ..., xk) combines scalars x1, x2, ... xk into a vector of
length k.  For example, you can enter a small set of data by
  Cmd> x <- vector(3.5, 9.6, 2.5, 2.3, 7.7, 2.6, 6.3, 6.5, 6.6, 4.1)
```

```
The arguments x1, x2, ..., xk can be REAL, LOGICAL or CHARACTER and must
all have the same type.  For example, you can create a CHARACTER vector
by
  Cmd> varNames <- vector("Length","Width", "Weight")

vector() is identical to cat().  However, cat() is a deprecated
function, that is, it will remain available for the immediate future,
but at some time it may be disabled.  Use vector() instead.

                          Non scalar arguments
Arguments may also be vectors.  In that case vector(x1, x2, ...., xk)
combines all the arguments into a single vector.  For example,
  Cmd> vector(run(3), run(3,1))
is equivalent to
  Cmd> vector(1,2,3,3,2,1).

More generally, any or all of the arguments may be vectors, matrices or
arrays, as long as they all have the same type.  In that case,
vector(x1, x2, ...., xk) has the same effect as vector(vector(x1),
vector(x2), ...., vector(xk)), combining all the elements of its
arguments into one long vector.  See the next paragraph for what
vector(x) does when x is a matrix or array.

vector(x) creates a vector from a matrix or array x by "unravelling" it,
with the first subscript changing fastest, the second changing next
fastest, etc.  Specifically, if the dimensions of x are n1, n2, ..., nk,
vector(x) is a vector with length n1*n2*...*nk, with elements
x[1,1,...,1], x[2,1,...,1], ..., x[n1,1,...,1], x[1,2,...,1],
x[2,2,...,1], ..., x[n1,2,...,1], ..., x[1,3,...,1], ...,
x[n1,n2,...,nk].  x may be REAL, LOGICAL, or CHARACTER.

                          Structure argument
When Str is a structure with n components, vector(Str) is a vector
equivalent to vector(vector(Str[1]),...,vector(Str[n])), defined
recursively if any component is a structure.  All the data components
must be of the same type, REAL, LOGICAL or CHARACTER.  This should not
be confused with strconcat() which combines structures into a larger
structure.

                             NULL argument
Any argument of type NULL is ignored.  For example, vector(NULL, a) or
vector(a, NULL) are equivalent to vector(a).  When all arguments to
vector() have type NULL, so does the result.

                        Attaching labels or notes
You can specify labels for the first (and only) dimension of the result
using keyword 'labels'.  See topic 'labels' for details.

You can attach a CHARACTER vector of descriptive notes to the result
using keyword phrase 'notes:Notes'.  See topic 'notes' for details.

After
  Cmd> y <- vector(x)
```

```
where x is already a vector, y will have the same coordinate labels or
descriptive notes as x.

                            Cross references
See also topics 'vectors', 'structures'.
```

2.388 vectors

Usage:

```
   Create a vector:    x <- vector(x1,x2, x3, ...)
   Extract element(s)  x[i], i an integer scalar or vector or LOGICAL
     vector
```

Keywords: variables, syntax

```
                                Description
A vector is an array with only one dimension, its length.  It can be
thought of as just a list of one or more numbers (REAL vector), logical
values (LOGICAL vector) or strings (CHARACTER vector).  You can't mix
different types of data in a single vector.  For that you need a
structure (see structure() and 'structures').

The most usual way to create a vector is to use vector():

                                 Examples
  Cmd> x <- vector(1,3,2.5,6) # create REAL vector

  Cmd> y <- vector("Hi","Lois") # create CHARACTER vector

  Cmd> z <- vector(T,T,F,F,T) # create LOGICAL vector

  Cmd> list(x,y,z) # all are vectors, having only 1 dimension
  x               REAL   4
  y               CHAR   2
  z               LOGIC  5

                            Vectors as matrices
Practically always, MacAnova commands treat a n by 1 matrix (column
vector) or a n by 1 by 1 ... by 1 array the same as a vector.  For
example, if y is a n by p matrix, then factor(y[,3]) is legal, even
though factor() expects a vector as argument, because y[,3] has only one
column.  Moreover, isvector(x) is True when x is a vector, a n by 1
matrix, or a n by 1 by 1 ... by 1 array.

Conversely, in a context requiring a matrix, practically always MacAnova
treats a vector of length n as a n by 1 matrix.

                            Cross references
See also topics vector(), isvector(), factor(), 'matrices'.
```

2.389 vt()

Usage:
```
vt()
```

Keywords: plotting

```
                                  Usage
vt() (no argument) is designed to be used with a terminal emulator
implementing both VT100 and Tektronix 4014 emulation.  When running
MacAnova using such an terminal emulator, vt() automatically puts your
terminal in vt100 emulation mode using the character sequence obtained
by getoptions(tekset:T)[2].  See topic 'options'.

vt is normally not needed, since MacAnova should automatically switch
back to vt100 mode when a plot is finished if option 'tekset' has been
set appropriately.  See subtopic 'options:"tekset"'.

vt is implemented as a pre-defined macro (Unix/Linux versions only).

See topic vtx() if you are running MacAnova in an Xterm window on a
workstation.

                             Cross references
See also tek(), plot(), chplot(), lineplot(), boxplot(), showplot().
```

2.390 vtx()

Usage:
```
vtx()
```

Keywords: plotting

```
                                  Usage
vtx() switches a Unix/Linux workstation Xterm terminal emulator to vt100
mode from Tektronix 4014 mode.  You don't normally need vtx since
MacAnova recognizes when it is running in a Xterm environment (the value
of environmental variable $HOME is "xterm") and automatically switches
back to vt100 mode after drawing a high resolution graph.

vtx is implemented as a pre-defined macro (Unix/Linux versions only).

See also topics tek(), tekx(), vt(), 'graphs', 'unix'.
```

2.391 while

Usage:
```
while(Logical){statement1;statement2;...;}
```

Keywords: syntax, control

```
                                  Usage
while(Logical){statement1;statement2;....;} repeatedly executes the
statements enclosed in '{' and '}' as long as Logical has value True.
Logical should be a scalar LOGICAL variable or expression, but not a
constant expression.  Unless the last statement in {...} is empty (';;'
just before '}'), its value may be printed on every repetition.

The opening '{' must be on the same line as 'while' unless that line
terminates with '\'.

To avoid "infinite" loops, a 'while' loop will automatically terminate
after 1000 repetitions.  This limit can be changed by option
'maxwhile'.  After
  Cmd> setoptions(maxwhile:10000)
a 'while' loop will terminate after 10000 repetitions.

It is essential that one of the statements modify the variable(s) used
in the LOGICAL expression, or that a break or breakall statement is
used.  Otherwise the loop will repeated until 1000 (or value reset by
option 'maxwhile') repetitions are completed.

A 'while' statement does not have a value.  Hence such constructs as yyy
<- while(n > 0){...} or zzz + while(n > 0){...} are illegal.

            Syntax elements 'break', 'breakall; and 'next'
You can terminate a "while loop" prematurely using syntax elements
'break' and 'breakall' or pre-defined macro breakif().

You can skip to the end of a "while loop" using syntax element 'next'.
Be careful to modify the variable(s) used in the LOGICAL expression
before 'next' so as to avoid an "infinite" loop.

                                 Examples
Examples:
  Cmd> @s <- 0;@n <- length(x);while(@n>0){\
        @s <- @s+x[@n];@n <- @n-1;;};@s

This would print the sum of all the elements in x.

  Cmd> ex <- macro("@x <- argvalue($1,\"x\",\"real nonmissing\")
                  @dims <- dim(@x);@x <- vector(@x)
                  @neg <- 1-2*(@x<0);@x <- abs(@x);@eps <- 1e-12
                  @s <- @t <- @k <- 1
                  while(max(@t) > @eps*max(@s)){
                    @t <- (@x/@k)*@t
                    @s <- @s+@t;@k <- @k+1
                  }
                  array(@s^@neg,@dims)",dollars:T)

will create a macro ex() such that ex(x) computes exp(x) using a power
```

series, when x is a vector, matrix, or array. See topics 'macro_syntax', argvalue(), macro() for information about writing macros.

Cross references
See also topics 'for', 'if', 'break', 'breakall'.

2.392 workspace

Usage:

```
The "workspace" consists of all MacAnova variables and macros.
Saving the workspace to a file
  save(fileName) saves whole workspace in binary format
  save(fileName,var1 [,var2 ...]) saves variables in binary format
  asciisave(fileName) saves whole workspace in text format
  asciisave(fileName,var1 [,var2 ...]) saves variables in text format
Restoring the workspace from a file
  restore(fileName [keywords])
Seeing contents of workspace
  listbrief([var1, var2, ...] [keywords]) just lists names
  list([var1, var2, ...] [keywords])  includes details
```

Keywords: general

Definition of workspace

The "workspace" consists of all MacAnova variables and macros. These include not only variables and macros explicitly created by you as you use MacAnova, but all pre-defined macros such as hist(), pre-defined variables such as PI and variables such as RESIDUALS created as "side-effects" of MacAnova functions.

The workspace "resides" in memory (RAM) and not on disk. One consequence is that when you quit MacAnova, your variables are lost. Before you quit you can save a copy of the workspace on disk using save() or asciisave(). In a later MacAnova session, you can use restore() to recover the workspace you saved. See save(), asciisave() and restore().

Cataloging your workspace

To get a catalog of everything in your workspace, type 'list()' or 'listbrief()'. Using keyword phrases such as 'real:T' or 'nrows:100' you can restrict the catalog to variables of particular types or sizes. See list() and listbrief() for details.

Deleting items

Use delete() to remove items from your workspace. Be careful not to delete something you want to keep since there is no "undelete" function.

2.393 write()

Usage:

```
write(a, b, ...[,format:Fmt or nsig:m, header:F, labels:F, notes:T,\
  width:w, height:h, macroname:T, missing:missStr, name:setName]\
  [, file:fileName [,new:T]]]), Fmt, missStr, fileName, setName
  CHARACTER scalars, m > 0, w >= 30, h >= 12 integers
```

Keywords: output, files

```
                                  Usage
write(a, b, ...) prints objects (variables, expressions, macros) a, b,
..., ordinarily retaining more decimal places than print().  By default,
write() formats REAL items using the format identified by 'wformat' on
getoptions() output.  By default this specifies 9 significant digits in
floating point form but may be changed by setoptions().

Except for using a different default format, write() is identical with
print(), and recognizes the same keywords.  It is provided as an easy
way to print results with more significant digits than does print(),
without having explicitly to specify a format.  See print() for a full
description of the various keywords and output.

  Cmd> print(PI*run(5))
  VECTOR:
  (1)      3.1416      6.2832      9.4248      12.566      15.708

  Cmd> write(PI*run(5))
  VECTOR:
  (1)       3.14159265       6.28318531       9.42477796
  (4)       12.5663706       15.7079633

When either of keywords 'nsig' or 'format' are used, write() behaves
identically to print().

                             Cross references
See also topics setoptions(), matprint(), matwrite(), macrowrite(),
'options'.
```

2.394 writedata()

Usage:

```
writedata(fileName,x1,x2,... [,missing:M] [, putNames:F] \
  [,fieldwidth:w  or  format:fmt]), CHARACTER scalar fileName, vectors
  x1, x2, ..., all with same length, CHARACTER scalars M, fmt,
  integer w > 0
writedata(x1,x2,..., keep:T [,missing:M] [, putnames:F] \
  [,fieldwidth:w  or  format:fmt])
```

Keywords: output

```
                                Introduction
```

writedata() is a macro designed to write one or more data vectors of arbitrary as columns of a data file. By default, columns are headed by the variable names so reading the file using readdata() should restore the variables. Optionally, writedata() can return a CHARACTER scalar containing an image of the file instead of writing it.

Usage

writedata(fileName,x1,x2,...) writes data vectors x1, x2, ... as columns in the file named by CHARACTER scalar fileName.

x1, x2, ... must be vectors of any type, REAL, LOGICAL or CHARACTER. If any vector is a factor and has row labels such that all cases with a factor level have the same label, the row labels are written instead of the factor levels.

REAL data are formatted using the current default format as returned by getoptions(format:T), except that integer values are written as integers with no decimal point.

CHARACTER data is written right justified in a field whose width is taken from the default format.

LOGICAL data are translated to "T" or "F" and then written like CHARACTER data.

MISSING values are written as "?" unless keyword 'missing' provides another string; see below.

If a data vector is specified by a keyword phrase (x1:X[,1], for example), the keyword is used as the vector name. For this usage, the keyword may not be 'keep', 'new', 'fieldwidth' or 'missing'.

Any data already in the file is destroyed unless 'new:F' is an additional argument.

writedata(x1,x2,...,keep:T) acts similarly, except that no file is written. Instead, what would be written is returned as a CHARACTER scalar to be assigned to a variable or CLIPBOARD.

writedata(fileName, x1, x2, ..., missing:M) and writedata(x1, x2, ..., keep:T,missing:M) do the same, except that CHARACTER scalar M (for example, "NA" or "*") is used instead of "?" for MISSING values.

By default, each column has the same width. You can suppress this behavior, so there is minimual space between columns, by keyword 'format'; see below.

Keyword 'putnames'

writedata(fileName, x1, x2, ..., putnames:F) and writedata(x1, x2, ..., keep:T, putnames:F) do the same, except that no line of column headers is written or returned.

Keywords 'format' and 'fieldwidth'

```
When 'format:Fmt' is an argument, REAL values are written using format
Fmt.  Fmt must be a CHARACTER scalar which is a legal value for print()
keyword 'format'.  For details, see subtopic
print:"details_on_format_keyword".  If Fmt is of the form "w.dg" or
"w.df", where w and d are integers, each column will have width at least
w.  If Fmt is of the form ".dg" or ".df", there will be minimal space
between columns and columns will probably not be straight.

When fieldwidth:w is an argument, where w is a positive integer, columns
will be w characters wide and the format will be "w.dg", where d =
max(w-7,0).  You can't use 'fieldwidth' with 'format'.

                            Cross references
See also readdata(), clipwritedat(), print(), 'keywords', 'files'
```

2.395 wtanova()

Usage:

```
wtanova([Model] ,Wts [print:F or silent:T, coefs:F, pvals:T, fstats:T]),
  Model a CHARACTER scalar, Wts a REAL vector.
```

Keywords: glm, anova

```
                                    Usage
wtanova(Model,Wts) is equivalent to anova(Model,weights:Wts).  See
anova() for details.
```

2.396 wtmanova()

Usage:

```
wtmanova([Model] ,Wts[,print:F or silent:T, coefs:F, pvals:T, fstats:T,\
   sssp:F or T]), Model a CHARACTER scalar, Wts a REAL vector.
```

Keywords: glm, anova

```
                                    Usage
wtmanova(Model,Wts) is equivalent to manova(Model, weights:Wts).  See
manova() for details.
```

2.397 wtregress()

Usage:

```
wtregress([Model], Wts [, print:F or silent:T, pvals:T])
  Model a CHARACTER scalar, Wts a REAL vector.
```

Keywords: glm, regression

Usage

wtregress(Model,Wts) is equivalent to regress(Model,weights:Wts). See regress() for details.

2.398 xrows()

Usage:

```
xrows(variates [,factors]), variates and factors REAL matrices or
  vectors or NULL.
```

Keywords: glm

Usage

xrows(Variates, Factors) computes rows of an X-variable (design) matrix corresponding to variate values in Variates and factor levels in Factors, using the information saved by the preceding GLM command.

In the following, let nv = number of variates in the model and nf = the number of factors in the model.

Variates should be either a REAL vector with length(Variates) = nv, or a REAL matrix with ncols(Variates) = nv. when nv = 0, Variates should be NULL.

Factors should be either a REAL vector with length(Factors) = nf or a REAL matrix with ncols(Factors) = nf. All the elements of Factors should be positive integers not exceeding the maximim level for each factor. When nf = 0, Factors should be NULL or should be omitted entirely.

Let nrowv be 1 if Variates is a vector and nrows(Variates) otherwise, and let nrowf be 1 if Factors is a vector and nrows(Factors) otherwise. Then if nrowv != nrowf, you must have either nrowv = 1 or nrowf = 1 and the single row of Variates or Factors is used for every row of the output.

The result is a REAL matrix with max(nrowv, nrowf) rows. Each row consists of the values of the X-variables (design matrix) corresponding to that row of Variates and Factors.

Examples

Examples:

After anova("y=x1+x2+a*b"), xrows(hconcat(x1,x2), hconcat(a,b)) is equivalent to xvariables().

After regress("y=x1+x2"); xrows(x0) %*% COEF is equivalent to regpred(x0,seest:F,sepred:F) where x0 is a matrix of values for x1 and x2.

Cross references

See also topics xvariables(), modelinfo(), regpred(), glmpred(), 'glm'.

2.399 xvariables()

Usage:

```
xvariables(Model [, missing:val]), Model a CHARACTER scalar, val a REAL
  scalar
```

Keywords: glm

```
                                  Usage
xvariables(Model) returns the full design matrix (matrix of X-variables)
associated with Model, including a column for the intercept (constant
term), if any.  Model must be a scalar CHARACTER variable or quoted
string.

See topic 'models' for information on specifying Model.

xvariables() (without a model) does the same except it uses the model
specified in STRMODEL which is usually the model used by the most recent
GLM (generalized linear or linear model) command such as regress(),
anova(), or poisson().

xvariables(Model, missing:val) and xvariables(missing:val), where val is
a REAL scalar, provides a value for cases with missing values.

Each variate in the model will appear as a column of the output.

Factors and interactions are translated to one or more "dummy" variables
with values 1, 0, or -1, or to products of dummy variables or of dummy
variables and variates.

Any row in which the dependent variable or any factor or variate is
MISSING is set entirely to 0, or, when 'missing:val' is an argument, to
val.  For example, xvariables(Model, missing:?)  or
xvariables(missing:?)  results in rows with missing data being set to
MISSING rather than to 0.

                          Use after regress()
Note: When no model is specified and the previous command was regress(),
xvariables() does not compute dummy variables associated with any
factors in the model, but treats them as if they were variates.  Except
in this special case, the behavior of xvariables() differs sharply from
the behavior of modelvars() which retrieves factors and variates
unchanged.

                          No factors in model
When there are no factors in the model, a regression of the dependent
variable on the columns of xvariables()[,-1], that is, the result of
xvariables() excluding the constant column, should yield the same
coefficients as does coefs() after regress(Model) or anova(Model).

                         With factors in model
When there are factors in the model, the regression coefficients in such
a regression will differ from those produced by coefs() after
anova(Model), although the fitted values and residuals will be the same.
```

For example, following anova("y=a.x"), where a is a factor and x is a variate, the coefficients from coefs("a.x") will be the slopes in a model fitting separate lines for each level of a, but with a common intercept. This is not the parametrization implicit in a regression of y on the columns of xvariables[,-1].

Cross references

See also topics varnames(), modelvars(), modelinfo(), 'models'.

2.400 yates()

Usage:

```
yates(x), x a REAL vector
```

Keywords: glm, anova

Usage

yates(x) performs Yates' algorithm for the effects in a 2-series factorial experiment. The argument x should be a REAL vector (univariate case) or matrix (multivariate case) containing the 2^k observations in standard order, that is, the levels of the first factor changing most rapidly.

When x is a vector, the value is a vector of the (2^k)-1 effects in standard order and divisor 2^(k-1). For example, for a 2^3 experiment with data vector(x111,x211,x121,x221,x112,x212,x122,x222), the result is vector(A, B, AB, C, AC, BC, ABC), where

```
  A  = (-x111 + x211 - x121 + x221 - x112 + x212 - x122 + x222)/4
  B  = (-x111 - x211 + x121 + x221 - x112 - x212 + x122 + x222)/4
  AB = (+x111 - x211 - x121 + x221 + x112 - x212 - x122 + x222)/4
```

and so on. The mean, (x111+x211+x121+x221+x112+x212+x122+x222)/8, is not included.

In matrix notation, the k = 3 case can be expressed as

```
  [ A   ]         [ -1   1  -1   1  -1   1  -1   1 ]  [ x111 ]
  [ B   ]         [ -1  -1   1   1  -1  -1   1   1 ]  [ x211 ]
  [ AB  ]         [  1  -1  -1   1   1  -1  -1   1 ]  [ x121 ]
  [ C   ] = (1/4) [ -1  -1  -1  -1   1   1   1   1 ]  [ x221 ]
  [ AC  ]         [  1  -1   1  -1  -1   1  -1   1 ]  [ x112 ]
  [ BC  ]         [  1   1  -1  -1  -1  -1   1   1 ]  [ x212 ]
  [ ABC ]         [ -1   1   1  -1   1  -1  -1   1 ]  [ x122 ]
                                                      [ x222 ]
```

When x is a matrix with m columns, the value is a (2^k) - 1 by m matrix, each column of which is the result of applying Yates's algorithm to the corresponding column of x.

2.401 yulewalker()

Usage:

```
yulewalker(vec [, inverse:T]), vec a REAL vector or matrix.
```

Keywords: time series

```
                                    Usage
yulewalker(Rho) computes a REAL vector of autoregressive (AR)
coefficients phi[1], phi[2], ... phi[p] of a p-th order AR time series
whose autocorrelations r[1], r[2], ... r[p], are in REAL vector Rho,
where p = nrows(Rho).  The values of phi satisfy the Yule-Walker
equations rho[j] = sum(phi[k]*rho[j-k],j=1,...,p), k = 1,...,p, with
rho[0] = 1, rho[-j] = rho[j].

When Rho is a matrix, yulewalker(Rho) computes AR coefficients from each
column separately, that is, yulewalker(Rho)[,j] = yulewalker(Rho[,j]), j
= 1,...,ncols(Rho).  If Rho is a generalized matrix (at most two
dimensions >= 1), yulewalker(Rho) = yulewalker(matrix(Rho)) (see
'matrices', matrix()).

When any column of Rho is not a valid autocorrelation function, that is,
if the implied Toeplitz correlation matrix is not positive definite,
yulewalker() prints a warning message, sets the element in the result
where the violation occurred to the most extreme value possible and any
subsequent elements to zero.  For instance, yulewalker(vector(-.3,
-.9,.5)) returns the result vector(-.6, -1, 0).

                        Use in estimation of AR model
A typical usage is
  Cmd> rhohat <- autocor(y, 10) # compute 1st 10 autocorrelations

  Cmd> phihat <- yulewalker(rhohat)

phihat is a vector of length 10 containing the coefficients of the
autoregressive series whose first 10 autocorrelations are the same as
rhohat.  Thus yulewalker() provides a method-of-moments way to estimate
the parameters of an AR (autoregressive) model.

When time series y is in fact a normal stationary AR series of order
length(rhohat), these estimates are asymptotically equivalent to maximum
likelihood estimates.

NOTE: autocor() is a macro in file tser.mac.  Type help(autocor) for
details.

                               Keyword 'inverse'
yulewalker(Phi,inverse:T) computes auto correlations corresponding to
autoregressive coefficients in the columns of REAL vector or matrix Phi.

Effectively yulewalker(Phi,inverse:T) is the inverse function to
yulewalker() in that yulewalker(yulewalker(Rho),inverse:T) should the
same as Rho, except for rounding error.
```

```
One important usage is
  Cmd> rho <- yulewalker(padto(Phi,n), inverse:T)

where n >= nrows(Phi).  This computes the first n autocorrelations of
the autoregressive series with autoregression coefficients in Phi.

                            Cross references
See also padto(), partacf(), toeplitz().
```

2.402 zinterval()

Usage:

```
zinterval(x[,y],cover:fraction,var:variance,[upper:T or lower:T])
```

Keywords: probabilities, descriptive statistics, comparisons

```
zinterval() computes a z-confidence interval for a population mean
or difference of population means, depending on whether one or two
variables are given as arguments.  By default, zinterval() computes
a two-sided interval, but you may choose one-sided alternatives
by using one of lowerb:T or upperb:T.

You specify the coverage rate via cover:value. You specify the
variance via var:variance.  For a two-sample interval, you may specify
different variances by using a vector of length two.

The output is a vector containing the estimate and interval bounds.

Because the variance of the data must be known, these intervals are
rarely used in practical data analysis.
```

2.403 ztest()

Usage:

```
ztest(x[,y],null:val,var:variance,[upper:T or lower:T])
```

Keywords: probabilities, descriptive statistics, comparisons

```
ztest() performs a one- or two-sample z-test, depending on whether
one or two variables are given as arguments.  By default there is
a two-tailed alternative, but you may choose one-sided alternatives
by using one of lowertail:T or uppertail:T.

You specify the null value via null:value.  You specify the
variance via var:variance.  For a two-sample test, you may specify
```

different variances by using a vector of length two.

The output is a vector containing the z-statistic and the p-value.

Because the variance of the data must be known, these tests are
rarely used in practical data analysis.

Chapter 3

Arima Macros Help File

This Chapter contains help for a set of macros doing least squares nonlinear fitting, including fitting ARIMA models to time series by unconditional least squares, that are distributed with MacAnova in the file Arima.mac.txt. The material here is a reformatting of the help in file Arima.mac.txt.

3.1 acfarma()

Usage:

```
acfarma(phi,theta [,lag:L][,nfreq:Nfreq][,arsign:Arsign,\
  masign:Masign]), REAL vectors phi, theta, positive integer L, integer
  Nfreq != 0, Arsign and Masign either +1 or -1
```

Keywords: arima models, time domain, autocovariance

```
                                  Usage
Macro acfarma() computes the theoretical autocovariance function (ACVF)
of an ARMA time series with given coefficients and innovation variance
1.

gamma <- acfarma(phi, theta, lag:L), where phi and theta are REAL
vectors and L > 0 is an integer, returns the ACVF from 0 to L lags of
an ARMA time series with REAL coefficient vectors phi and theta.
gamma[1] is the variance and gamma[h+1] is the lag h autocovariance, h
= 1, ..., L.  The innovation variance is assumed to be 1.

To omit part of model, use phi = 0 or theta = 0.

'lag:L' can be omitted, in which case the default value is L = 100.

You can make an "impulse" plot of the first 50 lags of the ACFV by
  Cmd> tsplot(acfarma(phi, theta, lag:50),0,impulse:T,lines:F)

If you omit 'lines:F', lines are drawn between successive values.

                            Sign conventions
The model assumed for series X is
```

```
  (1 + Arsign*sum(phi*B^run(p)))X[t] =
                    (1 + Masign*sum(theta*B^run(q)))Z[t].

where Arsign and Masign are either +1 or -1 and {Z[t]} is zero mean
white noise with standard deviation 1.

The default value for Arsign is variable ARSIGN if it exists and -1
otherwise.  The default value for Masign is variable MASIGN if it exists
and -1 otherwise.  Or you can include either or both keyword phrases
'arsign:Arsign' and 'masign:Masign' as arguments.  See topic 'MASIGN'
for an explanation.  If you want the convention used by Brockwell and
Davis, you should do the following before you start your analysis

  Cmd> MASIGN <- 1; ARSIGN <- -1

It is an error for ARSIGN, MASIGN or supplied values of Arsign and
Masign to be other than +1 or -1.

                              Keyword nfreq
acfarma() uses the fast discrete Fourier transform to compute the ACVF.
The number of frequencies used is the smallest integer Nfreq >=
max(500,L+1) with no prime factors > 29.  Or you can specify Nfreq
by including nfreq:Nfreq as an argument, where Nfreq is non-zero
integer.  When Nfreq < 0, abs(Nfreq) is used without further checking;
when Nfreq > 0 it is an error when Nfreq has a prime factor > 29.

                            Cross references
See also specarma(), tsplot().
```

3.2 arima()

Usage:

```
   arima(y [,pdq:vector(p,d,q)] [,PDQ:vector(P,D,Q),seasonal:period]\
     [,x:x,fitmean:T, start:b0,active:active,cast:n, cycles:m,mle:T,\
     mlecycles:m1, masign:Masign, arsign:Arsign, maxit:itmax, minit:itmin,\
     crit:vector(numsig, nsigsq, delta), print:T,keep:T,quiet:T]),
     REAL vector y, REAL variable x, nonnegative integers p, d, q, P, D, Q,
     period > 1, cast, n, m, m1, REAL vector b0, LOGICAL vector active with
     length(active) = length(b0), Masign and Arsign +1 or -1, integers
     itmax, itmin, numsig, nsiqsq, REAL scalar delta >= 0
```

Keywords: arima models, time domain, nonlinear fitting

```
                                    Usage
arima(y [,pdq:vector(p,d,q)] [,PDQ:vector(P,D,Q),seasonal:Period]),
where y is a REAL vector with no MISSING values and p, d, q, P, D, Q
are non-negative integers and Period > 1 is an integer, uses
unconditional least squares to estimate the parameters of an
ARIMA(p,d,q)x(P,D,Q) time series model, with season length Period.

When d = D = 0, a non-zero mean is fit; otherwise no mean is fit.
```

Keyword 'seasonal' is required when keyword 'PDQ' is used.

arima(y, x:x [,pdq:vector(p,d,q)] [,PDQ:vector(P,D,Q),seasonal:Period])
does the same, except a linear regression with ARIMA errors of y on the columns of REAL matrix x is carried out. x must have no MISSING elements and nrows(x) = nrows(y). It should not include a constant column.

arima(y [, x:x], pdq:pdq, PDQ:PDQ, mle:T) does the same, except maximum likelihood estimation is used instead of unconditional least squares.

You can use keywords 'minit', 'maxit', 'cast', 'cycles', and 'mlecycles' to control certain details of the estimation algorithm. See below for more information.

You can use keywords 'masign' and 'arsign' or define variables MASIGN and ARSIGN to control the sign conventions assumed for moving average and autoregressive coefficients. See below and topic 'MASIGN' for details.

Model coefficients

The model coefficients estimated include some or all of the following:

mu	mean (of differenced data when d > 0 or D > 0)
beta	vector of coefficients of columns of x, if any
phi	vector of AR coefficients
theta	vector of MA coefficients
phiS	vector of seasonal AR coefficients
thetaS	vector of seasonal MA coefficients.

They are grouped in a coefficient vector b = vector(mu, beta, phi, theta, phiS, thetaS), omitting any coefficients not in the model.

You can provide starting values using keyword 'start'; see below.

To force inclusion of mu when d > 0 or D > 0, use 'fitmean:T' as an argument. To force exclusion of mu when d = D = 0, use 'fitmean:T'. See below.

You can omit certain coefficients from being included in the estimation algorithm using keyword 'active'; see below. This is useful when fitting models using a subset of lags. Any starting values you provide for them (non-zero starting value for mu) using 'start' are not changed but will be used in computing residuals.

When mu is in the model, but is not an active parameter, it is estimated by the sample mean of the possibly differenced data unless a non-zero starting value is given.

Output

arima() prints a table of the estimated coefficients, their approximate standard errors, t = coef/StdErr, and a nominal P-value based on the t distribution. It also prints the mean square error (MSE), its degrees of freedom (DF), -2*log(L), where L = likelihood, a modification of

```
Akaike's information critirion (AICC), and the complete sum of squared
residuals, including backcast values, if any (RSS).  Printing can be
suppressed by either quiet:T or keep:T; see below.  Side effect
variables are always produced; see below.

DF = n - d - D*Period - (number of parameters being estimated).  When mu
is estimated by a sample mean, it counts as a parameter, even if it is
not active in the optimization (active[1] is false).

                          Side effect variables
arima() creates the following side effect variables
  COEF = bhat = vector(muhat,betahat,phihat,thetahat,phiShat,thetaShat),
    omitting any coefficients not in the model.  When coefficient b[i]
    is inactive (seek keyword 'active' below), COEF[i] = the starting
    value, if any, or 0.
  ALLRESIDUALS = residuals from fitted model including incomplete and
    backcast residuals
  RESIDUALS = ALLRESIDUALS, omitting backcast and incomplete residuals
  RSS = sum(ALLRESIDUALS^2)
  NEG2LOGL = -2*log(likelihood) assuming a Gaussian series.  The likeli-
    hood includes a factor of (2*PI)^((n-d-D*Period)/2)
  NPAR = number of active parameters plus 1 for the mean if estimated
    as sample mean.  The active parameters correspond to rows k of
    XTXINV with XTXINV[k,k] != 0
  JACOBIAN = REAL matrix of derivatives of ALLRESIDUALS with respect to
   active parameters.
  XTXINV = analogue of solve(X' %*% X) matrix in regression computed
   from JACOBIAN, with rows and columns corresponding to inactive
   parameters set to 0
  HII, REAL vector of leverages of length n - d - D*Period, computed
   from JACOBIAN
  GRADIENT = REAL gradient vector (approximation to to partial
   derivatives sum of squares or log likelihood with respect to the
   coefficients) with an element for each active parameter

The innovation variances is estimated as MSE = RSS/(n-d-D*Period-NPAR).
MSE is used to compute estimated standard errors as
sqrt(MSE*diag(XTXINV))

                          Choosing starting values
When there is no seasonal part to the model and no omitted lags, you may
be able to speed convergence by finding starting values using macros
hannriss() or innovest() and providing them to arima() using keyword
'start' (see below).

                       Difference from other programs
Other ARIMA estimation programs may differ in what is computed as the
estimated innovation variance.  Two alternatives to MSE as computed by
arima() are sum^2(ALLRESIDUALS)/DF and
sum(ALLRESIDUALS^2)/(n-d-D*Period), the maximum likelihood estimate.

Programs also may differ in sign conventions used for the autoregressive
and moving average coefficients.  In arima(), the signs are determined
```

```
either by variables MASIGN and ARSIGN (defaults are -1 and -1) or the
value of keywords 'arsign' and 'masign'.  See topic 'MASIGN'.

                            Sign conventions
arima() allows you to choose among several conventions that have been
used for the signs of AR and MA coefficients.  These are specified by
numbers Arsign and Masign with values +1 or -1.  Either or both can be
set as values of keywords 'arsign' and 'masign'; see below.  If not set
by 'arsign', the default value for Arsign is the value of variable
ARSIGN if it exists, or -1 otherwise.  Similarly the default value for
Masign is MASIGN or -1.  For details, see topic 'MASIGN'.

To use the Brockwell and Davis convention, before you start your
analysis you should do the following:
  Cmd> MASIGN <- 1; ARSIGN <- -1

See topic 'MASIGN'.

                 Other Optional Keyword Phrases
  Keyword phrase               Explanation
  ------------------------------------------------------------------------
  fitmean:T or F Include mu in model when T, omit mu otherwise. Default
                 is T with d = D = 0 and F otherwise.  When d > 0 or D
                 > 0, mu is the mean of the differenced series
  active:active  LOGICAL vector with length(active) = npars = length(b)
                 where b = vector(mu, beta, phi, theta, phiS, thetaS).
                 When active[i] is F, b[i] is "inactive" and does not
                 change from its starting value.  If you want to
                 estimate mu by the sample mean rather than least
                 squares or MLE, use active[1] = F and b0[1] = 0
  start:b0       REAL vector b0 = vector(mu0, beta0, phi0, theta0,
                 phiS0, thetaS0) of starting values values for the
                 iteration.  Default is rep(0,npar).  b0 should include
                 values for "inactive" parameters.  When the mean is in
                 the model, b0[1] = 0 is equivalent to b0[1] = sample
                 mean.
  cast:nback     Residuals will be computed by "backcasting" for nback
                 >= 0 time points before the start of the series
  cycles:m       m (non-negative integer) cycles of forecasting/
                 backcasting cycles used in computing residuals; default
                 m = 1.
  masign:Masign  Alters definition of MA parameters.  Masign must be +1
                 or -1.  As an example, a MA(2) model is
                  Y(t) = Z(t) + Masign*(theta[1]*Z(t-1) +
                       theta[2]*Z(t-2))
                 instead of
                  Y(t) = Z(t) - theta[1]*Z(t-1) - theta[2]*Z(t-2)
  arsign:Arsign  Alters definition of AR parameters.  Arsign must be +1
                 or -1.  As an example, an AR(2) model is
                  Y(t) = Z(t) - Arsign*phi[1]*Y(t-1) -
                       Arsign*phi[2]*Y(t-2)
                 instead of
                  Y(t) = Z(t) + phi[1]*Y(t-1) + phi[2]*Y(t-2)
```

```
  maxit:itmax    Maximum number of iterations allowed (default 30, 0 is
                 ok).  With mle:T, when itmax < 0, no iterations of
                 least squares fitting is done and up to abs(itmax)
                 iterations of MLE using sigmahat based on coefficients
                 in b0.  When itmax = 0, no iteration is done at all,
                 but calculations are done using starting values.
  minit:itmin    Minimum number of iterations carried out (default = 0)
  crit:vector(numsig, nsigsq, delta)
                 Integer numsig >= 1 is the number of accurate digits
                 wanted in coefficients.
                 Integer nsiqsq >= 1 is the number of accurate digits
                 wanted in the residual sum of squares.
                 When delta > 0, iterations stop when the norm of the
                 gradient <= delta
                 Iteration terminates the first time any of these
                 criteria is met.  When any of numsig, nsigsq or delta
                 are 0, that criterion is ignored.
                 Example: crit:vector(6,0,0) specifies iteration
                 continues until the relative change in all
                 coefficients is less than 1e-6.
  print:T        Partial results are printed at each iteration
  keep:T         arima() returns a structure as value (see below).
  quiet:F        No summary results are printed, although side effect
                 variables are created.  Default is F except with keep:T

                            Value returned
Without keyword phrase 'keep:T', arima() returns NULL as value.

With keyword phrase 'keep:T', arima() returns a structure with the
following components:
  Name           Contents
  coefs          vector(muhat,betahat,phihat,thetahat,phiShat,thetaShat)
                 with coefficients not in the model omitted.
  hessian        NPAR by NPAR matrix JACOBIAN %c% JACOBIAN
  gradient       Same as GRADIANT
  residuals      Same as ALLRESIDUALS
  nobs           length(y) - d - D*Period
  npar           number of active parameters
  pdq            zero padded value of keyword 'pdq' or rep(0,3)
  PDQ            zero padded value of keyword 'PDQ' or rep(0,3)
  seasonal       value of keyword 'seasonal' or 0
  active         logical vector the same length as coefs; active[i] is
                 True if and only if coefs[i] is an active parameter
  iter           Number of iterations to convergence or termination
  iconv          0: not converged by any criterion
                 1: converged by relative coefficient change criterion
                 2: converged by relative RSS or log L change criterion
                 3: changed by norm of gradient criterion
                 4: could not further reduce RSS or -2logL
  rss            Same as RSS
  neg2logL       Same as NEG2LOGL
  aicc           Same as AICC
```

```
                          Cross references
See also hannriss(), innovest().
```

3.3 arimahelp()

Usage:

```
arimahelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
arimahelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
  Subtopics
arimahelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
arimahelp(key:Key), CHARACTER scalar Key
arimahelp(index:T [,scrollback:T])
```

Keywords: general

```
                                Usage
arimahelp(Topic1 [, Topic2, ...]) prints help on topics Topic1, Topic2,
... related to macros in file arima.mac.  The help is taken from file
arima.mac.

arimahelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these macros.

arimahelp(index:T) or simply arimahelp() prints an index of the topics
available using arimahelp().

arimahelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER
scalar or vector, prints subtopics of topic Topic.  With subtopic:"?", a
list of subtopics is printed.

arimahelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where Suptopics1
and Subtopics2 are CHARACTER scalars or vectors, prints the specified
subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to arimahelp().  In windowed
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

arimahelp(key:key) where key is a quoted string or CHARACTER scalar
lists all topics cross referenced under Key.  arimahelp(key:"?") prints
a list of available cross reference keys for topics in the file.

arimahelp() is implemented as a predefined macro.

                           Cross reference
See help() for information on direct use of help() to retrieve
information from arima.mac.
```

3.4 arimares()

Usage:

```
residuals <- arimares(b,x,y,params), REAL vectors b, y, REAL
  vector or matrix x, params = structure(pdq:vector(p,d,q),
  PDQ:vector(P,D,Q),seasonal:n1,cast:n2,cycles:n3,fitmean:T or F
  [,sigmahat:s]) integers >= 0 p, d, q, P, D, Q, n1, n2, n3,
  integer seasonal > 0, REAL scalar sigmahat
```

Keywords: arima models, time domain, nonlinear fitting

```
                                Usage
residuals <- arimares(b,x,y,params) returns a REAL vector of residual
from a (p,d,q)x(P,D,Q) ARIMA model with specified coefficients.  The
model may optionally have one or more linear predictors.  Depending on
arguments it can do one or more forecast/backcast cycles to estimate
past residuals.  It is intended for use by other macros.

b = vector(mu,beta,phi,theta,phiS,thetaS) is a REAL vector of
coefficients.  mu is the mean, beta is a vector of coefficients of
linear predictors in the columns of x, phi, theta, phiS and thetaS are
vectors of AR, MA, seasonal AR and seasonal MA coefficients,
respectively.  Any of mu, beta, ..., thetaS are NULL if they are not in
the model.

x is either 0 (no linear predictors) or a nrows(y) by k REAL matrix of
linear predictors.

REAL vector y contains the time series being analyzed.

                            Argument params
params = structure(pdq:vector(p,d,q),PDQ:vector(P,D,Q),seasonal:n1,
cast:n2,cycles:m3,fitmean:T or F [,sigmahat:s]) defines the form of the
model and details about the algorithm used.

                            Value returned
The value is as follows
  residual vector                              sigmahat is not provided
  vector(sigmahat*sqrt(log(det)),residuals)    sigmahat > 0
  sqrt(log(det))                               sigmahat < 0

                              ARIMA model
The ARIMA model is Phi(B)*PhiS(B)*((1 - B)^d*(1 - B)^D*Y[t] - mu) =
mu + Theta(B)*ThetaS(B)*Z(t), where B is the backshift
operator.  Here
  Phi(z) = 1 - phi[1]*z - phi[2]*z^2 - ... - phi[p]*z^p.
  Theta(z) = 1 - theta[1]*z - theta[2]*z^2 - ... - theta[q]*z^q
  PhiS(z) = 1 - phiS[1]*z^n1 - phiS[2]*z^(2*n1) - ... - phi[P]*z^(P*n1)
  ThetaS(z) = 1 - thetaS[1]*z^n1 - thetaS[2]*z^(2*n1) - ... -
     theta[Q]*z^(Q*n1)

When there are k linear predictors, there is also an term X %*% beta on
the right, where X is an nrows(y) by k matrix matrix of linear
predictors with coefficient vector beta
```

```
When another convention on the signs of the MA and/or AR coefficients is
wanted, the calling macro must make the adjustment.  That is, in the
notation used in describing arima, arimares() assumes Arsign = -1,
Masign = -1.

                      Details about argument params
The components of params are as follows:
 p,d,q   integers >= 0, AR order, difference, MA order
 P,D,Q   integers >= 0, seasonal AR order, seasonal difference,
         seasonal MA order
 n1      integer > 1, length of seasonal
 n2      integer >= - specifies how far to backcast
 n3      integer >= 0 number of cycles of fore/backcasting
 fitmean:T => mean will be fit
 s       REAL scalar, estimate of residual stddev.  Its presence
         signals that log(det(covariance matrix)) is to be
         computed. If sigmahat < 0, this is all that arimares()
         does, returning the scalar sqrt(log(det)).
length(b) should be p + q + (P + Q)*seasonal + 1*fitmean +
         ncols(x)*isscalar(x)
```

3.5 ARSIGN

Keywords: arima models

```
                                Description
Various macros, including arima(), hannriss(), innovest(),
neg2logLarma(), acfarma() and specarma(), allow for different sign
conventions in the definition of autoregression and moving average
coefficients.  This is controlled by keywords 'arsign' and 'masign'
and/or scalar variables ARSIGN and MASIGN, all with values -1 or +1.

                              Cross reference
See topic 'MASIGN' for details.
```

3.6 detarma()

Usage:

```
   detarma(phi, theta, n [,masign:Masign] [,arsign:Arsign] [,nfreq:Nfreq]),
     non-MISSING REAL vectors phi and theta, positive integer vector n,
     Masign and Arsign +1 or -1, positive integer scalar Nfreq
```

Keywords: arima models

```
                              Introduction
Help is not complete but is taken from the comments associated with
detarma().
```

```
detarma() computes the determinant of the Toeplitz covariance associated
with ARMA process.  At present it has no provision for seasonal
processes.

                                  Usage
 detarma(phi, theta, n [,masign:Masign] [,arsign:Arsign] [,nfreq:Nfreq])
  phi      non-MISSING REAL vector defining a causal AR operator
  theta    non-MISSING REAL vector defining a MA operator
  n        vector of positive integers
  Masign and/or Arsign must be +1 or -1; they determine sign conventions
           for interpreting phi and theta.  Type arimahelp(MASIGN)
           for information.
  Nfreq    Number of frequencies used in DFT to compute ACFV.  Default
           is the smallest integer >= 2*max(max(n), 200) that has no
           prime factors > 29

                              Value returned
The result is a REAL vector d, the same length as n, with d[i] =
det(Sigma(n[i])), where Sigma(n[i]) = Cov[(X(1),...,X(n[i]))] when X(t)
is ARMA with AR coefficients phi and MA coefficients theta

detarma() computes the autocovariance function as the inverse Fourier
transform of the spectrum computed at nfreq frequencies.  From this it
computes that partial autocorrelations which are used, together with the
variance to compute the determinant.
```

3.7 hannriss()

Usage:

```
hannriss(x, pdq:vector(p,d,q) [,degree:d1] [,maxlag:m] \
   [,polish:T, cycles:nc] [,arsign:Arsign] [,masign:Masign]), integers
   p >= 0, d >= 0 q >= 0, m >= p + q, nc >= 0, d1, Arsign and Masign +1
   or -1
```

Keywords: arima models, time domain, preliminary estimation

```
                                  Usage
hannriss(x,pdq:vector(p,d,q)), where p >= 0, d >= 0 and q >= 0 are
integers computes preliminary estimates of ARIMA coefficients using the
Hannan-Rissanen method.

The first step computes yulewalker estimates based on m autocorrelations
unless q = 0 when only p autocorrelations are used.  The default for m
is 20 + p + q.

Its value is structure(phi:phihat,theta:thetahat,xtxinv,rss:rss,nobs:n)

                        Keywords maxlag and degree
hannriss(x,pdq:vector(p,d,q),maxlag:m), integer m >= p + q, does the
same using m autocorrelations.
```

```
hannriss(x,pdq:vector(p,d,q),degree:d1 [,maxlag:m]) does the same,
except the possibly differenced data is detrended with a polynomial of
order d1.  d1 < 0 means nothing is subtracted.  The default for d1 is 0
when d = 0 and -1 when d > 0.

                          Side effect variables
In addition to returning a value, hannriss() creates the following side
effect variables
  COEF = vector(phihat,thetahat)
  XTXINV = analogue of solve(X' %c% X) matrix in regression
  ALLRESIDUALS = residuals from fitted model including backcast
   residuals
  RSS = sum(ALLRESIDUALS^2)
  NEG2LOGL = -2*log(likelihood)
  NPAR = p + q + degree1 + 1 = number of coefficients estimated
    where degree1 = max(d1,-1).

                            Sign conventions
phi is defined so the autoregressive operator is Phi(B) = 1 +
Arsign*phi[1]*B + Arsign*phi[2]*B^2 + ... + Arsign*phi[p]*B^p.

theta is defined so that the moving average operator is Theta(B) = 1 +
Masign*theta[1]*B + Masign*theta[2]*B^2 + ... + Masign*theta[q]*B^q,
where B is the backshift operator.

The default values for Arsign and Masign are -1 and -1, but you may
change them by keyword phrases 'arsign:Arsign' and 'masign:Masign' or by
creating variables ARSIGN and/or MASIGN with values +1 or -1.  See topic
'MASIGNS' for details.

                       Keywords polish and cycles
You can also include the following keyword phrases as arguments
  polish:T      carry out one or more extra "polishing" steps that
                should move the estimates closer to the unconditional
                least squares estimates.
  cycles:nc   nc > 0 polishing cycles will be carried out; default is 1

                                Limitation
There is currently no provision for seasonal ARIMAs
```

3.8 innovations()

Usage:

```
   innovations(gamma [,lag:m] [,final:T]), REAL vector gamma, integer m >
     0
```

Keywords: arima models, time domain, preliminary estimation

```
                               Introduction
innovations() computes the "innovation" algorithm given on p. 71 of
```

```
Brockwell, and Davis, computing a M by M matrix containing
coefficients and prediction variances.  It actually uses Cholesky
decomposition rather than the algorithm as given in Brockwell and Davis.

                                    Usage
innovations(gamma [,lag:M] [,final:T]), returns structure(theta:Theta,
v:V) where Theta is a M by M REAL matrix and V is a vector of length
M+1.  gamma is either a REAL n by n covariance matrix or a REAL vector
of length n containing an autocovariance function (ACVF).  M < n is a
positive integer with default value n - 1.

When gamma is an ACFV, innovations(gamma [,lag:M]) is the same as
innovations(toeplitz(gamma) [,lag:M])

The result is not affected by variables MASIGN if it exists.

                                  Definition
Suppose {X[1], X[2], ..., X[M+1]} is a sequence of random variables
with covariance matrix gamma (toeplitz(gamma) in ACVF case).

Then Theta is the M by M matrix with the property that

    Theta[j,j]*Z[j] + Theta[j,j-1]*Z[j-1] + ... + Theta[j,1]*Z[1]

is the best one step predictor of X[j+1] based on prediction
errors Z[1] = X[1], Z[2] = X[2] - Theta[1,1]*X[1], ..., Z[j] =
Theta[j-1,j-1]*Z[j-1]+Theta[j-1,j-2]*Z[j-2]+...+Theta[j-1,1]*Z[1].

Note that Z[j] could be expressed as a linear combination of X[j],
X[j-1], ..., X[1] so that the prediction is also the best prediction of
X[j+1] as a linear combination of X[j], X[j-1], ..., X[1].

V is a length M+1 vector with V[1] = gamma[1,1] = Var[X[1]] and V[j+1]
= prediction error variance when X[j+1] is predicted using X[1], X[2],
..., X[j] (or Z[1], ..., Z[j]).

                                Keyword final
When final:T is an argument, the result is
   structure(theta:vector(Theta[m,]), v:V[m+1])

                               Cross reference
See also innovest().
```

3.9 innovest()

Usage:

```
innovest(x,pdq:vector(p,d,q) [,maxlag:m] [,degree:degree]\
  [,polish:T,cycles:nc] [,checkroots:F] [,silent:T] [,arsign:Arsign]\
  [,masign:Masign]), integers p>=0, d >= 0, q >= 0, maxlag >= p+q,
  degree, nc >= 1, Arsign and Masign = +1 or -1.
Value is structure(phi:phihat, theta:thetahat, xtxinv, nobs:n, npar,
  rss:residSS, neg2logL:-2*log(likelihood), aicc:AICC)
```

Keywords: arima models, time domain, preliminary estimation

```
                              Introduction
innovest() is a macro to compute the innovations preliminary estimate of
coefficients for an ARIMA(p,d,q) time series as described on pp 151-153
of Brockwell and Davis.  Keywords 'arsign' and 'masign' allow you to
specify the sign convention to be used in defining parameters.  See
below.

                                   Usage
innovest(x, pdq:vector(p,d,q) [,maxlag:M]), where x is a REAL vector
and p, d and q are nonnegative integers return a structure summarizing
the results of the preliminary innovations estimates for an
ARIMA(p,d,q) model fit to x.  M >= p + q is an integer with default
value p + q + max(15, p + q).  See below for the form of the results.

There is currently no provision for estimating seasonal ARIMA models

                           Side effect variables
innovest() creates the following side effect variables
  COEF = vector(phihat,thetahat), the estimated AR and MA coefficients
  ALLRESIDUALS = residuals from fitted model including backcast
    residuals
  RSS = sum(ALLRESIDUALS^2)
  NEG2LOGL = -2*log(likelihood) using the estimates found
  NPAR = p + q + degree + 1 = number of coefficients estimated

                     Optional keyword phrase arguments
There are several optional keyword phrases which affect what innovest()
does:
  degree:d1      A polynomial trend of order d1 is removed (after
                 differencing when d > 0).  Nothing is removed, not
                  even a mean, when d1 < 0.  The default for d1 is
                  -d (mean removed when d = 0, nothing otherwise).
  polish:T       The estimates are adjusted by one or more cylcels
                 of an approximate iteration in the direction of the
                  least squares estimates.
  cycles:nc      nc > 0 an integer, default 1 is the number of
                 "polishing" cycles.
  checkroots:F   suppresses checking for stationarity and invertability
  silent:T       suppress warning messages
  arsign:Arsign  +1 or -1; alters definition of AR paramaters; see below
  masign:Masign  +1 or -1; alters definition of MA paramaters; see below
```

```
                          Value returned
innovest() returns as value
  structure(phi:phihat, theta:thetahat, nobs:n, xtxinv:xtxinv,
    npar:p+q+d1+1, rss:residSS, neg2logL:-2*log(likelhihood),aicc:aicc)

phihat and thetahat are the estimated AR and MA coefficients (NULL when
p or q is 0).

xtxinv is NULL without polish:T or when p = q = 0, Otherwise xtxinv is
an analogue of the regression solve(X' %*% X) derived from the final
polishing step.  Its diagonal elements can be used to compute
approximate standard errors of the coefficients.

residSS = sum of squares of all residuals, including those backcast.

The likelihood is the normal likelihood computed using backcasting and
includes a factor of (2*PI)^(-n/2) and aicc is a modification of
Akaike's information criteria (AIC).

Note that innovest() does not return a mean or other estimates of
detrending parameters.

                          Check on operators
By default, estimated AR and MA coefficients are checked to see if they
define stationary (causal) and invertable operators, respectively.  If
they do not, a warning message printed and any "polishing" cycles (see
below) are suppressed and components rss, neg2logL and aicc of the
result are set to MISSING.

                          Sign conventions
See topic 'MASIGN' for information on how Arsign (default -1 or the
value of ARSIGN if it exists) and Masign (default -1 or the value of
MASIGN if it exists) modify the definitions of the AR and MA parameters.

To use the convention used by Brockwell and Davis before you start the
analysis, you should use
  Cmd> MASIGN <- 1; ARSIGN <- -1

The convention used by Box and Jenkins (the default) corresponds to
Arsign = -1, Masign = -1.

                          Cross references
See also arima(), hannriss(), innovations().
```

3.10 MASIGN

Keywords: arima models

```
                          Sign conventions
Various macros, including arima(), hannriss(), innovest(),
neg2logLarma(), acfarma() and specarma(), allow for different sign
```

```
conventions in the definition of autoregression and moving average
coefficients.  The convention to be used may be determined by keyword
phrases 'arsign:Arsign' and 'masign:Masign' or the values of variables
ARSIGN or MASIGN, when they exist.

Some functions and macros, including movavg(), autoreg(), polyroot(),
and moveoutroots() *always* use a sign convention equivalent to Arsign
= Masign = -1 and are *not* affected by keywords 'arsign' and 'masign'
or the values of ARSIGN and MASIGN.

For the the affected macros, an ARMA model is assumed to have the form
 X[t] + Arsign*(phi[1]*X[t-1]+phi[2]*X[t-2]+...+phi[p]*X[t]) =
     Z[t] + Masign*(theta[1]*Z[t-1]+theta[2]*Z[t-2]+...+theta[q]*Z[t-q])
where {Z[t]} is white noise.

Arsign is -1 or +1.  Without 'arsign:Arsign', the default value of
Arsign is the value of variable ARSIGN when it exists or -1 when it
does not.

Masign is -1 or +1.  Without 'masign:Masign', the default value of
Masign is the value of variable MASIGN when it exists or -1 when it
does not.

                      Examples of sign conventions
Examples for an ARMA(2,2) model with zero mean and AR coefficients phi
and MA coefficients theta.

  For the defaults arsign:-1 and masign:-1, the model is
    Y[t] = phi[1]*Y[t-1] + phi[2]*Y[t-1] +
                Z[t] - theta[1]*Z[t-1] - theta[2]*Z[t-2]
  With arsign:-1 and masign:1, the model is
    Y[t] = phi[1]*Y[t-1] + phi[2]*Y[t-1] +
                Z[t] + theta[1]*Z[t-1] + theta[2]*Z[t-2]
  With arsign:1 and masign:-1, the model is
    Y[t] = -phi[1]*Y[t-1] - phi[2]*Y[t-1] +
                Z[t] - theta[1]*Z[t-1] - theta[2]*Z[t-2]
  With arsign:1 and masign:1, the model is
    Y[t] = -phi[1]*Y[t-1] - phi[2]*Y[t-1] +
                Z[t] + theta[1]*Z[t-1] + theta[2]*Z[t-2]

                                 Defaults
The convention used by Box and Jenkins and many others corresponds to
Arsign = -1 and Masign = -1.  This is the convention that is implicit
in the definition of MacAnova functions autoreg(), movavg() and
polyroot().  These functions do not recognize 'arsign' and 'masign' and
are not affected by the values of ARSIGN and MASIGN.

The convention used by Brockwell and Davis corresponds to Arsign = -1
and Masign = +1.  If you want results consistent with this convention
you should do the following before you start your work:
  Cmd> MASIGN <- 1; ARSIGN <- -1

This is easier than using 'masign:1' whenever you are using one of the
```

```
affected macros.

                Macros not affected by masign and arsign
Note that macros innovations() and arimares() are not affected by this
notation, but macros that use these compensate.

                             Cross references
See also autoreg(), movavg(), polyroot(), acfarma(), arima(),
arimares(), hannriss(), innovest(), neg2logLarma(), specarma().
```

3.11 moveoutroots()

Usage:

```
moveoutroots(theta), REAL vector theta
```

Keywords: time domain, arima models

```
                                    Usage
moveoutroots(theta) where theta is a REAL vector defining the
polynomial P(z) = 1 - sum(theta*zˆrun(n)), n = length(theta), returns a
new vector theta1 defining a new polynomial P1(z) of the same form which
has the same zeros as P(z) except that any zero z0 with |z0| < 1 is
replaced by 1/z0.

Thus all the zeros of P1(z) are outside the unit circle in the complex
plane and moveoutroots(theta) are the coefficients of an invertible MA
operator and, when no zero has modulus 1, the coefficients of a
stationary (causal) AR operator.

Equivalently, if Q(z) = zˆn - sum(theta*zˆrun(n-1,0)) and Q1(z) =
zˆn - sum(theta1*zˆrun(n-1,0), any zero z0 of Q(z) with |z0| > 1 is
replaced by 1/z0.

                                Sign convention
moveoutroots assumes the sign convention used by Box and Jenkins and
implicit in movavg(), autoreg() and polyroot() (Arsign = -1, Masign =
-1).  When theta are MA coefficients using the sign convention of
Brockwell and Davis (Masign = +1), you should use moveoutroots(-theta).
See topic 'MASIGN'.

                                   Examples
Examples:
  Cmd> moveoutroots(2)
  (1)         0.5

This is correct because 1 - 2*z has zero .5 < 1 and 1 - .5*z has zero 2
> 1.

  Cmd> moveoutroots(vector(2,-1.25))
  (1)         1.6         -0.8
```

```
This is correct because 1 - 2*z + 1.25*z^2 has zeros z = .8 +- .4*i with
|z| = sqrt(.8) = 0.89443 < 1 and 1 - 1.6*z + .8*z^2) has roots z =
(1+-.5*i) = 1/(.8 -+ .4*i) with |z| = sqrt(1.25) = 1.118 > 1.

                            Cross references
See polyroot(), movavg(), autoreg().
```

3.12 neg2logLarma()

Usage:

```
   neg2logLarma(y,coefficients, [,x:x] [,pdq:vector(p,d,q)] \
     [,PDQ:vector(P,D,Q),seasonal:s] [,fitmean:T or F] \
     [,cast:m] [,cycles:ncyc] [,sigmasq:sigmasq] [,neg2logL:F],
     [,residuals:T] [,logdet:T] [,sigmahatsq:T] [,all:T]
     [,masign:1 or -1] [,arsign:1 or -1])
     nonMISSING REAL vector y, optional nonMISSING REAL matrix x,
     integers p, d, q, P, D, Q, m, ncyc, >= 0, integer s > 0,
     sigmasq REAL scalar > 0 or MISSING
```

Keywords: arima models, time domain

```
                              Introduction
Macro to compute -2*log(L) and other quantities for an ARIMA model with
specified parameters.  L is the likelihood or concentrated likelihood.

The calling sequence is very similar to that of arima(), except you must
provide values of the coefficients and there are keywords which specify
what is returned.

                            Usage and arguments
 neg2logLarma(y,b, [,x:x], [pdq:vector(p,d,q)] \
  [,PDQ:vector(P,D,Q),seasonal:s] [,fitmean:T or F] \
  [, sigmasq:sigmasq] [,cast:m] [cycles:ncyc] [,neg2logL:F],
  [,masign:Masign] [,arsign:Arsign]\
  [,residuals:T] [,logdet:T] [,sigmahatsq:T] [,all:T])

y is a real nonmissing vector of length n (the response)
b = vector(mu, beta, phi, theta, phiS, thetaS) of coefficients (mu =
   mean of mean difference, beta = slopes of linear predictors, phi =
   non-seasonal AR coefficients, theta = non-seasonal MA coefficients,
   phiS = seasonal AR coefficients, thetaS = non-seasonal MA
   coefficients).  Any parts that are not needed are NULL or omitted.
x = optional non-MISSING REAL matrix of linear predictors, with
   nrows(x) = nrows(y)
p, d, q >= 0 are integers specifying the non-seasonal ARIMA component
P, D, Q >= 0 are integers specifying the seasonal ARIMA component
s > 1 an integer specifying the season length; required with any of P,
  D or Q non-zero
fitmean:T means mu is in the model; fitmean:F means mu is not in
  model.  The defualt is T when d = D = 0 and F otherwise
```

```
sigmasq a REAL scalar, either MISSING or positive.  If MISSING (the
  default) the innovation variance is estimated and the "concentrated"
  log likelihood is computed.  Otherwise, the log(L) is computed
  assuming innovation variance sigmasq
m >= 0 and ncyc >= 0 are integers controlling backcasting in computing
  residuals.  See arima() and arimares() for more information.
Arsign and Masign must be +1 or -1 and control what sign convention is
  used for coefficients.  See topics 'MASIGN' and 'ARSIGN' for details.

                              Value returned
What is returned is controlled by keywords 'neg2logL', 'residuals',
'logdet', 'sigmahatsq' and 'all'.

  Keyword                Value returned
 neg2logL:T    -2*log(L).  When sigmasq is MISSING, L is the
                           concentrated likelihood
 residuals:T   vector of residuals, including backcast ones
 logdet:T      log(det(Gamma)) where sigmasq*Gamma is covariance matrix
 sigmahatsq:T  The estimated innovation variance when sigmasq is MISSING
               or sigmasq, otherwise
 all:T         Same neg2logL:T,residuals:T,logdet:T,sigmahatsq:T

If more than items is to be returned, the output is a structure with
the keywords as component names; if only one is T, the output is a
scalar or vector.  With all:T you can suppress a result by, say,
'residuals:F',

                                 Examples
To get the concentrated likelihood for an ordinary ARMA(p,q) model, use
  Cmd> result <- neg2logLarma(y,vector(mu,phi,theta), pdq:vector(p,0,q))

For an ARIMA model with d > 0 and zero mean differences, use
  Cmd> result <- neg2logLarma(y,vector(phi,theta), pdq:vector(p,d,q))

For a seasonal (p,q)x(P,Q) seasonal ARMA with no differences and, say
quarterly data, use
  Cmd> result <- neg2logLarma(y,vector(mu,phi,theta),\
    pdq:vector(p,0,q),PDQ:vector(P,0,Q),season:4)

For a seasonal (p,d,q)x(P,D,Q) seasonal ARIMA with no differences and,
say quarterly data, use
  Cmd> result <- neg2logLarma(y,vector(phi,theta),\
    pdq:vector(p,d,q),PDQ:vector(P,D,Q),season:4)

For an ARMA(p,q) model with mean that is cubic in time, try
  Cmd> predictors <- (run(n) - (n+1)/2)^run(0,3)'

  Cmd> result <- neg2logLarma(y,vector(beta,phi,theta),x:predictors,\
      pdq:vector(p,0,q), fit meanF) # beta a vector of length 4

                              Cross reference
Use arimahelp() to also see topic arima().
```

3.13 rhatcovar()

Usage:

```
rhatcovar(rho,i, j) or rhatcovar(rho,lag:L), REAL vector rho, integers
  i > 0, j > 0, L > 0
```

Keywords: time domain, autocorrelation

```
                                Usage
rhatcovar(rho [,lag:L]), where rho is a REAL vector of auto-
correlations, computes n*Var[rhohat], where Var[rhohat] is the large
sample covariance matrix of the sample autocorrelation function
rhohat, computed using Bartletts' formula.

The result is a REAL L by L matrix.  The default value of L is
(maximum lag for rho)/2.

When rho[1] == 1, rho[k] is assumed to contain the lag k-1 auto-
correlation; otherwise, rho[k] is assumed to contain the lag k
autocorrelation.  Thus if gamma is a REAL vector containing the
autocovariance function with gamma[1] = Var[X(t)], both
rhatvar(gamma/gamma[1]) and rhatvar(gamma[-1]/gamma[1]) return the same
result.

rhatcovar(rho,i,j) returns n*covar{rhohat(i),rhohat(j)}, that is,
rhatcovar(rho)[i,j].

                            Cross reference
See also rhatvar().
```

3.14 rhatvar()

Usage:

```
rhatvar(rho [,lag:L]), REAL vector of correlations rho, integer L > 0
```

Keywords: time domain, autocorrelation

```
                                Usage
rhatvar(rho [,lag:L]), where rho is a REAL vector of autocorrelations
computes the vector of n*(var(rhohat(1)),var(rhohat(2)),...,
var(rhohat(lag)) using Bartlett's formula, where rhohat are sample
autocorrelations from a series with ACF rho.

When rho[1] == 1, rho[k] is assumed to contain the lag k-1 auto-
correlation; otherwise, rho[k] is assumed to contain the lag k
autocorrelation.  Thus if gamma is a REAL vector containing the
autocovariance function with gamma[1] = Var[X(t)], both
rhatvar(gamma/gamma[1]) and rhatvar(gamma[-1]/gamma[1]) return the same
result.

The default value of L is (maximum lag for rho)/2
```

```
                              Cross reference
See also rhatcovar().
```

3.15 specarma()

Usage:

```
   specarma(phi ,theta [, nfreq:nf]), REAL vectors phi and theta, integer
     nf > 0
```

Keywords: frequency domain, time series, spectrum analysis

```
                                    Usage
specarma(phi, theta [, nfreq:Nfreq]) computes the spectrum of an ARMA
time series with AR coefficients in phi and MA coefficients in theta and
innovation variance 1.

phi and theta are REAL vectors with no MISSING values. To omit part of
the model, use phi = 0 or theta = 0

Nfreq must be a positive integer with no prime factors > 29.

The result is a vector of length nf containing the spectrum at
frequencies 0, 1/Nfreq, 2/Nfreq, ..., (Nfreq-1)/Nfreq cycles per unit
time.

Without nfreq:Nfreq, when no positive integer scalar S exists, the
default is Nfreq = 400.  When S does exist and is a positive integer
scalar, the default for Nfreq is S.  It is an error if S has a prime
factor > 29.

                               Sign conventions
You can use keyword phrases arsign:Arsign and masign:Masign, where
Arsign and Masign are +1 or -1, to modify the interpretation of the
coefficients in the ARMA model.  The default for Arsign is variable
ARSIGN if it exists or -1 if not.  The default for Masign is variable
MASIGN if it exists or -1 if not.

The ARMA model assumed is defined as
  (1+Arsign*sum(phi*B^run(p)))X[t] = (1+Masign*sum(theta*B^run(q)))Z[t].
where B is the backshift operator and {Z[t]} is white noise with mean 0
and variance 1.

Arsign = -1 and Masign = -1 correspond to the convention used by Box
and Jenkins, and Arsign = -1 and Masign = +1 correspond to the
convention used by Brockwell and Davis

                                    Example
You can plot the spectrum for the ARMA model fit by arima() by
  Cmd> result <- arima(y,pdq:vector(p,0,q), keep:T)
```

```
  Cmd> ffplot(specarma(result$phi, result$theta))

                              Cross references
See also arima(), acfarma(), ffplot().
```

Chapter 4

Design Macros Help File

This Chapter contains help for the set of design and analysis of experiments macros that are distributed with MacAnova in the file Design.mac.txt. The material here is a reformatting of file Design.hlp.txt.

4.1 aberration2()

Usage:

```
aberration2(basis), REAL matrix basis of alias generators
```

Keywords: aliasing, design, factorial

```
aberration2(basis) computes the aberration for the fractioned
two series design indicated by the basis matrix.  The basis matrix
should have elements 0 and 1 (or -1).  The aberration is
returned as a vector of counts giving the number of aliases
of length j, for j up to the number of columns in basis.
```

4.2 aliases2()

Usage:

```
aliases2(basis [,effect:vec] [,length:j]), REAL matrix basis of alias
  generators, REAL vector vec of 0's and 1's, positive integer j
```

Keywords: aliasing, design, factorial

```
                                  Usage
aliases2(basis) finds all aliases of I in a 2^(k-p) fractional factorial
and returns a CHARACTER vector of these aliases as its value.  k must be
no larger than 25, and factors are labeled A, B, C, ..., H, J, ... Z
(skipping I).

The p x k matrix basis contains the generators for the aliasing, one row
for each generator and one column for each factor in the design.  The
```

```
elements in basis are 0, -1, or 1.  A nonzero entry indicates that a
factor is present in the generator for that row. The sign of a generator
is the product of the signs of the nonzero elements of the generator.
For example, 1 0 1 0 0 -1 means -ACF is a generator (alias of I).

If basis is a (column) vector, it is changed to a row vector (1 by k
matrix) before proceding.

aliases2(basis,effect:vec) returns the aliases of vec, a vector of k 0s
and 1s that specifies an effect.

aliases2(basis [,effect:vec],length:j) does the same but returns only
aliases of length j.

Examples:
 Cmd> print(format:"2.0f",b) # Matrix b is 2x5, so 2^(5-2) design
 b:
 (1,1)  1  1  1  0  0       [ABC is a generator]
 (2,1)  0  0  1  1 -1       [-CDE is a generator]

 Cmd> aliases2(b)   # aliases of I
 (1) "I"
 (2) "ABC"
 (3) "-CDE"
 (4) "-ABDE"

 Cmd> aliases2(b,length:3)  # length 3 aliases of I
 (1) "I"
 (2) "ABC"
 (3) "-CDE"

 Cmd> aliases2(b,effect:vector(1,1,0,0,0)) # aliases of AB
 (1) "AB"
 (2) "C"
 (3) "-ABCDE"
 (4) "-DE"

                            Cross references
See also aliases3(), confound2(), choosedef2() and choosegen2().
```

4.3 aliases3()

Usage:

```
aliases3(basis[,effect:vec][,length:j]), REAL matrix basis of alias
  generators, REAL vector vec of 0's, 1's and 2's, positive integer j
```

Keywords: aliasing, design, factorial

```
                                  Usage
aliases3(basis) finds all aliases of I in a 3^(k-p) factional factorial
design and returns a CHARACTER vector of these aliases as its values.  k
```

```
must be no larger than 25 and factors are labeled A, B, ..., H, J, ...,
Z (skipping I).

basis is p x k REAL matrix of 0s, 1s and 2s which contains the
generators for the aliasing, one row for each generator and one column
for each factor in the design.  For example 1 0 2 0 0 1 means AC^2F is a
generator (alias of I).

If basis is a (column) vector, it is changed to a row vector (1 by k
matrix) before proceding.

                        Keywords effect and length
aliases3(basis,effect:vec) returns the aliases of vec, a vector of k 0s,
1s and 2s representing an effect.  When vec:rep(0,k), the result is the
same as aliases3(basis).

aliases3(basis [,effect:vec] , length:j) returns only aliases of length
j.

Examples:
  Cmd> print(c,format:"2.0f") # Matrix c is 2x4, so 3^(4-2)
  c:
  (1,1)  1  2  0  2        [A B^2 D^2 is a generator]
  (2,1)  0  1  2  2        [B C^2 D^2 is a generator]

  Cmd> aliases3(c)  # all aliases of I
  (1) "I"
  (2) "A^1 B^2 D^2 "
  (3) "A^1 B^2 D^2 "
  (4) "B^1 C^2 D^2 "
  (5) "A^1 C^2 D^1 "
  (6) "A^1 B^1 C^1 "
  (7) "B^1 C^2 D^2 "
  (8) "A^1 B^1 C^1 "
  (9) "A^1 C^2 D^1 "

  Cmd> aliases3(c,effect:vector(1,1,0,0)) # aliases of A^1B^1
  (1) "A^1 B^1 "
  (2) "A^1 D^1 "
  (3) "B^1 D^2 "
  (4) "A^1 B^2 C^2 D^2 "
  (5) "A^1 B^2 C^1 D^2 "
  (6) "C^1 "
  (7) "A^1 C^1 D^1 "
  (8) "A^1 B^1 C^2 "
  (9) "B^1 C^1 D^2 "

                             Cross references
See also aliases2(), confound3()
```

4.4 all3anova()

Usage:

```
all3anova(y, a, b, c [,s2:val]), REAL vector y, factors a, b, c of same
  length as y, REAL scalar val > 0.
```

Keywords: anova

```
                                  Usage
all3anova(y, a, b, c) fits all hierarchical 3 factor ANOVA models.  y
must be a REAL vector and a, b and c must be factors (created by
factor() or makefactor()) of the same length as y.

For each of the 18 models fit, all3anova() prints the following, in
order of increasing C(p).
  p          Number of degrees of freedom in the model, including the
             constant term
  C(p)       Mallow's Cp statistic
  Adj R^2    Adjusted R^2
  R^2        Unadjusted R^2
  Model      The right hand side of the fitted model using the symbols
             a, b and c for the 3 factors

The estimate of variance used in computing C(p) is the error mean square
from the model fitting all factors and their interactions ("y=a*b*c").
Thus this model must not fit perfectly (have zero residual sums of
squares).

                          Keywords s2 and keep
all3anova(y, a, b, c, s2:v), where v > 0 is a REAL scalar does the same,
except v is used in computing C(p) instead of the error mean square from
the model fitting all factors and their interactions.

all3anova(y, a, b, c [, s2:v], keep:T) does the same except nothing is
printed.  Instead a structure with the following component is returned:
 Component Contents
  p        Vector of integers containing p for all the models fit
  cp       REAL vector containing C(p) for all the models fit
  adjrsq   REAL vector containing adjusted R^2 for all the models fit
  rsq      REAL vector containing unadjusted R^2 for all the models fit
  model    CHARACTER vector containing right hand side of each model fit
           using a, b and c as factor names.
The models are in increasing order of C(p).

                         Keywords keep and print
all3anova(y, a, b, c, [, s2:v], keep:T, print:T) returns a structure and
prints results.

                            Cross references
See also all4anova() and screen().
```

4.5 all4anova()

Usage:

```
all4anova(y, a, b, c, d [,s2:val]), REAL vector y, factors a, b, c, d
  with same length as y, REAL scalar val > 0.
```

Keywords: anova

```
                                  Usage
all4anova(y, a, b, c, d) fits all hierarchical 4 factor ANOVA models.  y
must be a REAL vector and a, b, c and d must be factors (created by
factor()) of the same length as y.

For each of the 166 models fit, all4anova() prints the following, in
order of increasing C(p).
  p          Number of degrees of freedom in the model, including the
             constant term
  C(p)       Mallows' Cp statistic
  Adj R^2    Adjusted R^2
  R^2        Unadjusted R^2
  Model      The right hand side of the fitted model using the symbols
             a, b, c and d for the 4 factors

The estimate of variance used in computing C(p) is the error mean square
from the model fitting all factors and their interactions ("y=a*b*c*d").
Thus this model must not fit perfectly (have zero residual sums of
squares).

all4anova(y, a, b, c, d, s2:v), where v > 0 is a REAL scalar does the
same, except v is used in computing C(p) instead of the error mean
square from the model fitting all factors and their interactions.

all4anova(y, a, b, c, d [, s2:v], keep:T) does the same except nothing
is printed.  Instead a structure with the following component is
returned:
 Component Contents
  p        Vector of integers containing p for all the models fit
  cp       REAL vector containing C(p) for all the models fit
  adjrsq   REAL vector containing adjusted R^2 for all the models fit
  rsq      REAL vector containing unadjusted R^2 for all the models fit
  model    CHARACTER vector containing right hand side of each model fit
           using a, b and c as factor names.
The models are in order of increading C(p).

                          Keywords keep and print
all4anova(y, a, b, c, d, [, s2:v], keep:T, print:T) returns a structure
and prints results.

                             Cross references
See also all3anova() and screen().
```

4.6 allaliases2()

Usage:

```
allaliases2(basis), REAL matrix basis of alias generators
```

Keywords: aliasing, design, factorial

```
                                  Usage
allaliases2(basis) finds the full set of aliases in a 2^(k-p) fractional
factorial and returns a CHARACTER vector of these aliases as its value.
k must be no larger than 25, and factors are labeled A, B, ..., H, J,
..., Z (skipping I).

The p x k matrix basis contains the generators for the aliasing, one row
for each generator and one column for each factor in the design.  The
elements in basis are 0, -1, or 1.  A nonzero entry indicates that a
factor is present in the generator for that row. The sign of a generator
is the product of the signs of the nonzero elements of the generator.
For example, 1 0 1 0 0 -1 means -ACF is a generator (alias of I).

Examples:
 Cmd> print(b,format:"2.0f")  # Matrix b is 2x5, so 2^(5-2) design
 b:
 (1,1)  1  1  1  0  0     [ABC is a generator]
 (2,1)  0  0  1  1 -1     [-CDE is a generator]

 Cmd> allaliases2(b)  # alias table
 (1) "I = ABC = -CDE = -ABDE"
 (2) "A = BC = -ACDE = -BDE"
 (3) "B = AC = -BCDE = -ADE"
 (4) "AB = C = -ABCDE = -DE"
 (5) "D = ABCD = -CE = -ABE"
 (6) "AD = BCD = -ACE = -BE"
 (7) "BD = ACD = -BCE = -AE"
 (8) "ABD = CD = -ABCE = -E"
```

4.7 boxcoxvec()

Usage:

```
boxcoxvec(Model [,powers:pow]), CHARACTER scalar Model, REAL vector pow
boxcoxvec(rhs_model,y[,powers:pow]), CHARACTER scalar rhs_model, REAL
  vectors y and pow.
```

Keywords: anova, analysis

```
                                  Usage
boxcoxvec(Model, powers:Pow), where Model is a CHARACTER scalar
specifying a GLM model of the form "y = Rhs", computes the error SS for
y transformed to the boxcox powers given in Pow.  Example: Model =
"yield = x + a + b", where response yield is a REAL vector, x is a
variate and a and b are factors.  For this model Rhs is "x + a + b".
```

```
Any factors or variates in Rhs must have the same length as y.  In the
example, x, a, and b must all be the same length as yield.

If powers:Pow is omitted, the default is Pow = run(-1,2,.25)

For each power P in Pow, the model "{boxcox(y,P)} = RHS" is fit using
anova() and the error SS is saved.  The value returned by boxcoxvec()
is structure(power:Pow, SS:Ss), where Pow and Ss are vectors of powers
and error sums of squares.

boxcoxvec(Rhs,y [,powers:Pow]), where Rhs is a quoted string or
CHARACTER, does the same as boxcoxvec("y = Rhs" [,powers:Pow]).  This
somewhat clumsier usage is maintained for backward compatibility.

A power that minimizes the error SS may be chosen to transform the data
for analysis.

Examples:
  Cmd> y <- rnorm(20);ey <- exp(y)
  NOTE: random number seeds set to 59622139 and 172924584

  Cmd> a <- factor(rep(run(5),4))

  Cmd> boxcoxvec("ey = a")  # default powers; or boxcoxvec("a", ey)
  component: power
   (1)           -1        -0.75         -0.5        -0.25            0
   (6)         0.25          0.5         0.75            1         1.25
  (11)          1.5         1.75            2
  component: SS
   (1)       179.93       72.889       34.971       21.487       17.979
   (6)       20.624       31.191       58.929       131.81       332.94
  (11)       915.71         2673       8140.8

  Cmd> boxcoxvec("ey = a",powers:run(-.25,.25,1/16))#explore powers ~= 0
  component: power
  (1)        -0.25      -0.1875       -0.125      -0.0625            0
  (6)       0.0625        0.125       0.1875         0.25
  component: SS
  (1)       21.487       19.938       18.875       18.236       17.979
  (6)       18.083       18.544        19.38       20.624

  Cmd> lineplot(boxcoxvec("ey = a"),title:"Resid SS vs Powers")#graph it

                            Cross references
See also boxcox(), lineplot(), 'models'.
```

4.8 buildfactor()

Usage:

```
a <- buildfactor(jsub, dims, [,reverse:T]), integer jsub > 0, dims a
  vector of positive integers, length(dims) >= jsub
```

Keywords: anova, factorial

```
                                  Usage
fac_j <- buildfactor(j, dims) creates a factor corresponding to the
values of subscript j >= 1 for a balanced factorial design.  dims must
be a vector of positive integers with M = length(dims) >= j.

fac_j will be a factor with dims[j] levels, with N = length(fac_j) =
prod(dims), the product of the elements of dims.

fac_j may be used in an ANOVA of a balanced factorial experiment where
each case is identified by m <= M subscripts with the first changing
fastest and the last changing slowest, with subscript i running from 1
to dims[i].  When m < M, there will be N/prod(dims[run(m)])
replications.

fac_j <- buildfactor(j, dims, reverse:T), does the same except first
subscripts change slowest.  buildfactor(j, dims, reverse:T) is
equivalent to buildfactor(M+1-j, reverse(dims)).

Example:
  Cmd> dims <- vector(2,3,2,2)#for 2 x 3 x 2 in 2 reps

  Cmd> fac_1 <- buildfactor(1,dims)

  Cmd> fac_2 <- buildfactor(2,dims)

  Cmd> fac_3 <- buildfactor(3,dims)

  Cmd> print(fac_1,fac_2,fac_3,format:"1.0f")
  fac_1:
  (1) 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
  fac_2:
  (1) 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3
  fac_3:
  (1) 1 1 1 1 1 1 2 2 2 2 2 2 1 1 1 1 1 1 2 2 2 2 2 2

  Cmd> list(fac_1,fac_2,fac_3)
  fac_1           REAL   24    FACTOR with 2 levels
  fac_2           REAL   24    FACTOR with 3 levels
  fac_3           REAL   24    FACTOR with 2 levels

These factors could be used in analysis of two replicates of a 2 by 3 by
2 design in standard order, say by anova("y=fac_1+fac_2+fac_3").

  Cmd> print(buildfactor(2,dims,reverse:T),format:"1.0f")
  VECTOR:
  (1) 1 1 1 1 2 2 2 2 3 3 3 3 1 1 1 1 2 2 2 2 3 3 3 3
```

```
Compare this with fac_2 above.

                              Cross references
See also factor(), makefactor(), rep()
```

4.9 choosedef2()

Usage:

```
   choosedef2(k,p, all:T  or  tries:m), k, p, m > 0 integer scalars
```

Keywords: confounding, design, factorial

```
                                    Usage
choosedef2(k, p, all:T) searches all potential defining contrasts to
find a set for a 2^k factorial design in 2^p blocks.  k must be an
integer between 1 and 32 and p an integer < k.

choosedef2(k,p,tries:m) does the same, except only m random sets of
defining contrasts are searched.

In both cases, the defining contrasts for the minimum aberration design
among those searched is returned.

The return value is
  structure(generators:bestgen, aberration:aber, basis:genmat).

bestgen is a CHARACTER vector of length p with the names such as "ABDF"
of the defining contrasts in the set chosen.

aber is a vector of k integers >= 0 with ab[i] containing the number
of contrast with i letters that are confounded with "I".

genmat is a p by k matrix with entries 0 and 1, one row for each element
of bestgen.  genmat[i,j] is 1 if and only if factor j is bestgen[i].
For example, if bestgen[i] is "ABDF", genmat[i,] is vector(1,1,0,1,0,1
[,...])'.  genmat can be used as an argument to aliases2() to determine
all the aliases of any contrast.

                              Cross references
See also aliases2() choosegen2(), confound2().
```

4.10 choosegen2()

Usage:

```
   choosegen2(k,p, all:T [,res:r]), positive integers k, p, r.
   choosegen2(k,p, tries:m [,res:r]), positive integers k, p, m, and r.
```

Keywords: aliasing, design, factorial

```
                                  Usage
choosegen2(k,p,all:T) , where k, p and r are positive integers, searches
all potential sets of generators for a 2^(k-p) fractional factorial
design.  It returns information on a generator with maximum resolution.

choosegen2(k,p, tries:m), where m > 0 is an integer, does the same,
except it searches only m randomly selected sets of generators.

choosegen2(k,p,all:T,res:r) and choosegen2(k,p,tries:m,res:r), where r
is a positive integer, do the same, except they stop the search at the
first design of resolution r.  If none is found, they return information
on a set having the highest resolution found.

                               Value returned
The value returned is
  structure(resolution:bestres,generators:bestgen,aberration:aber,\
      basis:genmat)

Integer bestres > 0 is the best resolution found.

bestgen is a CHARACTER vector of length p with the names such as "ABDF"
of the defining contrasts in the set chosen.

aber is a vector of k integers >= 0 with ab[i] containing the number
of contrast with i letters that are confounded with "I".

genmat is a p by k matrix with entries 0 and 1, one row for each element
of bestgen.  genmat[i,j] is 1 if and only if factor j is bestgen[i].
For example, if bestgen[i] is "ABDF", genmat[i,] is vector(1,1,0,1,0,1
[,...])'.  genmat can be used as an argument to aliases2() to determine
all the aliases of any contrast.

Examples:
  Cmd> choosegen2(5,2,all:T)  # find best resolution
  component: resolution
  (1)           3
  component: generators
  (1) "ABCD"
  (2) "BCE"
  component: aberration
  (1)           0           0           2           1           0
  component: basis
  (1,1)           1           1           1           1           0
  (2,1)           0           1           1           0           1

  Cmd> choosegen2(5,2,all:T,res:4)  # try to find 4 (we won't)
  component: resolution
  (1)           3
  component: generators
  (1) "ABCD"
  (2) "BCE"
  component: aberration
```

```
(1)           0           0           2           1           0
component: basis
(1,1)           1           1           1           1           0
(2,1)           0           1           1           0           1

Cmd> # look for 2^(9-4) resol. 4, just make 1000 tries, since\
     there are lots of combinations to explore

Cmd> choosegen2(9,4,tries:1000,res:4)# look for 2^(9-4) resol. 4
component: resolution
(1)           4
component: generators
(1) "CDEF"
(2) "BDEG"
(3) "ABCEH"
(4) "BCDJ"
component: aberration
(1)           0           0           0           7           7
(6)           0           0           0           1
component: basis
(1,1)           0           0           1           1           1
(1,6)           1           0           0           0
(2,1)           0           1           0           1           1
(2,6)           0           1           0           0
(3,1)           1           1           1           0           1
(3,6)           0           0           1           0
(4,1)           0           1           1           1           0
(4,6)           0           0           0           1

                          Cross references
See also aliases2(), choosedef2(), confound2().
```

4.11 confound2()

Usage:

```
confound2(basis), REAL matrix basis containing confounding generators
```

Keywords: confounding, design, factorial

```
                                  Usage
confound2(basis) confounds a two series factorial into blocks based on
the generators given in the matrix basis.  Results are returned in a
structure with component names 'block1', 'block2', etc.  Each component
has a CHARACTER vector of factor/level combinations for that block.

The p x k matrix basis contains the generators for the confounding, one
row for each generator and one column for each factor in the design.
The elements in basis are 0 or 1.  A nonzero entry indicates that a
factor is present in the generator for that row.
```

```
Examples:
  Cmd> # Matrix d is 2x4 (2 generators so 4 blocks of size 4)

  Cmd> print(d,format:"2.0f")
  d:
  (1,1)  1  1  0  0     [AB is a generator]
  (2,1)  0  1  1  1     [BCD is a generator]

  Cmd> confound2(d)
  component: block1
  (1) "(1)"
  (2) "abc"
  (3) "abd"
  (4) "cd"
  component: block2
  (1) "a"
  (2) "bc"
  (3) "bd"
  (4) "acd"
  component: block3
  (1) "ab"
  (2) "c"
  (3) "d"
  (4) "abcd"
  component: block4
  (1) "b"
  (2) "ac"
  (3) "ad"
  (4) "bcd"

                            Cross references
See also aliases2(), choosegen2(), choosedef2().
```

4.12 confound3()

Usage:

```
confound3(basis), REAL matrix basis containing confounding generators
```

Keywords: confounding, design, factorial

```
                                    Usage
confound3(basis) confounds a three series factorial into blocks based on
the generators given in the matrix basis.  Results are returned in a
structure with component names 'block1', 'block2', etc.  Each component
has a CHARACTER vector of factor/level combinations for that block.

The p x k matrix basis contains the generators for the confounding, one
row for each generator and one column for each factor in the design.
The elements in basis are 0, 1, or 2, indicating the exponent of each
factor in the generator.
```

```
Example:
  Cmd> print(e,format:"2.0f")# Matrix e is 1x2 (one generator in a 3^2)
  e:
  (1,1)  1  2        [A^1 B^2 is a generator]

  Cmd> confound3(e)
  component: block1
  (1) "00"
  (2) "11"
  (3) "22"
  component: block2
  (1) "10"
  (2) "21"
  (3) "02"
  component: block3
  (1) "20"
  (2) "01"
  (3) "12"

                            Cross references
See also aliases3(), confound2()
```

4.13 doconfound2()

Usage:

```
doconfound2(p:fctrs,k:blcks or cfdout:str or basis:mat [,confeff:T]\
       [,aber:T] [,assign:T]) fctrs and blocks positive integer
scalars, str the output from choosedef2, mat a basis
matrix for confounding
```

Keywords: confounding, design, factorial

```
doconfound2() is a front end combining many of the tasks of confound2(),
choosedef2(), and aberration2().  There are two sorts of arguments:
those that determine the design, and those that determine what is
printed.  All arguments are keyword phrases.

To determine the design, you may enter p and k, or cfdout, or basis.
Using p and k says to create a new design in 2^p treatments (p factors)
with 2^k blocks.  The other options use a previously created design,
either cfdout, the output from a previous usage of choosedef2(), or
basis, a k by p matrix of 0's and 1's indicating the defining contrasts
for the design.

The remaining three arguments determine what is printed:
confeff:T      Print all confounded effects
aber:T         Print the number of confounded effects by word length
assign:T       Print the assignment of treatments to blocks.

See also: choosedef2(), aberration2(), confound2().
```

4.14 doff2()

Usage:

```
doff2(p:fctrs,k:blcks or alout:str or basis:mat [,Ialiases:T]\
       [,allalias:T] [,aber:T] [,showfrac:T] [random:T])
fctrs and blocks positive integer scalars, str the
        output from choosegen2, mat a basis matrix for
aliasing
```

Keywords: aliasing, design, factorial

```
doff2() is a front end combining many of the tasks of aliases2(),
choosegen2(), and aberration2().  There are two sorts of arguments:
those that determine the design, and those that determine what is
printed.  All arguments are keyword phrases.

To determine the design, you may enter p and k, or alout, or basis.
Using p and k says to create a new 2^(k-p) design. The other options
use a previously created design, either alout, the output from a
previous usage of choosegen2(), or basis, a k by p matrix of 0's
and 1's indicating the generators for the design.  The phrase
random:T means to choose a random fraction from the family determined
by the generators.

The remaining four arguments determine what is printed:
Ialiases:T     Print the aliases of I
allalias:T     Print all aliases
aber:T         Print the number of confounded effects by word length
showfrac:T     Print the treatments in the fraction.

See also: choosegen2(), aberration2(), aliases2(), allaliases2().
```

4.15 ems()

Usage:

```
ems(Model,randomvars[,marg:T] [,restrict:F] [,nonhier:T] [,keep:T]\
  [,print:T]), CHARACTER scalar model, CHARACTER vector randomvars
```

Keywords: anova, analysis, factorial

```
                                Usage
ems(Model,Randomvars) computes the expected mean squares for the terms
in the ANOVA for the model given in CHARACTER scalar Model.  Randomvars
is a CHARACTER vector specifying the names of factors in the model which
are random.  Randomvars can also be REAL with integer elements
specifying the index of a factor in the model.  If there are no random
factors, Randomvars should be NULL.
```

In this default use, ems() computes sequential (Type I) sums of squares for the the restricted (mixed effects add to zero across fixed factors) model, prints these expected means squares for each term, and returns no value. Contributions from random terms are shown as multiples of the variance component (for example, 16V(a.b)); contributions from fixed terms are shown as a multiple of a quadratic function for the term, for example, 32Q(c). In a balanced design, the Q() function is the sum of the squared coefficients divided by degrees of freedom, for example, sum(cˆ2)/(K-1). In an unbalanced situation, Q(c) is a more complicated quantity defined using matrix algebra.

See below for the use of keywords to change the action of ems().

ems() works only for factors -- no variates are allowed in the model.

ems() works for both balanced and unbalanced data.

Details

ems() assumes that if a factor first appears in an interaction, then that factor is nested in the other terms of the interaction. For example, if the first appearance of factor c is in the term a.b.c, then c is assumed nested in the a.b combinations. This nesting is assumed in the remainder of the model. That is, continuing the example, if there is a later term c.d, it will be interpreted as a.b.c.d even though a.b.c.d is not specifically in the model.

When a term contains the first appearance in the model of more than one factor, ems() assumes that the new factors are merged to make a single factor, whose number of levels is the product of the numbers of levels in the factors being merged. For example, if the first appearance of factors b and c with 5 and 3 levels, respectively is in the term a.b.c, then b and c together are considered a single factor with 15 levels. This grouping is assumed in the remainder of the model. That is, continuing the example, if there is a later term c.d, it will be interpreted as b.c.d even though b.c.d is not specifically in the model. This grouped factor is interpreted as random if any of the factors in the group is random.

ems() uses the "synthesis" method of Hartley, as explained in 10.5.2 of R. R. Hocking (1985), The Analysis of Linear Models, Brooks/Cole, Belmont, CA.

Examples

The examples below are based on balanced two factor and three factor models with a total of 64 responses. All factors have 2 levels so two factor and three factor models have 16 and 8 replications, respectively. In some examples, one of the responses is set to MISSING to destroy balance.

A fully nested model, with c fixed and both d and e random.

```
  Cmd> ems("y=c/d/e",vector("d","e"))
  EMS(CONSTANT) = V(ERROR1) + 8V(c.d.e) + 16V(c.d) + 64Q(CONSTANT)
  EMS(c) = V(ERROR1) + 8V(c.d.e) + 16V(c.d) + 32Q(c)
```

```
  EMS(c.d) = V(ERROR1) + 8V(c.d.e) + 16V(c.d)
  EMS(c.d.e) = V(ERROR1) + 8V(c.d.e)
  EMS(ERROR1) = V(ERROR1)

A 3 factor crossed model, with c and d fixed, e random.
  Cmd> ems("y=c*d*e",3) # e is factor 3
  EMS(CONSTANT) = V(ERROR1) + 32V(e) + 64Q(CONSTANT)
  EMS(c) = V(ERROR1) + 16V(c.e) + 32Q(c)
  EMS(d) = V(ERROR1) + 16V(d.e) + 32Q(d)
  EMS(c.d) = V(ERROR1) + 8V(c.d.e) + 16Q(c.d)
  EMS(e) = V(ERROR1) + 32V(e)
  EMS(c.e) = V(ERROR1) + 16V(c.e)
  EMS(d.e) = V(ERROR1) + 16V(d.e)
  EMS(c.d.e) = V(ERROR1) + 8V(c.d.e)
  EMS(ERROR1) = V(ERROR1)

A 2 factor crossed model with unbalanced data, c fixed and d random
  Cmd> y1 <- y[1]; y[1] <- ? # make data unbalanced

  Cmd> ems("y=c*d",2) # d is factor 2
  EMS(CONSTANT) = V(ERROR1) + 0.0080645V(c.d) + 31.508V(d) +
    0.0079365Q(c) + 63Q(CONSTANT)
  EMS(c) = V(ERROR1) + 15.746V(c.d) + 0.0081925V(d) + 31.492Q(c)
  EMS(d) = V(ERROR1) + 0.0042316V(c.d) + 31.484V(d)
  EMS(c.d) = V(ERROR1) + 15.742V(c.d)
  EMS(ERROR1) = V(ERROR1)

            Use of Keywords to change the action of ems().
ems(Model,Randomvars,keep:T) suppresses printed output but returns a
structure (described below) containing the results.  If you want the
printed output too, use keep:T,print:T.

ems(Model,Randomvars,marg:T) computes expected mean squares based on
adjusted (Type III) sums of squares.

ems(Model,Randomvars,restrict:F) computes expected mean squares assuming
no marginal restrictions on any random effects in the model that have
two or more dimensions.  Thus, for example, when the model is
"y=c+D+c.D", where c is fixed and D is random, it is not assumed that
the c.D effects sum to zero for each level of D.

ems(Model,Randomvars,nonhier:T) computes expected mean squares for an
analysis of variance that does not enforce the usual MacAnova hierarchy
assumptions.  That is, for example, model "y=a+b+c+a.b.c" does not imply
that the two-way interaction degrees of freedom are part of the "a.b.c"
term.  You cannot use anova() to compute such an analysis although it
can be done (if you know how) using swp().

These keywords can be used together.  For example, ems(Model,Randomvars,
marg:T,restrict:F) provides answers equivalent to the EMS in SAS PROC
GLM.

                          More examples
```

```
These examples still assume y[1] is MISSING.
  Cmd> ems("y=c*d",2,marg:T) # crossed with d random
  EMS(CONSTANT) = V(ERROR1) + 31.475V(d) + 62.951Q(CONSTANT)
  EMS(c) = V(ERROR1) + 15.742V(c.d) + 31.475Q(c)
  EMS(d) = V(ERROR1) + 31.475V(d)
  EMS(c.d) = V(ERROR1) + 15.742V(c.d)
  EMS(ERROR1) = V(ERROR1)

  Cmd> ems("y=c*d",2,restrict:F) # crossed with d random
  EMS(CONSTANT) = V(ERROR1) + 15.762V(c.d) + 31.508V(d) + 0.0079365Q(c)
     + 63Q(CONSTANT)
  EMS(c) = V(ERROR1) + 15.754V(c.d) + 0.0081925V(d) + 31.492Q(c)
  EMS(d) = V(ERROR1) + 15.746V(c.d) + 31.484V(d)
  EMS(c.d) = V(ERROR1) + 15.738V(c.d)
  EMS(ERROR1) = V(ERROR1)

  Cmd> ems("y=c*d",2,marg:T,restrict:F) # same as SAS PROC GLM
  EMS(CONSTANT) = V(ERROR1) + 15.738V(c.d) + 31.475V(d) + 62.951Q(CONSTANT)
  EMS(c) = V(ERROR1) + 15.738V(c.d) + 31.475Q(c)
  EMS(d) = V(ERROR1) + 15.738V(c.d) + 31.475V(d)
  EMS(c.d) = V(ERROR1) + 15.738V(c.d)
  EMS(ERROR1) = V(ERROR1)

  Cmd> y[1] <- y1 # restore value for y[1] to regain balance

  Cmd> ems("y=c*d+e+c.d.e",3)
  EMS(CONSTANT) = V(ERROR1) + 32V(e) + 64Q(CONSTANT)
  EMS(c) = V(ERROR1) + 8V(c.d.e) + 32Q(c)
  EMS(d) = V(ERROR1) + 8V(c.d.e) + 32Q(d)
  EMS(c.d) = V(ERROR1) + 8V(c.d.e) + 16Q(c.d)
  EMS(e) = V(ERROR1) + 32V(e)
  EMS(c.d.e) = V(ERROR1) + 8V(c.d.e)
  EMS(ERROR1) = V(ERROR1)

  Cmd> ems("y=c*d+e+c.d.e",3,nonhier:T)
  EMS(CONSTANT) = V(ERROR1) + 32V(e) + 64Q(CONSTANT)
  EMS(c) = V(ERROR1) + 32Q(c)
  EMS(d) = V(ERROR1) + 32Q(d)
  EMS(c.d) = V(ERROR1) + 8V(c.d.e) + 16Q(c.d)
  EMS(e) = V(ERROR1) + 32V(e)
  EMS(c.d.e) = V(ERROR1) + 8V(c.d.e)
  EMS(ERROR1) = V(ERROR1)

Note that 'nonhier:T' makes the c.d.e term disappear from the EMS for
the fixed terms; compare with the default hierarchical model that
precedes it.

                    Structure returned with keep:T
When 'keep:T' is an argument, the structure returned has components
'df', 'ss', 'termnames', 'coefs', and 'rterms'.
Component   Description
 df         REAL Vector of degrees of freedom for all terms in model
 ss         REAL Vector of sums of squares for all terms in model
```

```
 termnames  CHARACTER vector of labels for each term
 coefs      REAL matrix with coefs[i,j] the coefficient for term j in
            the EMS of term i
 rterms     LOGICAL vector with T indicating that a term is random.

Components ss and df are just those computed from a MacAnova anova()
command (possibly with marg:T as needed), and may not be in conformance
with the model as used by ems() for the following reasons:
1. anova() computes only hierarchical models, while you may specify
   nonhierarchical models in ems() by using nonhier:T.
2. ems() enforces nesting and grouping.  If b first appears in a.b then
   b is nested in a and any appearance of b in a later term implies the
   presence of a.  anova() does no such enforcing.  For example, in
   "y=a+a.b+c+b.c", b.c would be interpreted by ems() as a.b.c while
   anova() would not include a.b.c in the model.  If b and c first
   appear together, then "y=b.c+d+c.d" is interpreted in ems() as
   "y=b.c+d+b.c.d".

                                Examples
These examples assume a 2^2 design in 16 replicates:
  Cmd> ems("y=c*d",2)
  EMS(CONSTANT) = V(ERROR1) + 32V(d) + 64Q(CONSTANT)
  EMS(c) = V(ERROR1) + 16V(c.d) + 32Q(c)
  EMS(d) = V(ERROR1) + 32V(d)
  EMS(c.d) = V(ERROR1) + 16V(c.d)
  EMS(ERROR1) = V(ERROR1)

  Cmd> ems("y=c*d",2,keep:T)
  component: df
  (1)          1          1          1          1         60
  component: ss
  (1)    0.76155   0.036871    0.31116      1.623     56.318
  component: termnames
  (1) "CONSTANT"
  (2) "c"
  (3) "d"
  (4) "c.d"
  (5) "ERROR1"
  component: coefs
  (1,1)         64          0         32          0          1
  (2,1)          0         32          0         16          1
  (3,1)          0          0         32          0          1
  (4,1)          0          0          0         16          1
  (5,1)          0          0          0          0          1
  component: rterms
  (1) F       F       T       T
```

4.16 ffdesign2()

Usage:

```
ffdesign2(basis), integer matrix basis containing confounding generators
```

Keywords: design, aliasing, factorial

```
                                  Usage
ffdesign2(basis) finds the set of factor/level combinations used in the
2^(k-p) fractional factorial corresponding to the given generators.  The
result is a CHARACTER vector giving the factor/level combinations.

The p x k matrix basis contains the generators for the aliasing, one row
for each generator and one column for each factor in the design.  The
elements in basis are 0, -1, or 1.  A nonzero entry indicates that a
factor is present in the generator for that row. The sign of a generator
is the product of the signs of the nonzero elements of the generator.
For example, 1 0 1 0 0 -1 means -ACF is a generator (alias of I).

Examples:
  Cmd> print(b, format:"2.0f") # Matrix b is 2x5, so 2^(5-2) design
  b:
  (1,1)  1  1  1  0  0      [ABC is a generator]
  (2,1)  0  0  1  1 -1      [-CDE is a generator]

  Cmd> ffdesign2(b)
  (1) "ce"
  (2) "a"
  (3) "b"
  (4) "abce"
  (5) "dc"
  (6) "ade"
  (7) "bde"
  (8) "abdc"
```

4.17 findncp()

Usage:

```
findncp(means,nis,sigma2) means a REAL vector, nis a vector of positive
integers, sigma2 a positive REAL; means and nis must be the same length.
```

Keywords: design

```
                                  Usage
findncp(means,nis,sigma2) computes the noncentrality for the F-test in
the one-way anova testing the null hypothesis of no treatment
differences when the means, sample sizes, and error variance are as
given.  Arguments means and nis must be vectors of the same length
(one element for each treatment).  Argument nis must be postive
integers and sigma2 must be a postive REAL.
```

```
                                Examples
Here we have three treatment groups with means 2.2, 2.8, and 3.1, and
sample sizes 2, 2, and 8 respectively.  The error variance is 2, and
the noncentrality parameter is found to be .66
  Cmd> findncp(vector(2.2,2.8,3.1),vector(2,2,8),2)
  (1)         0.66
```

4.18 findpower()

Usage:

```
findpower(means,nis,sigma2,alpha) means a REAL vector, nis a vector of
positive integers, sigma2 and alpha positive REALs; means and nis must
be the same length and alpha must be less than 1.
findpower(ncp,df1,df2,alpha), all positive scalars.
```

Keywords: design

```
                                  Usage
findpower(means,nis,sigma2,alpha) computes the power for the F-test in
the one-way anova testing the null hypothesis of no treatment
differences when the means, sample sizes, error variance, and type 1
error rate are as given.  Arguments means and nis must be vectors of the
same length (one element for each treatment).  Argument nis must be
postive integers; sigma2 must be a postive REAL, and alpha must be
between 0 and 1.

findpower(means,nis,sigma2,alpha,rcb:T) computes the power for the same
set up considered to be a randomized complete block. This requires that
all groups have the same sample size.

findpower(ncp,df1,df2,alpha) is a synonym for power2(ncp,df1,df2,alpha)

                                Examples
Here we have three treatment groups with means 2.2, 2.8, and 3.1, and
sample sizes 2, 2, and 8 respectively.  The error variance is 2 and
the type 1 error rate is .05; power is found to be .088
  Cmd> findpower(vector(2.2,2.8,3.1),vector(2,2,8),2,.05)
  (1)     0.087928

Instead suppose that we had 40 observations in each treatment, then the
power is .73
  Cmd> findpower(vector(2.2,2.8,3.1),vector(40,40,40),2,.05)
  (1)      0.72816
```

4.19 findsampsize()

Usage:

```
findsampsize(means,sigma2,alpha,pow[,rcb:T,prop:propvec])
means a REAL vector, sigma2, alpha, and pow positive REALs; alpha
and pow must be less than 1. Elements of propvec must be positive,
and means and propvec must be the same length. Cannot use rcb:
and prop: together.
findsampsize(ncp1,ngrps,alpha,pow[,rcb:T]) all arguments positive
scalars; ngrps is integer.
```

Keywords: design

```
                                  Usage
findsampsize(means,sigma2,alpha,pow) computes the minimum sample size
for the power of the F-test in the one-way anova testing the null
hypothesis of no treatment differences to be at least pow when the
means, sample sizes, and error variance are as given.  Argument sigma2
must be positive; alpha and pow must be between 0 and 1.  This usage
assumes that the sample sizes will be equal.  The result is a structure
with components for the sample sizes and the achieved power.

findsampsize(ncp1,ngrps,alpha,pow) computes the minimum sample size
for the power of the F-test in the one-way anova testing the null
hypothesis of no treatment differences to be at least pow when the
n=1 noncentrality parameter, number of groups, and error variance
are as given.  ncp1 must be positive. This usage assumes that the
sample sizes will be equal.  The result is a structure with
components for the sample sizes and the achieved power.

                                 Examples
Here we have three treatment groups with means 2.2, 2.8, and 3.1, the
error variance is 2, the type 1 error rate is .05, and the minimum
acceptable power is .7; the required sample size is 38
  Cmd> findsampsize(vector(2.2,2.8,3.1),2,.05,.7)
  component: nis
  (1)           38           38           38
  component: power
  (1)       0.7039

Here we have three groups with a noncentrality of .21, the type 1
error is .05, and the minimum power is .7, then
  Cmd> findsampsize(.21,3,.05,.7)
  component: nis
  (1)           38           38           38
  component: power
  (1)       0.7039
(This is just the previous example.)


                              Keyword prop
The additional keyword argument prop:propvec may also be used.  propvec
must be a vector of positive reals the same length as means.  In this
usage, findsampsize() will find a vector of sample sizes that achieves
```

```
the desired power but is (nearly) proportional to propvec.

                                Examples
Here we have three treatment groups with means 2.2, 2.8, and 3.1, the
error variance is 2, the type 1 error rate is .05, and the minimum
acceptable power is .7; we also require that the sample sizes be in
proportion 1:1:4
  Cmd> findsampsize(vector(2.2,2.8,3.1),2,.05,.7,prop:vector(1,1,4))
  component: nis
  (1)          24          24          94
  component: power
  (1)      0.70055

                              Keyword rcb
The keyword argument rcb:T is used to indicate that the error df
should be computed as for a randomized complete block.  This will
usually increase the needed sample size (all other parameters held
constant, although the whole point of blocking is to reduce the error
variance).

                                Examples
Here we have three treatment groups with means 2.2, 2.8, and 3.1, the
error variance is 2, the type 1 error rate is .05, and the minimum
acceptable power is .7; we also require an RCB design.
  Cmd> findsampsize(vector(2.2,2.8,3.1),2,.05,.7,rcb:T)
  component: nis
  (1)          39          39          39
  component: power
  (1)      0.71011
```

4.20 interactplot()

Usage:

```
interactplot(y,a [,b,c ...] [errors:T or errors:x,pool:T|F,errorvar:v]
  [graphics keywords]), y a REAL vector, a, b, c, ... vectors of positive
  integers the same length as y, x and v positive scalars.
interactplot(means [errors:T or errors:x,errormat:s] [graphics
  keywords]), means a REAL matrix or array,x a positive scalar, s a
  positive matrix the same shape as means.
interactplot(a [,b,c ...],frommodel:T,[errors:T or errors:x] [graphics
  keywords]), a, b, c, ... factors in the current model, x a positive
  scalar.
```

Keywords: factorial, plots

```
                            Usage_with_data
interactplot(y,a [,b,c ...] [graphics keywords]) makes an interaction
plot of the marginal means of REAL vector y for all combinations of
variables a, b, c, ....  The variables a, b, ... must be vectors of
```

positive integers the same length as y.

The levels of variable a will be put on the horizontal axis and separate lines drawn for each combination of variables values of b, c, ... When a is the only factor argument, only one line is drawn. Lines are numbered lb.lc.ld ... with lx denotine the level of factor x.

Usage_with_means

interactplot(means [graphics keywords]) where means is a REAL matrix or array will make an interaction plot with the first dimension of means on the horizontal axis, and separate lines for each combination of the other dimensions. The lines are numbered 1, 2, 3, ... with the last dimension varying slowest. interactplot(tabs(y,a,b,means:T)) makes the same plot as interactplot(y,a,b).

Usage_with_models

interactplot(a[,b,c ...],frommodel:T [graphics keywords]) makes an interaction plot of the least squares means of the term a.b.c ... from the most recent glm model. This is similar to doing a plot of the matrix glmtable(a,b,c,estimate:T,seest:F). NOTE: because this usage involves glmtable(), it will not work for balanced anova() models unless unbal:T was used as an argument.

Usage_with_errors

Use of the argument errors:T or errors:x cause interactplot to draw error bars around each mean (and slightly offset the horizontal plotting positions). errors:T is equivalent to errors:2, meaning that the bars should be plus or minus two standard errors. errors:x will plot bars that are plus or minus x standard errors.

When errors:T is used with frommodel:T, errors for the LS means are determined from the model via glmtable().

When errors:T is used with a matrix of means, you must specify the standard errors via errormat:s, where s is a real matrix of the same shape as the means.

When errors:T is used with data and splitting factors, the standard errors are computed as the square root of a variance divided by the number of data elements in each mean. By default, a separate variance is estimated from the data used for each mean. Use of pool:T pools all of the variance estimates into a single estimate of variance. Use of errorvar:v causes v to be used as the variance for all means, regardless of the variability in the data.

Graphics keywords

In both usages, any graphics keywords will be passed to function chplot() which actually makes the plot. In particular, for example,

```
  Cmd> interactplot(y,a,b,symbols:vector("B1","B2","B3"),\
        xticklabs:vector("A1","A2","A3","A4"))
```

will label the curves B1, B2 and B3 instead of 1, 2 and 3 and the

```
horizontal axis location A1, A2, A3 and A4.
```

4.21 interblock()

Usage:

```
interblock(y,block,trt,[contrast:coefs])
```

Keywords: analysis, anova

```
                                  Usage
interblock(y,block,trt,[contrast:coefs]) does recovery of interblock
information in an incomplete block design, where y is the vector of
responses, block is the factor of block levels, trt is the factor of
treatment levels, and coefs is an optional vector of contrast
coefficients (it must be the same length as the number of levels of
trt).  If no contrast is specified, the output is the intra-block,
inter-block, and combined estimates of treatment effects and their
standard errors.  If a contrast is specified, the output is the
intra-block, inter-block, and combined estimates of the contrast with
their standard errors.

Example:

  Cmd> y<-vector(19,17,11,6,26,23,21,19,28,20,7,20,17,26,19,15,23,31,\
       20,26,31,16,23,21,13,7,20,20,24,19,17,6,29,14,24,21)

  Cmd> session<-factor(1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6,7,7,7,8,8,\
       8,9,9,9,10,10,10,11,11,11,12,12,12)

  Cmd> trt<-factor(1,2,3,4,5,6,7,8,9,1,4,7,2,5,8,3,6,9,1,5,9,2,6,7,3,4,\
       8,1,6,8,2,4,9,3,5,7)

  Cmd> interblock(y,session,trt)
     intra est   intra se  inter est   inter se combined est combined se
       0.33333    0.49414    0.33333    0.91174     0.33333     0.43443
       -2.2222    0.49414         -4    0.91174     -2.6259     0.43443
       -6.2222    0.49414         -6    0.91174     -6.1718     0.43443
       -12.889    0.49414        -13    0.91174     -12.914     0.43443
        5.8889    0.49414     6.6667    0.91174      6.0655     0.43443
        3.5556    0.49414     4.6667    0.91174      3.8078     0.43443
        1.6667    0.49414    0.33333    0.91174      1.3639     0.43443
      -0.22222    0.49414 8.8818e-16    0.91174    -0.17177     0.43443
        10.111    0.49414         11    0.91174      10.313     0.43443

  Cmd> cfs<-vector(.25,.25,.25,.25,-.25,-.25,-.25,-.25,0)

  Cmd> interblock(y,session,trt,contrast:cfs)
               estimate           se
  intra         -7.9722       0.3706
  inter         -8.5833      0.68381
```

```
  combined       -8.111      0.32583
```

4.22 mixed()

Usage:

```
   mixed(Model,randomvars[,marg:T] [,restrict:F] [,nonhier:T] [,useneg:T]\
     [,keepmixed:T]), CHARACTER scalar model, CHARACTER vector randomvars
   mixed(emsResult [,useneg:T] [,keepmixed:T]), emsResult a structure
     returned by ems() with keep:T
```

Keywords: anova, analysis, random effects, factorial

```
                                    Usage
mixed(Model, randomvars) computes and prints an "ANOVA" table
appropriate for the model and random factors given in CHARACTER scalar
Model and CHARACTER vector randomvars.  These arguments are exactly the
same as for ems().  You can also use ems() keywords 'marg', 'restrict'
and 'nonhier'.

mixed(emsResult), where emsResult has been computed by emsResult <-
ems(Model, randomvars [,...],keep:T), does the same.

mixed(Model, randvars [,...], keepmixed:T) returns the table as an
matrix with appropriately labelled rows and columns but does not print
it.

The anova table has one row for each term in the model and the following
seven columns.
  Col. 1   Label for term
  Col. 2   DF = degrees of freedom for term
  Col. 3   MS = mean square for term (numerator of F)
  Col. 4   Error DF = degrees of freedom for appropriate error term
  Col. 5   Error MS = mean square for appropriate error term
           (denominator of F)
  Col. 6   F = F-statistic = MS/(Error MS)
  Col. 7   P value = tail probability for F test

With keepmixed:T, the matrix returned consists of columns 2 through 7 of
the table, with column 1 used to label rows.

                Description of numerator and denominator MS
Numerator and denominator MS's are linear combinations of mean squares
whose expectations differ only by a multiple of the variance component
associated with the line.  When the numerator or denominator is not a
simple ANOVA mean square, its degrees of freedom are found using the
Satterthwaite approximation.

By default, only linear combinations of mean squares with positive
coefficients are used.  This means that the numerator for a term may be
the sum of the mean square for the term and one or more mean squares
from other terms.  If the keyword useneg:T is used, then the numerator
```

for a term will be the mean square for that term, and denominators may contain differences as well as sums of mean squares.

Example:
Three populations, all crosses between 4 males and 4 females in each population with six offspring from each mating randomly assigned to three environments. Male and female are random. First the simple ANOVA.

```
  Cmd> anova("y=(pop+m.pop+f.pop+m.f.pop)*env")
  Model used is y=(pop+m.pop+f.pop+m.f.pop)*env
                  DF           SS           MS
  CONSTANT         1       5.4299       5.4299
  pop              2       2091.4       1045.7
  pop.m            9        112.5         12.5
  pop.f            9       370.02       41.113
  pop.m.f         27       56.774       2.1027
  env              2       206.15       103.08
  pop.env          4      0.16527     0.041316
  pop.m.env       18       3.4185      0.18992
  pop.f.env       18       8.2354      0.45752
  pop.m.f.env     54       17.117      0.31698
  ERROR1         144       30.448      0.21144
```

Now compute the expected mean squares, and keep the ems() output.

```
  Cmd> emsstuff<-ems("y=(pop+m.pop+f.pop+m.f.pop)*env",vector("m","f"),
     keep:T,print:T)
  EMS(CONSTANT) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f) + 24V(pop.m) +
     288Q(CONSTANT)
  EMS(pop) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f) + 24V(pop.m) + 96Q(pop)
  EMS(pop.m) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.m)
  EMS(pop.f) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f)
  EMS(pop.m.f) = V(ERROR1) + 6V(pop.m.f)
  EMS(env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env) +
     8V(pop.m.env) + 96Q(env)
  EMS(pop.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env) +
     8V(pop.m.env) + 32Q(pop.env)
  EMS(pop.m.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.m.env)
  EMS(pop.f.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env)
  EMS(pop.m.f.env) = V(ERROR1) + 2V(pop.m.f.env)
  EMS(ERROR1) = V(ERROR1)
```

Now use mixed().

```
  Cmd> mixed(emsstuff)
                  DF        MS  Error DF  Error MS         F    P value
  CONSTANT     1.914     7.533     14.01     53.61    0.1405     0.8617
  pop          2.008      1048     14.01     53.61     19.54  8.745e-05
  pop.m            9      12.5        27     2.103     5.945  0.0001412
  pop.f            9     41.11        27     2.103     19.55  1.242e-09
  pop.m.f         27     2.103       144    0.2114     9.945          0
  env          2.012     103.4     30.75    0.6474     159.7          0
```

```
  pop.env      56.12    0.3583     30.75    0.6474    0.5534     0.9729
  pop.m.env       18    0.1899        54     0.317    0.5991     0.8844
  pop.f.env       18    0.4575        54     0.317     1.443     0.1496
  pop.m.f.env     54     0.317       144    0.2114     1.499    0.03044
  ERROR1         144    0.2114         0         0   MISSING    MISSING

The test for environment should be
(MS(env)+MS(m.f.env))/(MS(m.env)+MS(f.env)) = (103.08+.32)/(.190+.458) =
(103.4)/(.6474) = 159.7 as reported in the table.

                            Cross references
See also ems(), reml().
```

4.23 pairwise()

Usage:

```
   pairwise(factorname,lev [,method:T] [,error:term]), CHARACTER scalar
     factorname, positive REAL scalar lev < 1, positive integer or
     CHARACTER scalar term, keyword phrase method:T one of 'lsd:T',
     'bsd:T', 'snk:T', 'hsd:T', 'regwb:T', or 'regwr:T'
   pairwise(factorname,critval:val), positive REAL scalar val
```

Keywords: anova, analysis

```
                                   Usage
pairwise(factorname,siglevel) prints a summary of all paired comparisons
between the levels of the factor given in factorname at the level of
significance siglevel.  Comparisons are done using the Bonferroni
method.  factorname must be a CHARACTER scalar or quoted string naming a
factor in the current GLM model and siglevel must be a REAL scalar
between 0 and 1.  It is an error if there is no current GLM model or if
the current GLM model does not contain the named factor.

                                  Methods
pairwise(factorname,siglevel,METHOD:T), where METHOD:T is one of
'bsd:T', 'lsd:T', 'snk:T', 'hsd:T', 'regwb:T', or 'regwr:T', does the
same, except METHOD specifies the multiple comparison method to be used.
  METHOD       Description
   bsd         Bonferroni method (the default)
   lsd         Least significant difference method
   hsd         Tukey's honestly significant difference or Studentized
               range method
   snk         Student-Newman-Keuls method
   regwb       Step down Bonferroni using REGW tail probabilities
   regwrs      Step down Studentized range using REGW tail probabilities

The REGW tail probabilities were proposed in papers by Ryan, Einot and
Gabriel, and Welsch.

                               Keyword critval
pairwise(factorname,critval:val) does the same, except it uses val as
```

```
the critical value for a t-test between the levels of factorname rather
than a computed cutoff.

Examples:
After anova("y=trt"),
  Cmd> pairwise("trt",.01,hsd:T)
does paired comparisons between the levels of trt at signficance .01
using the HSD method.

  Cmd> pairwise("trt",\
         critval:invstudrng("trt",1 - .01, max(trt), DF[3])/sqrt(2))
does the same, directly computing the HSD critical value.  See
invstudrng().

pairwise() prints only a summary of the results and returns no value.
The printed output consists of one row for each level of the term,
sorted from smallest to largest effect, giving the "underlines"
identifying effects that are not significantly different, level number,
and effect.

                              Error mean square
By default, the error mean square used in the comparison tests is taken
from the last error term of the current model (the last line of the
ANOVA table).  You may specify a different error term with keyword
phrase error:term.  term must be a CHARACTER scalar or positive integer
which specifies the name or number of the line in the ANOVA table to be
used as the error mean square.  Examples are 'error:4' (use line 4 as
error term) or 'error:"a.b.c"' (use the ABC interaction as error term).

                              Use of contrast()
The contrast() command is used to make each comparison.  In particular
this implies that the comparisons are adjusted for any other terms in
the model and that there should be no missing degrees of freedom in the
factor.

Examples:
  Cmd> anova("y=a")
  Model used is y=a
                   DF           SS           MS
  CONSTANT          1       15.082       15.082
  a                 4       67.535       16.884
  ERROR1           15       20.132       1.3421

  Cmd> pairwise("a",.05,hsd:T) #hsd method
    |        5    -1.86
    | |      2    -1.18
    | |      4    -1.15
      | |    3     1.13
        |    1     3.07

  Cmd> pairwise("a",.05,lsd:T) #lsd w/ alpha=.05
    |    5    -1.86
    |    2    -1.18
```

```
    |    4    -1.15
         3     1.13
         1     3.07

  Cmd> pairwise("a",critval:2.13) #lsd w/ alpha=.05 a different way
    |    5    -1.86
    |    2    -1.18
    |    4    -1.15
         3     1.13
         1     3.07
```

4.24 quadmax()

Usage:

```
quadmax(A,b [,eq:eqmat] [,gte:gtemat] [,ckbounds:F]), square REAL matrix
  A, REAL vector b with nrows(b) = nrows(A), REAL matrices eqmat and
  gtemat with ncols(eqmat) = ncols(gtemat) = nrows(A) + 1,
```

Keywords: analysis

```
                                  Usage
quadmax(A,b) finds the x that maximizes x'Ax + b'x; if the problem is
unbounded then quadmax() returns an error.  A is a p by p REAL matrix
and b is a p by 1 REAL vector.

quadmax(A,b,eq:eqmat) does the same, except that eqmat is a q by p+1
REAL matrix that specifies q linear constraints on z.  eqmat is the
partitioned matrix [Q y], where Q is q by p and y is q by 1 and the
solution is constrained to satisfy Qx = y.  If the constrained problem
is unbounded, then quadmax() returns an error.  If the constraints
cannot be met, then quadmax() returns NULL.

quadmax(A,b,gte:gtemat) does the same except that gtemat is a g by p+1
REAL matrix specifying linear inequality constraints on the solution.
gtemat is the partitioned matrix [G z], where G is g by p and z is g by
1 and the solution is constrained to satisfy Gx >= z elementwise.  If
the constrained problem is unbounded, then quadmax() returns an error.
If constraints cannot be met, then quadmax() returns NULL.

You can use both eq:eqmat and gte:gtemat to specify both equality and
inequality constraints.

                                  Example
Suppose in a three variable mixture problem you have the equality
constraint that the sum of the x's is 1 and each element of x is at
least .05.  Then you can use quadmax(A,b,eq:eqmat,gte:gtemat)
                                      [ 1 0 0 .05 ]
where eqmat = [1 1 1 1] and gtemat =  [ 0 1 0 .05 ]
                                      [ 0 0 1 .05 ]

                            Keyword ckbounds
```

```
quadmax(A,b [,eq:eqmat] [,gte:gtemat], ckbounds:F) does the same but
there is no test for the solution being unbounded.  This can
substantially decrease the computational time when you know the problem
is bounded.  This is the case, for example, in a problem where the range
of the x's is totally bounded by inequality constraints, or in a problem
where x'Ax is known to have a unique maximum (A has all negative
eigenvalues).  However, ckbounds:F should not be used when you do not
know there is a bounded solution.

                                Algorithm
The algorithm used by quadmax() can be described as intelligent brute
force and will probably be overwhelmed by too many constraints.  When
you are sure the problem has a bounded solution, be sure to use keyword
phrase ckbounds:F.
```

4.25 randsign()

Usage:

```
randsign(diffs [,trials:n]), REAL vector diffs, positive integer n
```

Keywords: permutation test, analysis

```
                                  Usage
randsign(diffs) computes sum(s_i * diffs_i) for all 2^length(diffs)
possible combinations of signs s_i and returns these as a REAL vector.

randsign(diffs,trials:n) samples from the distribution of sum(s_i *
diffs_i) based on n sets of random signs.  This is appropriate when
diffs is long, as 2^length(diffs) grows quickly!!

You can use the results of randsign to compute p-values for the
randomization equivalent of the paired t-test by finding the fraction of
the total differences based on random signs as extreme or more extreme
than the observed total difference for the two groups.

Examples:
  Cmd> x1 <- vector(1,4,2,4,6,3,7) # data set 1

  Cmd> x2 <- vector(3,5,3,6,2,9,8) # data set 2

  Cmd> diffs <- x2-x1

  Cmd> diffs # the differences
  (1)           2           1           1           2          -4
  (6)           6           1

  Cmd> sum(diffs)  # observed total difference
  (1)           9

  Cmd> out <- randsign(diffs)  # all differences with random signs
```

```
  Cmd> stemleaf(out)
      1   -1s|7
      4   -1f|555
      9   -1t|33333
     16   -1*|1111111
     24   -0.|99999999
     32   -0s|77777777
     41   -0f|555555555
     52   -0t|33333333333
     64   -0*|111111111111
     64   +0*|111111111111
     52   +0t|33333333333
     41   +0f|555555555
     32   +0s|77777777
     24   +0.|99999999
     16    1*|1111111
      9    1t|33333
      4    1f|555
      1    1s|7

           1*|1  represents 11  Leaf digit unit = 1

  Cmd> sum(out >= 9)/128  # one sided randomization p-value
  (1)       0.1875
```

```
                            Cross references
See also randt(), randt2().
```

4.26 randt2()

Usage:

```
   randt2(y1, y2 [,trials:n]), REAL vectors y1 and y2, positive integer n
```

Keywords: permutation test, analysis

```
                                  Usage
randt2(y1, y2) computes all possible values of xbar_1 - xbar_2 where
xbar_1 is the mean of a subset of size M = length(y1) from vector(y1,y2)
and xbar_2 is the mean of the length(y2) remaining elements.  y1 and y2
must be REAL vectors.  If there are MISSING values in y1 or y2, they are
immediately deleted and a warning message printed.

The returned value is a vector containing all the differences.

Thus randt2() produces the permutation distribution for a test of
equality of means based on the differences of the means of y1 and y2.

randt2(x1, x2, trials:N) does the same except it randomly samples N
values from the permutation distribution of ybar_1 - ybar_2.
```

```
                          Computing P-values
You can use the results of randt2 to compute p-values for the
randomization equivalent of the two sample t-test by finding the
fraction of the differences as extreme or more extreme than the observed
difference for the two groups.  You may need to be a little careful in
making the comparison because there may not be an exact match of the
observed difference of means and the values returned by randt2 because
of minor differences in rounding .

Examples:
  Cmd> y1 <- vector(6.5,6,7.1,7.1,3.5,6.1) # group 1 data

  Cmd> y2 <- vector(3.9,4.9,2.1,7.7,4.9) # group 2 data

  Cmd> d <- sum(y1)/6 - sum(y2)/5; d  # difference of means
  (1)        1.35

  Cmd> out <- randt2(y1,y2)

  Cmd> length(out) # how many are there?
  (1)         462

  Cmd> sum(round(out - d,12) >= 0)/462 #one-sided permutation P-value
  (1)     0.20996

                           Cross references
See also randsign(), randt().
```

4.27 randt()

Usage:

```
   randt(dvec, m [,trials:n]), REAL vector dvec, positive integer n
```

Keywords: permutation test, analysis

```
                                  Usage
randt(dvec,m) computes xbar_1 - xbar_2 for all combinations with m data
values from dvec in group 1 and the remainder in group 2.  The returned
value is a vector containing all the differences.  dvec must be a REAL
vector with no MISSING values and integer m > 0 with m < length(dvec).

randt(dvec,m,trials:n) samples from the distribution of xbar_1 - xbar_2
using n independent sets of random assignments to the groups.

                          Computing P-values
You can use the results of randt() to compute p-values for the
randomization equivalent of the two sample t-test by finding the
fraction of the differences as extreme or more extreme than the observed
difference for the two groups.  You may need to be a little careful in
```

```
making the comparison because there may not be an exact match of the
observed difference of means and the values returned by randt() because
of minor differences in rounding .

Examples:
  Cmd> y1 <- vector(6.5,6,7.1,7.1,3.5,6.1) # group 1 data

  Cmd> y2 <- vector(3.9,4.9,2.1,7.7,4.9) # group 2 data

  Cmd> d <- sum(y1)/6 - sum(y2)/5; d  # difference of means
  (1)        1.35

  Cmd> out <- randt2(vector(y1,y2), length(y1))

  Cmd> length(out) # how many are there?
  (1)         462

  Cmd> sum(round(out - d,12) >= 0)/462 #one-sided permutation P-value
  (1)     0.20996

  Cmd> stemleaf(out)  # how do they look
     3  -2.|665
    12  -2*|222200000
    37  -1.|888888887766666666665555
    83  -1*|444444433333322222222222221111111110000000000
   151  -0.|99999999998888888888888887777777777777777766666666666555*
  ( 81) -0*|444444444444444444333333333333333333332222222222221111111*
   230  +0*|000000000001111111111111111111112222222222222222223333333*
   154  +0.|555555555555555566666666666666666667777777778888888889999*
    86   1*|00000000000011111111111122222223333333333444444444
    36   1.|5555566666677788899999
    14   2*|0000001334444
     1   2.|8

                             Cross references
See also randsign(), randt2().
```

4.28 reml()

Usage:

```
reml(Model,random:Randomvars,Z:Zmatrices,[,restrict:F,nonhier:T,marg:T,
    maxiter:k,usemle:T,retV:T,tolerance:x])
```

Keywords: analysis

```
                                    Usage
reml(Model,random:Randomvars) performs a restricted maximum likelihood
analysis for the model given in CHARACTER scalar Model.  Randomvars
is a CHARACTER vector specifying the names of factors in the model which
are random.  Randomvars can also be REAL with integer elements
specifying the index of a factor in the model.
```

```
reml(Model,Z:Zmatrices) performs a restricted maximum likelihood
analysis for the model given in CHARACTER scalar Model plus additional
random terms specified via Z.  Zmatrices is a CHARACTER vector
specifying the names of matrices, each of which has the same number of
rows as the response in the model is long.  If Z is vector("Z1","Z2"),
then random terms of the form Z1*gamma1 + Z2*gamma2 are added to the
model, where gamma1 and gamma2 are iid normal vectors with variances
phi1 and phi2 respectively.

At least one of random: or Z: should be used, and if there are no
random factors, Randomvars should be NULL.

The return value of reml() is a structure with the following components:
     theta:   estimates of the fixed effects
       phi:   estimates of the variance components
  thetavar:   variance matrix of the fixed effects
    phivar:   variance matrix of the variance components
     phidf:   equivalent degrees of freedom for the variance components
     gamma:   estimates (predictions) of random effects
  gammavar:   variances of predictions of random effects
         L:   REML log likelihood
         V:   optional component giving the estimate covariance of the
              data

Any variates in the model must be fixed effects.

                                Assumptions
reml() assumes that if a factor first appears in an interaction, then
that factor is nested in the other terms of the interaction.  For
example, if the first appearance of factor c is in the term a.b.c, then
c is assumed nested in the a.b combinations.  This nesting is assumed in
the remainder of the model.  That is, continuing the example, if there
is a later term c.d, it will be interpreted as a.b.c.d even though
a.b.c.d is not specifically in the model.

reml() works for both balanced and unbalanced data.

                      Keywords restrict and nonhier
reml(Model,Randomvars,restrict:F) performs the REML analysis assuming
no marginal restrictions on the random effects in the model.

reml(Model,Randomvars,nonhier:T) performs the REML analysis for an
analysis of variance that does not enforce the usual MacAnova hierarchy
assumptions.

That is, for example, model "y=a+b+c+a.b.c" does not imply
that the two-way interaction degrees of freedom are part of the "a.b.c"
term.  You cannot use anova() to compute such an analysis although it
can be done (if you know how) using swp().

                  Keywords usemle, tolerance and maxiter
reml(Model,Randomvars,usemle:T) performs a maximum likelihood analysis
```

```
instead of the REML analysis.

reml(Model,Randomvars,tolerance:value) uses value as a tolerance for
determining singularity and convergence (default is 1e-10).

reml(Model,Randomvars,maxiter:value) uses value as the maximum number of
iterations in the fitting process (default is 60).

reml(Model,random:vars,retV:T) returns the estimated covariance
of the data in a component named V.

                          Cross references
See also mixed().
```

4.29 rscanon()

Usage:

```
rscanon(y,x1,x2,...,xk [,block:var1,block:var2,...]), REAL vectors y,
  x1, ...,xk, factors var1, var2, ...; all should have the same number
  of rows.
```

Keywords: analysis

```
                                Usage
rscanon(y,x1,x2,...,xk) performs the canonical analysis for the
quadratic response surface model with response y and predictors x1,
... , xk.  y and x1 through xk must be REAL vectors of the same length.

The result is a structure with components 'b0', 'b', 'B', 'x0', 'y0',
'H', and 'lambda', giving the intercept, linear coefficients, the
quadratic/ cross product coefficient matrix, the stationary point, the
predicted response at the stationary point, the matrix of canonical
directions, and the eigenvalues, respectively.

If the design was blocked, you can specify the blocking factors using
one or more keyword phrases of the form 'block:var', where var is a
factor.  For example, if the design was blocked by factors date and
analyst, you might use
  Cmd> rscanon(yield,time,temperature,block:date,block:analyst).
The output is the same as before, but is block adjusted.

Examples:
  Cmd> #data from example 16-2 of Montgomery

  Cmd> x1 <- vector(-1,-1,1,1,0,0,0,0,0,1.414,-1.414,0,0)

  Cmd> x2 <- vector(-1,1,-1,1,0,0,0,0,0,0,0,1.414,-1.414)

  Cmd> y <- vector(76.5,77.0,78.0,79.5,79.9,80.3,80.0,79.7,\
  79.8,78.4,75.6,78.5,77.0)
```

```
Cmd> rscanon(y,x1,x2)
component: b0
(1)        79.94
component: b
(1)      0.99505       0.5152
component: B
(1,1)      -1.3764        0.125
(2,1)        0.125      -1.0013
component: x0
(1)      0.38923      0.30585
component: y0
(1)       80.212
component: H
(1,1)      0.28972      0.95711
(2,1)      0.95711     -0.28972
component: lambda
(1)      -0.9635      -1.4143
```

4.30 sidebyside()

Usage:

```
sidebyside([termlaby:y,labels:c,rescale:tf,showconst:tf,boxcut:int])
```

Keywords: plots

```
                                  Usage
sidebyside() produces a side-by-side plot of the effects and residuals
of the current model; there must be an active model.  A side-by-side
plot plots the effects for each term against the term number, showing
the relative sizes of the effects.  When there are many effects or
residuals, a boxplot is made instead of showing individual effects.
There are no required arguments, but the following arguments alter
the plot; in addition, any graphics arguments are passed through.

                                 Keywords
Optional keyword phrase arguments are
  termlaby:real      Specify y-value for term labels. This can be a
                     single value or a vector of length equal to number
                     of terms.
  labels:charvector  Specify your own term labels.
  rescale:logic      Should effects be divided by their standard errors.
                     Default is F.  This uses the standard errors as
                     reported by secoefs() and may not be correct for
                     all mixed models (secoefs() depends on terms
                     labeled ERRORX).  Residuals are divided by root
                     MSE.
  showconst:logic    Should the coefficient of the CONSTANT be shown?
                     Default is F.
  boxcut:integer     Cutoff for using a boxplot for a term instead of
                     plotting individual effects.  Default is 20.
```

```
Example:
  Cmd> y <- vector(9,13,12,43,48,57,60,65,70,77,70,91,\
       15,13,20,66,58,73,75,78,90,97,108,99)
  Cmd> acid<-factor(rep(run(2),rep(12,2)))
  Cmd> style<-factor(rep(rep(run(4),rep(3,4)),2))
  Cmd> anova("y=acid*style",silent:T)
  Cmd> sidebyside() #default plot
  Cmd> # specify term locations and labels
  Cmd> sidebyside(termlaby:vector(-50,-40,-30,-20),\
       labels:vector("a","b","c","d"),boxcut:30)
  Cmd> # show the constant and rescale
  Cmd> sidebyside(dumb:T,showconst:T,rescale:T)
```

4.31 stdordlabels()

Usage:

```
stdordlabels(k:count or letters:word) count an integer and word
a character scalar
```

Keywords: factorial

```
stdordlabels(k:count) produces a character vector of term labels
for count factors in standard order.  For example, stdordlabels(k:3)
produces "A","B","AB","C","AC","BC","ABC".

stdordlabels(letters:word) splits character scalar word up into
its component letters and uses these letters as a factor labels.

No more than 15 factors are allowed.
```

4.32 typeIIIss()

Usage:

```
typeIIIss(model), obsolete, use anova(model,marginal:T)
```

Keywords:

```
typeIIIss(model) is obsolete, use anova(model,marginal:T) instead.
```

4.33 varcomp()

Usage:

```
varcomp(model,randomvars [,marg:T] [,restrict:F] [,nonhier:T]),
  CHARACTER scalar model, CHARACTER vector randomvars
varcomp(emsResult), emsResult a structure returned by ems() with keep:T.
```

Keywords: anova, analysis, random effects, factorial

```
                                  Usage
varcomp(model, randomvars) computes the "ANOVA" estimates of the
variances of random effects in mixed effects analysis of variance as
well as estimates of their standard errors.  model is a CHARACTER scalar
or a quoted string specifying an ANOVA model and randomvars is a
CHARACTER vector of the names of some or all of the factors in the
model.  You can also use keyword phrases marg:T, restrict:F and
nonhier:T exactly as for macro ems().

varcomp(emsResult), where emsResult is computed as ems(model,
randomvars, keep:T) does the same.

The estimates are linear combinations of mean squares for random
effects.  They are unbiased but may be negative.

The value of varcomp() is a matrix with one row for each random term and
two columns giving the estimated variance component and its standard
error.

                               Keyword marg
varcomp(model, randomvars, marg:T) and varcomp(emsResult, marg:T) do the
same but use linear combinations of "marginal" EMS, that is the EMS for
each term is computed after adjusting for all other terms in the model.

varcomp() assumes that the EMS for random terms have no contributions
from fixed factors.  This is true for balanced data and may be
guaranteed in general by using marg:T.

Example:
Three populations, all crosses between 4 males and 4 females in each
population with six offspring from each mating randomly assigned to
three environments.  Male and female are random.  First the simple
ANOVA.

  Cmd> anova("y=(pop+m.pop+f.pop+m.f.pop)*env")
  Model used is y=(pop+m.pop+f.pop+m.f.pop)*env
                  DF           SS           MS
  CONSTANT         1       5.4299       5.4299
  pop              2       2091.4       1045.7
  pop.m            9        112.5         12.5
  pop.f            9       370.02       41.113
  pop.m.f         27       56.774       2.1027
  env              2       206.15       103.08
  pop.env          4      0.16527     0.041316
  pop.m.env       18       3.4185      0.18992
  pop.f.env       18       8.2354      0.45752
  pop.m.f.env     54       17.117      0.31698
  ERROR1         144       30.448      0.21144

Now compute the expected mean squares, and keep the ems() output.
```

```
  Cmd> emsstuff<-ems("y=(pop+m.pop+f.pop+m.f.pop)*env",vector("m","f"),
     keep:T,print:T)
  EMS(CONSTANT) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f) + 24V(pop.m) +
     288Q(CONSTANT)
  EMS(pop) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f) + 24V(pop.m) + 96Q(pop)
  EMS(pop.m) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.m)
  EMS(pop.f) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f)
  EMS(pop.m.f) = V(ERROR1) + 6V(pop.m.f)
  EMS(env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env) +
     8V(pop.m.env) + 96Q(env)
  EMS(pop.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env) +
     8V(pop.m.env) + 32Q(pop.env)
  EMS(pop.m.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.m.env)
  EMS(pop.f.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env)
  EMS(pop.m.f.env) = V(ERROR1) + 2V(pop.m.f.env)
  EMS(ERROR1) = V(ERROR1)

From the EMS, we see than (MS(pop.m.f.env)-MS(ERROR1))/2 is an unbiased
estimate of V(m.f.env); here, we have (.31698 - .21144)/2 = .05277.
Similarly, (MS(pop.f.env)-MS(pop.m.f.env))/8 is an unbiased estimate of
V(f.env); here we have (.45752 - .31698)/8 = .01757.  varcomp()
automates these calculations, as well as providing the standard error.

  Cmd> varcomp(emsstuff)
                Estimate           SE
  pop.m          0.43323      0.24668
  pop.f           1.6254      0.80788
  pop.m.f        0.31521     0.095472
  pop.m.env     -0.015883     0.010989
  pop.f.env      0.017568     0.020532
  pop.m.f.env    0.052766     0.032948
  ERROR1          0.21144     0.024919

Note that variance component estimates can be negative; varcomp() does
not truncate the estimates at 0.  We would get the same output from the
following command.

   Cmd> varcomp("y=(pop+m.pop+f.pop+m.f.pop)*env",vector("m","f"))

                            Cross references
See also ems(), mixed().
```

4.34 yatesplot()

Usage:

```
   yatesplot(data,[,fullnorm:T] [,letters:word]),
     data real with length a power of 2, word a character scalar of
     letters to label factors
```

Keywords: analysis, factorial, plots

```
yatesplot(y) computes the factorial effects for data y from a
two series design stored in standard order.  It then makes a
half-normal plot of the effects labelling the terms with labels
A, B, AB, C, and so on.

yatesplot(y,fullnorm:T) does the same, except that it normal scores
plot instead of a half-normal plot.

yatesplot(y[,fullnorm:T],letters:word) where word is a character
scalar such as "DEFG" will make the full- or half-normal plot, but
will label the terms with letters taken from the word, for example,
D, E, DE, F, and so on.  The number of letters in the word must
match the number of factors in y.

If there are more than 10 factors, the effects are not labelled.
```

Chapter 5

Graphics Macros Help File

This Chapter contains help for the set of macros related to graphics that are distributed with MacAnova in the file Graphics.mac.txt. The material here is a reformatting of the help in file Graphics.mac.txt.

5.1 bargraph()

Usage:

```
bargraph(edges,y [,save:T] [,draw:T] [,graphics keyword phrases]), REAL
  vectors edges and y, length(edges) <= 2 or = length(y) + 1
bargraph(vector(firstedge [,secondedge]), y, [,save:T] [,draw:T] ...),
  firstedge < secondedge REAL scalars, default for secondedge =
  firstedge + 1
bargraph(x,y,widths [,save:T] [,draw:T] ...), REAL vectors x, y with
  same length and width a positive vector of same length as x or a
  positive scalar, interpreted as rep(widths,nrows(x)
```

Keywords: bar graphs

```
                                  Usage
bargraph(edges, y [,graphics keyword phrases]) draws a bar graph with
touching bars and bar heights y.  Arguments edges and y are REAL vectors
with edges[j+1] - edges[j] > 0 and length(edges) = length(y) + 1.  Bar j
has height y[j] and boundaries edges[j] and edges[j+1].

The graphics keyword may include 'xlab', 'ylab', 'title', keywords
having to do with tick marks and so on.

bargraph(vector(edge1, edge2), y [,graphics keyword phrases]), where
edge1 < edge2 are REAL scalars does the same except the bar edges are
edge1, edge1 + width, ...,edge1 + length(y)*width, where width = edge1 -
edge1.

bargraph(edge1, y [, ...]) is the same a bargraph(vector(edge1,edge1+1),
y, ...).

bargraph(x, y, w [,...]) draws a bar graph with bars centered at x,
```

```
with heights y and widths w.  x, y and w must be REAL vectors with no
MISSING values and nrows(x) = nrows(y), and w vector of the same length
or a scalar equivalent to rep(w,nrows(x)).

                              Keyword keep
bargraph(edges,y, keep:T [,draw:T] [,graphics keyword phrases]) or
bargraph(x,y,w, keep:T [,draw:T] [,graphics keyword phrases])
returns
  structure(x:xvals, y:yvals [,graphics keyword phrases], lines:T)
where xvals and yvals are REAL vectors such that lineplot(xvals,yvals)
would draw the bars.  With draw:T, the graph also is drawn.

The structure can be assigned to GRAPHWINDOWS[j] to draw the bar graph
in window j or used with panelplot() to draw the bar graph in a panel
graph (see topics panelplot() and 'panel_graphs').

                            Cross references
See also hist(), lineplot(), 'graph_keys'.
```

5.2 boxplot5num()

Usage:

```
boxplot5num(x [,names:Names][,excludeM:T] [,keep:T] \
  [,graphics keyword phrases]), x a REAL vector or a structure with
  REAL vector components, Names a CHARACTER scalar or vector
```

Keywords: distribution graphs

```
boxplot5num(x), where x is a REAL vector or a structure whose components
are REAL vectors, draws a simplified boxplot of a vector or side-by-side
simplified boxplots of the components of x.  These reflect only the 5
number summary (minimum, lower quartile, median, upper quartile and
maximum) and display no information on outliers.

The upper and lower quartiles are the medians of the upper and lower
halves of the data as computed by describe().  When the sample size is
odd, the median is included in both halves unless the value of option
'excludeM' is True.  See setoptions() and keyword 'excludeM' below.

The plot drawn is based on what is presented in David S. Moore, The
Basic Practice of Statistics.  To replicate plots in the book, you
should set option 'excludeM' to True or use keyword phrase 'excludeM:T'
as an argument.  See below.

boxplot5num(x,names:Names), where Names is a CHARACTER scalar or
vector, does the same, except Names is used to label the boxes.  If
Names is a scalar, say "School ", the plots will be labelled "School 1",
"School 2",... .  If Names is vector, than x must be a structure with
ncomps(x) = length(Names).

boxplot5num(x,excludeM:T [,names:Names]), does the same except the
```

```
quartiles are computed as the medians of the lower and upper halves
*excluding* the median.  This matches the definition in Moore's book.

boxplot5num(x, keep:T [,excludeM:T, names:Names]) does the same, but
also returns a vector or structure containing the 5 number summaries
(Min, Q1, Median, Q3, Max) for each box plot drawn.

You can also use most of the usual graphics keywords such as 'title',
'ylab', 'show', and 'window'.

                            Cross references
See also boxplot() and vboxplot() which draw more elaborate boxplots.
```

5.3 colplot()

Usage:

```
colplot(x [, graphics keyword phrases]), x a REAL matrix
```

Keywords: line graphs, interaction graphs

```
colplot(x) makes an "interaction" plot of the data in the REAL matrix x.
The plotting positions are the row numbers and the values in x.  Points
within each column are joined by lines.  Any keywords useable in chplot
may follow x.

When option 'dumbplot' has been set False (see options), the plot will
be a low resolution plot unless 'dumb:F' is an argument.

See topic 'graph_keys', 'graph_border' and 'graph_ticks' for information
on other keywords that can be used with colplot().

Example:
  Cmd> colplot(run(20)^(.2*run(5)'),xlab:"X",\
        title:"X^.2, X^.4, X^.6, X^.8, X")

colplot() is implemented as a pre-defined macro.

                             Cross reference
See also topic rowplot().
```

5.4 contour()

Usage:

```
contour(x,y,vals,level [,checkargs] [,keyword phrases]), REAL vectors x,
  y, REAL scalar level, all with no MISSING, REAL matrix vals, optional
  LOGICAL scalar checkargs
```

Keywords: contour graphs

```
contour(x,y,vals,level) determines coordinates of a polygonal curve
(curve made up of line segments) that approximates a contour of constant
height of a surface whose height at (x[i], y[i]) is vals[i,j].

x and y must be REAL vectors of unique non-MISSING values.  vals must
be a REAL nrows(x) by nrows(y) matrix, which may have MISSING elements.
level must be a non-MISSING REAL scalar, preferably between the extreme
values in vals.  Most commonly, vals[i,j] = F(x[i],y[j]) for some
function F(x,y) of two variables.  When the surface is not defined or
is infinite at (x[i],y[j]), vals[i,j] should be MISSING.

contour(x,y,vals,levels,T) does the same, except that the arguments are
not checked for correctness.  In particular, x and y are assumed to be
increasing order, which is not ordinarily required.  This usage is
designed for use in macro contourplot() which has already checked
arguments and reordered x and y before calling contour().

The value returned is structure(x:xc, y:yc, level:level).  When level
is outside the range of vals, xc and yc are both NULL.  Otherwise xc
and yc are REAL vectors of the same length. The points (xc[k],yc[k]), k
= 1,..., nrows(xc), are either intersections of the contour with
gridlines, or (MISSING,MISSING).  When a contour consists of two or
more disjoint curves, they are separated by MISSING in both xc and yc.

contour(x,y,vals,level, graphics keyword phrases) does the same except
the result is structure(x:xc, y:yc, level:level, graphics keyword
phrases) in which all the keyword phrases in the argument are appended
to the result.

Example:
  Cmd> curve <- contour(x,y,vals,level,title:"Sample contour curve")

  Cmd> if(!isnull(curve$x)){
         lineplot(keys:curve)
       }

During execution of contour(), x, y and vals are copied to invisible
arrays __X__, __Y__ and __F__, respectively and macro _Follow() traces
out the contour, keeping status information in invisible matrix __MET__.
__X__, __Y__, __F__ and __MET__ are deleted before contour() is
finished.

                                Credits
Contour and associated macros are based on Fortran routines by Dan
LaLiberte, implementing methods in Crane, C.M.(1972), Contour plotting
algorithm, 'The Computer Journal', Vol. 15, pp. 382-384 and Cottafava,
G., Andle Moli, G. (1969). Automatic Contour Map, 'Comm. ACM', Vol. 12,
pp. 386-391.

                            Cross reference
See also contourplot().
```

5.5 contourplot()

Usage:

```
contourplot(x,y,vals,levels [,label:T] [,linefrom:T] [,draw:F]\
  [,save:T] [,graphics keywords]), REAL vectors x, y, levels, with
  no MISSING values, REAL matrix vals
```

Keywords: contour graphs

contourplot(x,y,vals,levels [,graphics keywords]) draws unlabelled contours of a surface whose height z is known at points on a rectangular grid defined by REAL vectors x and y. The desired contour levels are defined by REAL vector levels. The height at (x[i],z[j]) is z = vals[i,j].

The values of x and y must be distinct and non-MISSING.

vals must be a nrows(x) by nrows(y) REAL matrix and may have MISSING elements. Most commonly, vals[i,j] = F(x[i],y[j]) for some function F(x,y) of two variables. If the surface is not defined or is infinite at (x[i],y[j]), vals[i,j] should be MISSING.

You can use most of the usual graphics keywords such as 'xlab', 'ylab', 'title', 'add' and 'show'. In particular, you can use 'linetype' to control the type of line drawn. See topic 'graph_keys'.

contourplot(x,y,vals,levels, label:T [,graphics keywords]) does the same except that you use the mouse to position labels for each contour line that was actually drawn (some levels may be outside the minimum and maximum values in vals). A simple algorithm is used to find the contour line whose level is nearest the interpolated value at the point you click. Then this level is printed at that point. You can end labelling by pressing 'q' when the crosshairs are in the graphics window.

contourplot(x,y,vals,levels, linefrom:T [,graphics keywords]) is another way to label points. You use the mouse to draw a line starting at the contour and ending where the label is then printed.

contour uses macro findcontours() to locate and label contours. You can use findcontours() directly to label contours in a contour plot that was previously drawn.

Keyword save

result <- contourplot(x,y,vals,levels,save:T [,other keywords]) sets result to structure(Contour_1:comp1,Contour_2:comp2 ...). Component I of result is structure(x:xvals, y:yvals,level:levels[I]). When level[I] is outside the range of values, xvals and yvals are NULL. Otherwise they are the x- and y-values defining the intersections of the contour with gridlines.

You could use this returned structure to add the same contour curves to another graph. The following example might be appropriate when the contours were those of an estimated bivariate density function based on a bivariate sample hconcat(xvals, yvals), where xvals and yvals are REAL

```
vectors.

  Cmd> chplot(xvals, yvals, symbols:"\7",show:F) #scatter plot of sample

  Cmd> for(i,1,levels){
          @comp <- result[i]
          if (!isnull(@comp$x)){
            lineplot(keys:strconcat(@comp,add:T,show:F))
          }
       }

  Cmd> showplot(xmin:?,ymin:?,xmax:?,ymax:?) # display the plot

                              Keyword draw
In place of 'save:T' you could use 'draw:F'.  With both 'save:T' and
'draw:T' the contours are both drawn and returned in a structure.

contourplot() requires macros contour(), findcontour() and _Follow().
They are read in automatically if necessary and possible.

                            Cross references
See also contour() and findcontour().
```

5.6 ellipse()

Usage:

```
   ellipse(K, Q [,x0] [,npoints:m] [method:j] [,draw:T] \
     [,graphics keyword phrases]), REAL scalar K > 0, 2 by 2
     positive definite symmetric REAL matrix Q, REAL vector x0 or length
     2, integer j, 1 <= j <= 3
```

Keywords: shapes, line graphs

```
You can use ellipse() to compute and optionally draw an ellipse with
shape defined by a specified positive definite matrix and centered at a
specified point

ellipse(K, Q [,x0] [,graphics keywords]) computes xvals and yvals, the
x- and y-coordinates of points on the ellipse defined by the equation
        (x - x0)' %*% solve(Q) %*% (x - x0) = K^2
The value returned is structure(x:xvals,y:yvals [,graphics keywords]).

K > 0 must be a REAL scalar and Q must be a 2 by 2 REAL positive
definite symmetric matrix.  If x0 is an argument, it must be a REAL
vector of length 2.  Otherwise, rep(0,2) is used for x0.

The ellipse can be plotted by
  Cmd> result <- ellipse(K, Q [,x0] [,graphics keywords])

  Cmd> lineplot(keys:result)
```

```
ellipse(K, Q [,x0], draw:T [,graphics keywords]) draws the ellipse
directly and doesn't return the coordinates as a value.  If the ellipse
is to be added to an existing graph, include add:T as an argument.

                              Keyword npoints
ellipse(K, Q [,x0], npoints:m ...) computes m + 1 points on the
ellipse, with the first and last being identical.  The default value
for m is 200.

                              Keyword method
There are several ways to select points on an ellipse.  Macro ellipse()
allows you to use any of three methods.

ellipse(K, Q [,x0], method:k ...), where 1 <= k <= 3, does the same,
except the way xvals and yvals are computed depends on k (default is k
= 3).

All three methods compute unit vectors u(theta) for m+1 equally
spaced values of theta from 0 to 360 degrees, where
  u(theta) = vector(cos(theta), sin(theta))

Method 1 (Pure polar coordinates):
  x(theta) = x0 + K * u(theta)/r1(theta),
    r1(theta) = sqrt(u(theta)' %*% solve(Q) %*% u(theta))
Method 2
  x(theta) = x0 + K * Q %*% u(theta)/r2(theta), where
    r2(theta) = sqrt(u(theta)' %*% Q %*% u(theta))
Method 3 (default)
  x(theta) = x0 + K*cholesky(Q)' %*% u(theta)

The default method (3) seems to do the best job, but you may want to
try one of the others to see if they produce a better looking ellipse.

                              Cross references
See also 'graph_keys', lineplot(), cholesky(), solve(), 'matrices'
```

5.7 findcontour()

Usage:

```
   info <- findcontour(x,y,vals,levels [,linefrom:T]), REAL vectors x, y,
     and levels, all with no MISSING values, REAL matrix vals
```

Keywords: contour graphs

```
After drawing a contour plot of a surface by contourplot() or plotting
the results from repeated calls to contour(), you can use findcontour()
to identify contours with the mouse so that they can be labeled with the
contour level.

info <- findcontour(x,y,vals,levels) attempts to identify the contour
line (level curve) nearest the point (x0,y0) you select using the mouse.
```

```
x and y are non-MISSING REAL vectors defining a grid of points and
levels is a non-MISSING vector defining the levels of contours in the
plots.

vals is a nrows(x) by nrows(y) REAL matrix, possibly with MISSING
values, of surface heights with vals[i,j] = height of the surface at
(x[i],y[i]).  If the surface is not defined or is infinite at
(x[i],y[j]), vals[i,j] should be MISSING.

Normally, findcontour(x,y,vals,levels) is used after contourplot(x,y,
vals,levels).

info is set to structure(x:x0, y:y0, value:val0, level:level0), where
(x0, y0) is the point selected, val0 is the interpolated height of the
surface at (x0,y0), and level0 is the element of levels closest to
val0.  If the point selected is close to a contour, level0 should be
the level of the contour.  If the point selected is in the rectangle
defined by x[i], x[i+1] and y[j], y[j+1] and v[i,j], v[i+1,j], v[i,j+1]
and  v[i+1,j+1] are all MISSING, val0 and level0 will both be MISSING.

You can label a contour by

  Cmd> info <- findcontour(x,y,vals,levels)

  Cmd> addstrings(info$x,info$y,paste(info$level,format:".3f"))

info <- findcontour(x,y,vals,levels,linefrom:T) does the same except you
use Mouse() to define a line to be drawn from a contour to a label.
First select a point (x0,y0) close to the contour and then the point
(x1,y1) where the label should be.

With linefrom:T, info is set to structure(x:vector(x0,x1),
y:vector(y0,y1), value:val0, level:level0) where val0 is the
interpolated surface height at (x0,y0) and level0 is the contour level
nearest to val0.

You can label a contour by

  Cmd> info <- findcontour(x,y,vals,levels,linefrom:T)

  Cmd> addlines(info$x,info$y, show:F)

  Cmd> addstrings(info$x[2],info$y[2],paste(info$level,format:".3f")

You can avoid direct use of findcontour() by using 'label:T' on
contourplot().  This results in findcontour() being called once for
every contour actually drawn, omitting contours whose levels are outside
the range of val.

                          Cross references
See also contourplot(), contour(), Mouse().
```

5.8 graphicshelp()

Usage:

```
graphicshelp(topic1 [, topic2 ...] [,usage:T])
graphicshelp(index:T)
```

Keywords: general

```
graphicshelp(topicname) prints help on a topic related to file
graphics.mac.  Usually topicname is the name of a macro in the file.

When quoted, topicname may contain "wildcard" characters "*" and "?".
You can also use help keyword 'key'.  See help() for details.

graphicshelp(topicname1, topicname2, ...) prints help on more than one
topic.

graphicshelp(topicname1 [, topicname2 ...], usage:T) prints just a brief
summary of usage for the each topic.
```

5.9 hist()

Usage:

```
hist(x [, nbars] [,keyword phrases]), REAL vector x, integer nbars >= 2
hist(x, vector(anchor,width) [,keyword phrases]), anchor REAL scalar,
  width > 0 scalar
hist(x, edges [,keyword phrases]), edges REAL vector with increasing
  elements
Keyword phrases are relfreq:T, freq:T, leftendin:T, outsideok:T, draw:T,
  save:T plus most graphics keywords
```

Keywords: distribution graphs, bar graphs

```
hist(x, nbars) draws a histogram of the data in REAL vector x using with
nbars equal-width bars which include all data.  The bar edges are not
"neat".  For example, 1,1.5,2,2.5,3.0, ... are "neat", 2.71,3.82, 4.93,
6.04, ... are not "neat".

The default bar heights are in the so called "density scale" with height
= (M/N)/W, where M is the number of values in a bar with width W and N
is the number of non-MISSING values in x.  This choice makes the total
area of the bars = 1.  You can use keywords 'freq' and 'relfreq' (see
below) to get other bar heights.

A value x is included in bar i when L[i] < x <= R[i], where L and R are
vectors of the left and right edges of the bars.

hist(x, nbars, leftendin:T) does the same, except a value x is included
in bar i when L[i] <= x < R[i].  'leftendin:T' can be used with any
variant of hist() arguments and other keywords.
```

```
hist(x, nbars, freq:T) and hist(x, nbars, relfreq:T) do the same except
that bar heights are frequencies (M) or relative frequencies (M/N) with
no adjustment for bar width.  'freq:T' and 'relfreq:T' can be used with
any variant of hist() arguments.

hist(x [,keyword phrases]) does the same using floor(log2(N)) + 1 bars.

hist(x, vector(anchor, width) [,keyword phrases]) does the same, except
the edges of the bars are of the form anchor + j*width, with the lowest
and highest bar edges chosen to include all the data.

hist(x, Edges [,keyword phrases]) draws a histogram whose bar boundaries
are the elements of REAL vector Edges with length(Edges) > 2 and
satisfying Edges[i] < Edges[i+1].  The number of bars is nbars =
length(Edges) - 1.  A warning message is printed when bar widths are
not all equal and 'relfreq:T' or 'freq:T' is an argument.

                            Keyword outsideok
It is normally an error when extreme data values are outside the bars
defined by Edges.  Without 'leftendin:T' this occurs when min(x) <=
Edges[1] or max(x) > Edges[nbars+1].  With 'leftendin:T', this occurs
when min(x) < Edges[1] or max(x) >= Edges[nbars+1].

hist(x, Edges, outsideok:T) does the same, except it is not an error
when some extreme values are outside the bars defined by Edges.  When
values are outside, a warning message is printed.

All of the usual plotting related keywords, including 'dumb', 'xlab',
'ylab', and 'title', may be used with hist().  See also topics 'graphs',
'graph_keys', 'graph_borders' and 'graph_files'.

                               Keyword save
result <- hist(x [, bar info] [,hist keywords] [,graphics keywords],
save:T) returns structure(x:xvals, y:yvals, line:T [,graphics keywords])
instead of drawing the histogram.  REAL vectors xvals and yvals are
such that lineplot(xvals,yvals) draws the histogram.  You can force the
histogram to be drawn by also including 'draw:T' as an argument.

                               Keyword keys
An alternate way to specify keyword values is to create a structure
keyValues of keyword values and use 'keys:keyValues' as the only keyword
phrase argument.  For example

  Cmd> keyvals <- structure(xlab:"Bone length",relfreq:T,\
        title:"Bone histogram", ylab:"Relative frequency",save:T)

  Cmd> stuff <- hist(bones,vector(0,.25),keys:keyvals)

does the same as
  Cmd> stuff <- hist(bones,vector(0,.25),xlab:"Bone length",relfreq:T,\
        title:"Bone histogram", ylab:"Relative frequency",save:T)

                            Cross reference
```

```
See also topic panelhist().
```

5.10 news

Usage:

Keywords: general

```
011109 Fixed bugs in contour() and _Follow() which caused contour()
to fail to find certain contours.

010327 Fixed bug in boxplot5num() and made the value of option
'excludeM' the default for keyword 'excludeM'.

Added keyword phrase 'printname:F' to all invocations of getmacros.

010216 Enhancements to piechart: p can be row or column vector; change
in default labels

010125 New keyword phrases on macro boxplot5num():
 names:Names provides labels for boxplots to be put in margin
 keep:T returns 5 number summaries after drawing boxplots.
 excludeM:F includes median in upper and lower halves when computing
  quartiles.
```

5.11 panelhist()

Usage:

```
   panelhist(x, edges or nbars, pos:k or pos:vector(r,c), hstrips:h,\
     vstrips:v [,label:lab] [,hist keyword phrases]\
     [,graphics keyword phrases]), REAL vectors x and edges, positive
     integers k, r, c, h and v, CHARACTER scalar lab
```

Keywords: panel graphs, bar graphs

```
panelhist(x, nbars, pos:position, hstrips:h, vstrips:v) draws a
histogram with nbars equal width bars in a pane of an h by v panel
graph, an rectangular array of small graphs, where h and v are positive
integers.  See topic 'panel_graphs'.

When position is of the form vector(r, c) where 1 <= r <= h and 1 <= c
<= v are integers, the histogram is drawn in row r and column c.  When
position = n, with positive integer n <= h*v, the histogram is drawn in
pane n, counting across rows starting in row 1.

panelhist(x, vector(anchor,w), pos:position, hstrips:h, vstrips:v
[,label:lab]) does the same, except the bars all have width w, with the
```

```
edges all of the form anchor + j*w, where j is an integer.  For
example, vector(-.5, 1) specifies unit width bars centered on integers.

panelhist(x, edges, pos:position, hstrips:h, vstrips:v [,label:lab])
does the same, except the edges of the bars are taken from the elements
of REAL vector edges.  They must be in strictly increasing order.

With all usages, you can include keyword phrases 'freq:T', 'relfreq:T',
'leftendin:T' and 'outsideok:T' as used with macro hist().  See topic
hist()

                              Keyword label
With all usages you can include 'label:lab', where lab is a CHARACTER
scalar.  After the histogram is drawn you then use the mouse to specify
a place where lab should be added to the graph.

                           Graphics keywords
You can use graphics keywords 'window', 'show', 'title', 'xlab', 'ylab',
'add', 'yaxis', 'xmin', 'xmax', ymin', and 'ymax' plus keywords having
to do with files.  See topic 'graph_keys'.  With show:T, label:lab is
ignored.

You should use 'add:T' when drawing in a pane in an existing panel
graph.

You cannot use graphics keywords 'lines', 'linetype', 'impulse',
'thickness', 'xaxis', 'logx', and 'logy' and keywords having to do with
tick marks.

Example:
  Cmd> irisdata <- matread("macanova.dat","irisdata") #Fisher data

  Cmd> # column 1 is variety number; columsn 2 - 5 are data

  Cmd> colnames <- vector("SepLen","SepWid","PetLen","PetWid")

  Cmd> plotmatrix(irisdata[,-1],name:"Iris Data",diaglabs:F, show:F,\
    symbols:vector("\21","\22","\23")[irisdata[,1]], labels:colnames)

  Cmd> for(i,1,4) { # add 16 bar histograms to diagonal
    panelhist(irisdata[,i+1],16,pos:rep(i,2),hstrips:4,vstrips:4,\
    add:T,show:i==4) }

This uses plotmatrix() to create a scatter plot matrix of the Fisher
iris data, and then adds histograms of each variable to the diagonal,
displaying the plot only when the last histogram has been drawn.

                           Cross references
See also plotmatrix(), panelplot(), hist(), 'graph_keys'.
```

5.12 panelplot()

Usage:

```
panelplot(x,y, pos:k or pos:vector(i,j), hstrips:h, vstrips:v,\
  [label:structure(labx,laby)] [,graphics keyword phrases]), k, i, j,
  h, v > 0 integers, labx, laby CHARACTER scalars
panelplot(x,y, keys:structure(keyword phrases)
```

Keywords: panel graphs

```
                                Usage
panelplot(x,y,pos:position,hstrips:h,vstrips:v [,graphics keywords])
plots REAL vector y against REAL vector x in a pane of a "panel graph",
an h by v array of small graphs, where h and v are positive integers.
See topic 'panel_graphs' for information about such graphs.

When position is of the form vector(r, c) where 1 <= r <= h and 1 <= c
<= v are integers, the graph is drawn in the pane in row (horizontal
strip) r and column (vertical strip) c.  When position = n, with
integer n satisfying 1 <= n <= h*v, the graph is drawn in pane n,
counting across rows starting in row 1 and row 1.

When nrows(x) <= 2, x is "expanded" to a vector of equally spaced
spaced as happens with plot(), lineplot() and most other graphing
commands.  Otherwise, nrows(x) = nrows(y) is required.

panelplot() differs from plot() and most other other graphic commands,
in that y can have only one column.  Another difference is that the
graph has no tick marks or other information about actual values except
possibly the x- or y-axis.

                           Graphics keywords
Legal graphics keywords are 'symbols', 'add', 'window', 'show',
'title', 'xlab', 'ylab', 'xmin', 'xmax', 'ymin', 'ymax', 'xaxis',
'yaxis', 'lines', 'linetype', 'impulses' and keywords having to do with
files.  'title', 'xlab' and 'ylab' apply to the panel graph as a whole,
not to the pane being drawn.

You should use 'add:T' when drawing a pane in an existing panel graph.

If you use 'symbols', you should probably choose small symbols, ASCII
codes 7 and 9 - 24 ("\7" and "\11" - "\30" or "\07" and "\x09\ -
"\x18".  The default is symbols:"\7" (dot) except with lines:T, when
the default is "".

You may not use graphics keywords 'logx', 'logy' or any keywords
having to do with tick marks.

                           Keyword labels
panelplot(x,y,pos:position,hstrips:h,vstrips:v,labels:vector(labsx,
labsy) [,graphics keywords]), where labsx and labsy are CHARACTER
scalars, allows you interactively to position a label of the form
paste(labsy,"vs",labsx) on the graph.  'labels' is ignored with keyword
phrase 'nolabels:F' or graphics keyword phrase 'show:F'.
```

```
Example:
  Cmd> irisdata <- matread("macanova.dat","irisdata") #Fisher data

  Cmd> # col 1 is variety number, 2 - 5 are data

  Cmd> colnames <- vector("SepLen","SepWid","PetLen","PetWid")

  Cmd> first <- T;for (i,1,4){
        for (j,1,4) {
          if (i != j) {
            panelplot(y[,i],y[,j], pos:vector(i,j),hstrips:4,vstrips:4,\
              title:"Scatter plot matrix of Fisher Iris data",\
              symbols:vector("\21","\22","\23")[irisdata[,1]],\
              show:F,add:!first)
            first <- F
          }
        }
       }

  Cmd> showplot()

This plots every column of the Fisher iris data against every other in
a 4 by 4 panel graph similar to that produced by plotmatrix().

                            Cross references
See also topics plotmatrix(), 'graphics', 'graph_keys', plot(),
lineplot(), chplot().
```

5.13 panel_graphs

Usage:

`panelplot(x,y,...)`	draw a plot in a pane of a panel graph
`panelhist(x,...)`	draw a histogram in a pane of a panel graph
`plotmatrix(x [,y], ...)`	draw a panel graph containing a scatter plot matrix
`plotpanes(x, y, ...)`	draw a panel graph containing plots of columns of y against the corresponding column of x.

Keywords: panel graphs

```
                                Description
A panel graph is a rectagular array of "panes", each of which may
contain a small graph.  The small graphs lack tick marks and any
information as to the actual values plotted other than axes (x = 0
and/or y = 0 lines).

A panel graph consists of h horizontal strips (rows) and v vertical
strips (columns).

A panel graph is scaled so the left edge is the line x = 0, the bottom
```

```
edge is the line y = 0, the right edge is the line x = v and the top
edge is the line y = h.

x and y values for each plot are transformed by
  x -> (x - (max(x)+min(x))/2)/(1.05*(max(x)-min(x)))
and
  y -> (y - (max(y)+min(y))/2)/(1.05*(max(y)-min(y)))
and then centered in a pane.  This leaves a little space between
the plotted points and the edges of the pane.

                            Panel graph macos
There are several macros that draw panel graphs.
  panelplot(x,y,...)       draw a plot in a pane of a panel graph
  panelhist(x,...)         draw a histogram in a pane of a panel graph
  plotmatrix(x [,y], ...)  draw a panel graph containing a scatter plot
                           matrix
  plotpanes(x, y, ...)    draw a panel graph containing plots of each
                           column of y against the corresponding column
                           of x.

Each has its own help entry.

                       Keywords hstrips and vstrips
Keyword phrase 'hstrips:h,vstrips:v' are required on panelhist() and
panelplot() and are optional on plotmatrix() and plotpanes().

                            Graphics keywords
These macros recognize most of the usual graphics keywords such as
'title', 'xlab', 'show' and 'window'.  Exceptions include 'logx',
'logy' and any keywords having to do with tickmarks.

When using panelplot() or panelhist() to add a plot to an existing panel
graph, you must use 'add:T'.

When graphics keywords 'xmin', 'ymin', 'xmin' or 'xmax' are used as
arguments to macros drawing panel graphs, their values replace the
extremes of x and/or y in scaling data and values outside the limits
are omitted.

                            Cross references
See also plotmatrix(), plotpanes(), panelhist() and panelplot().
```

5.14 piechart()

Usage:

```
piechart(p [,start:p0] [,labels:labs] [,rlab:r1 [, rval:r2]\
  [, xwidth:w] [,graphics keyword phrases]), REAL vector p with p[i] >=
  0, p0 >= 0, REAL scalar, CHARACTER vector labs, REAL scalars r1 > 0,
  r2 > 0, w > 0
```

Keywords: distribution graphs

```
                                Usage
piechart(p) draws a pie chart of data in REAL vector p which should
have no MISSING values and satisfy min(p) >= 0.  The chart is drawn in
a circle with radius 1.

Usually p is a vector of proportions (sum(p) = 1) or percentages
(sum(p) = 100), but this is not required.  What is actually drawn is a
pie chart with sector angles computed from the proportions p1 =
p/sum(p).  The angle defining sector I of the pie chart, going
clockwise around the circle is 360*p1[I] degrees.  By default, the
starting edge of sector 1 is vertical.  Each sector is is labelled by
sector number, from 1 to nrows(p).

There several optional keywords.

                          Optional Keywords
start
  piechart(p, start:p0 [,other keywords]) where p0 is a REAL scalar
  draws a pie chart with the starting sector edge located p0 of a cycle
  (abs(p0) <= 1) or p0 percent of a cycle (abs(p0) > 1) around the
  circle from the vertical.  For instance, with p0 = .25 or 25, the
  starting sector edge is 90 degrees clockwise from the vertical.
  'start' can be used with other keywords.

labels
  piechart(p, labels:labs [,other keywords]), where labs is a CHARACTER
  vector with length(labs) = length(p), labels sector I with labs[I].
  You can suppress sector labelling with labels:"".

rlab
  piechart(p, rlab:r1 [,labels:labs] [,other keywords]) puts sector
  labels at radius r1 > 0 from the pie center.  Usually r1 < 1.  The
  default radius is .9.

rval
  piechart(p, rval:r2 [,other keywords]), prints p[I] at radius r2 > 0
  in sector I, I = 1, ..., length(p).  Usually r2 < 1.  This allows you
  to label sectors both with a descriptive tag and the value of the
  data.

xwidth
  piechart(p, xwidth:w [other keywords]) includes 'xmin:-w/2,xmax:w/2'
  in the argument list of the plotting commands that draw the pie
  chart.  You need to use 'xwidth' only when the plot without it looks
  like an ellipse instead of a circle.  The default is 'xwidth:3.2'.

You can also use many of the usual graphics keywords like 'xlab',
'ylab', 'title' and 'window'.  See topic 'graph_keys'.

                          Adding labels
If you don't like the default labeling, you may be able to use Mouse()
and addstrings() to position labels more to your liking.  See Mouse()
and addstrings().
```

5.15 plotmatrix()

Usage:

```
plotmatrix(x [, symbols:syms] [,bottomup:T] [,upper:T  or  lower:T]\
  [,hstrip:h] [,vstrip:v] [,labels:labsx or nolabels:T] \
  [,diaglabs:F] [,name:xname] [,graphics keyword phrases]),
  x REAL matrix, ncols(x) > 1, CHARACTER scalar or vector labsx,
  CHARACTER  scalar xname
plotmatrix(x, y [, symbols:syms] [,bottomup:T] [,hstrip:h] [,vstrip:v]\
  [,labels:structure(labsx,labsy) or nolabels:T]\
  [,names:vector(xname,yname)] [,graphics keyword phrases]), x, y REAL
  matrices with same number of rows, labsx and labsx CHARACTER scalars
  or vectors, xname and yname CHARACTER scalars
For both usages, h > 0, v > 0 are integers, syms is a CHARACTER or
  non-negative integer vector
```

Keywords: multivariate graphs, panel graphs

```
                                  Usage
plotmatrix(x) makes scatter plots of every column of REAL matrix x
against every other column in the panes of a panel graph (see topic
'panel_graphs').  When x has nx columns, the plot is a nx by nx array
of panes, with no graphs on the diagonals.

When x has labels (see topic 'labels'), the column labels are printed
in the diagonal boxes.  Otherwise, "X1", "X2",...  are used as labels.
Column numbers increase down the y axis and across the x axis.

                  Keywords diaglabels, upper and lower
plotmatrix(x, diaglabels:F) does the same except the labels are in
the margins rather than the diagonals.

plotmatrix(x, upper:T) and plotmatrix(x lower:T) do the same except
only the plots in the upper or lower triangle of the display are drawn
in a nx-1 by nx array of plots.  Labels are put in the margins.  You
can't use both 'upper:T' and 'lower:T'.

                          Two matrix arguments
plotmatrix(x,y), where x and y are REAL matrices with the same number
of rows and nx and ny columns, makes a similar plot of every column of
y against every column of x in a ny by nx array of small plots.
Column labels of x and/or y (default "X1","X2",... and "Y1","Y2",...)
are printed in the margins.

plotmatrix(x,y) differs from plotpanes(x,y) in that the latter plots
column j of y only against column j of x, and not all against all
columns of x.

                             Keyword symbols
plotmatrix(x [,y], symbols:syms ...), where syms is a CHARACTER scalar
```

```
or vector or an integer scalar or vector with values between 0 and 999
uses 'syms' as plotting symbols similarly to chplot().  The default
plotting symbols are "\7", the code for a dot.  Small plotting symbols
(ascii codes 7 and 17 - 24, that is, "\7' and "\21" - "\30") are
recommended.  'symbols:syms' can be used with any other keywords.

                          Keyword bottomup
plotmatrix(x [,y], bottomup:T ...) does the same, except that column
numbers increase up the y axis.  'bottomup:T' can be used with any
other keywords.

                      Keywords labels and nolabels
plotmatrix(x [,y], labels:labs ...) does the same, except column
labels are taken from labs.  With just x, labs should be a CHARACTER
scalar or vector labsx.  With x and y, labs should be structure(labsx,
labsy), where labsx and labsy are CHARACTER scalars or vectors.
When labsx or labsy are scalars, they are expanded by appending "1",
"2", ....  When they are vectors, their length must match the number
of columns of x and y.

plotmatrix(x [,y], nolabels:T ...) does the same except no variable
labels are added to the plot.

plotmatrix(x [,y], name:Name ...), does the same, except that
CHARACTER variable Name is used in constructing a title and axis
labels.  With just x, Name should a scalar; with x and y, Name should
be a vector of length 2.

                       Keywords hstrip and vstrip
plotmatrix(x [,y], hstrip:h, vstrip:v ...) does the same, except that
the overall array of small plots will be h by v, where h and v are
positive integers.  It is an error if h or v is too small to allow all
plots.

                          Graphics keywords
You can use many of the usual graphics keywords phrases with
plotmatrix(), including 'symbols', 'window', 'show', 'title', 'xlab',
'ylab', 'xaxis', 'yaxis', 'lines', 'linetype' and keywords related to
files.  See topic 'graph_keys'. Values for 'xmin', 'xmax', 'ymin' and
'ymax' may be vectors with lengths matching the appropriate number of
columns.

You cannot use keywords 'add', 'logx', 'logy', 'impulses', and keywords
related to tick marks.

Example:
  Cmd> irisdata <- matread("macanova.dat","irisdata") #Fisher data

  Cmd> # col 1 is variety number, 2 - 5 are data

  Cmd> colnames <- vector("SepLen","SepWid","PetLen","PetWid")

  Cmd> plotmatrix(irisdata[,-1],lower:T,name:"Iris Data",\
```

```
    symbols:vector("\21","\22","\23")[irisdata[,1]], labels:colnames)

  Cmd> plotmatrix(irisdata[,run(2,3)],irisdata[,run(4,5)],bottomup:T,\
    symbols:vector("\21","\22","\23")[irisdata[,1]],\
    names:vector("Iris Cols 1,2","Iris Cols 3,4"),\
    labels:structure(colnames[run(2)],colnames[-run(2)]))

                            Cross references
See also topics 'graphs', plot(), lineplot(), chplot().
```

5.16 plotpanes()

Usage:

```
   plotpanes(x,y [,addlabels:T, labels:structure(labsx,labsy)]\
     [,names:vector(xname,yname)] [,hstrips:h, vstrips:v]\
     [,graphics keyword phrases]), x and y REAL vectors or matrices,
     labsx, labsy CHARACTER scalars or vectors, xname,yname CHARACTER
     scalars, h > 0, v > 0 integers.
```

Keywords: panel graphs, line graphs

```
                                    Usage
plotpanes(x, y) draws a panel graph (see topic 'panel_graphs') whose
panes contain reduced size plots of each column of REAL vector or
matrix y against the corresponding column of REAL vector or matrix x.
If neither x and y are vectors, ncols(x) = ncols(y) is required.  If
one, say x, is a vector and the other (y) is a matrix, the each column
of y is plotted against x.

The plots are arranged in a h by v array of panes where either h = v or
h = v - 1.

Each pane is drawn using macro panelplot().

plotpanes(x,y) differs from plotmatrix(x,y) in that the plotpanes()
plots column j of y only against column j of x, and not all against all
columns of x.

                            Graphics keywords
You can use most of the standard standard graphics keywords except
'add', 'logx', 'logy' and any keywords having to do with tick marks.
In particular, you can use 'symbols', 'window', 'show', 'title',
'xlab', 'ylab', 'xmin', 'xmax', 'ymin', 'ymax', 'xaxis', 'yaxis',
'lines', 'linetype', 'impulses' and keywords having to do with files.
See topic 'graph_keys'.

The default value of 'symbols' is "\007" = "\x07" (dot), except with
lines:T, when no symbols are drawn.  Small characters, code 9 ("\011"
or "\x09") through 24 ("\030" or "\x18") are recommended.

There are no labelled ticks on the small graphs.  If either or both of
```

```
the x-axis or y-axis are within the limits of the data, they are drawn
unless suppressed by xaxis:F or yaxis:F.

                          Equally spaced x values
If nrows(x) <= 2 and differs from nrows(y), each plot has equally
spaced x-values starting with x[1,] and incrementing by x[2,] (by 1 if
nrows(x) = 1), just as happens with standard plotting commands such as
plot(), lineplot() and chplot().

                       Keywords hstrips and vstrips
plotpanes(x,y,hstrips:h, vstrips:v), h > 0, v > 0 integers, does the
same except the array of plots is h by v.  h and v must satisfy h*v >=
max(ncols(x),ncols(y)).  If just one of 'hstrips' and 'vstrips' is
provided, the value for the other is selected large enough to hold all
plots.  'hstrips' and 'vstrips' can be used with any other keywords.

                              Keyword names
plotpanes(x, y, names:vector(xname,yname)), where xname and yname are
CHARACTER scalars, does the same, with xname and yname used to generate
the graph title and labels for the x and y axes. 'names' can be used
with any other keywords.  If you use keywords 'title', 'xlab' or
'ylab', their values take precedence over labels generated from xname
and yname.

                       Keywords labels and addlabels
plotpanes(x, y, labels:structure(labsx, labsy)), where labsx and labsy
are CHARACTER scalars or vectors, enables interactive addition of
labels to each plot.  If labsx or labsy is a vector, it must have
length max(ncols(x), ncols(y)).  If either is a scalar, it is expanded
to a vector by appending "1", "2", ....  The labels generated for
positioning using Mouse() are of the form paste(labsy[i],"vs",labsx[i]).
Thus, labs(structure("X","Y")) generates labels "Y1 vs X1", "Y2 vs X2",
... .

plotpanes(x,y,addlabels:T) is another way to specify labels.  Column
labels of x and/or y, if they exist, are used for labsx and labsy.
When column labels do not exist, labsx and/or labsy are assumed to be
"X" and "Y", repectively.  With 'addlabels:F', the value of 'labels',
if any, is ignored.

With 'show:F' the values of 'labels' and 'addlabels', if any, are
ignored.

                    Keywords xmin, xmax, ymin and ymax
The values of any of 'xmin', 'xmax', 'ymin' or 'ymax' can either be
scalars to be used in every small graph, or vectors with length
matching ncols(x) or ncols(y).  As usual, MISSING means the extreme is
to be determined from the data

Example:
  Cmd> irisdata <- matread("macanova.dat","irisdata") #Fisher data

  Cmd> # col 1 is variety number, 2 - 5 are data
```

```
  Cmd> colnames <- vector("SepLen","SepWid","PetLen","PetWid")

  Cmd> plotpanes(1,irisdata[,-1], names:vector("Index","Iris data"),\
       labels:structure("Index",colnames),\
       symbols:vector("\21","\22","\23")[irisdata[,1]])

This creates a 2 by 2 panel plot containing plots of all variables
against the observation number.

                            Cross references
See also topics plotmatrix(), panelplot(), panelhist(), 'graphs',
'graph_keys', plot(), lineplot(), chplot().
```

5.17 plotresids()

Usage:

```
   plotresids(type,plots[plot options]) type one of "raw","scaled",
   "standardized", or "studentized", plots one or more of "yhat",
   "rankits","index".
```

Keywords: Residual graphs

```
plotresids does residual plots with an alternate front end from
that provided by resvsrankits, resvsyhat, resvsindex.

There are two required arguments.  The first is the residual type
as a character scalar and can be one of "raw", "scaled",
"standardized", or "studentized".  Scaled residuals have been divided
by the root MSE, standardized are divided by ((1-HII)MSE)^.5, and
studentized are outlier-t style.

The second argument is a character scalar or vector that tells which
plots to make.  The components can be one or more of "yhat",
"rankits", or "index" for plotting versus fitted values, normal scores,
or case numbers.  Finally, you can add plotting keywords.
```

5.18 rowplot()

Usage:

```
   rowplot(x [, graphics keyword phrases]), x a REAL matrix
```

Keywords: line graphs, interaction graphs

```
                                  Usage
rowplot(x) makes an "interaction" plot of the data in the matrix x.  The
plotting positions are the column numbers and the values in x.  Points
within each row are joined by lines.  Any keywords useable in chplot may
```

```
follow x.  Rowplot is implemented as a pre-defined macro.

If option 'dumbplot' has been set False (see options), the plot will be
a low resolution plot unless 'dumb:F' is an argument.

You can use all the usual graphics keywords, including 'title', 'xlab',
'ylab', and 'file'.  See topics 'graphs', 'graph_keys', 'graph_border'
and 'graph_ticks'.

Example:
  Cmd> rowplot(run(20)^(.2*run(5)'),\
        title:"X^vector(.2, X^.4, X^.6, X^.8, X)'")

                              Cross reference
See also topic colplot().
```

5.19 sampcdf()

Usage:

```
   sampcdf(x [,quiet:T, keep:T, draw:T] [,graphics keywords]), x REAL
     vector with no MISSING elements
```

Keywords: distribution graphs, line graphs

```
sampcdf(x [,graphics keywords]), where x is a REAL vector, draws the
sample cumulative distribution function (CDF) of x.  The graphics
keywords may include 'xlab', 'ylab', 'title', 'linetype', 'show' and
'keep'.  See topic 'graph_keys'.  Any MISSING elements are removed

sampcdf(x, quiet:T [,graphics keywords]) does the same except any
warning messages are suppressed.

sampcdf(x, save:T [,graphics keywords]) doesn't draw the CDF.  Instead
it returns as value
   structure(x:xvals,y:yvals [,graphics keywords],lines:T),
where xvals and yvals are REAL vectors defining the lines making up the
CDF.  Note that any graphics keyword phrases that are arguments are
components of the structure.

sampcdf(x, save:T, draw:T [,graphics keywords]) both draws the CDF and
returns a structure.

Example:
Both
   Cmd> sampcdf(x, xlab:"Bunny weight",\
                title:"Sample CDF of Bunny Weights", wind:1)
and
   Cmd> GRAPHWINDOWS[1] <- sampcdf(x, save:T, xlab:"Bunny weight",\
                title:"Sample CDF of Bunny Weights")

will draw a sample CDF in graphics window 1.
```

5.20 vboxplot()

Usage:

```
vboxplot(x1,x2,...,xk [, graphics keyword phrases]), arguments REAL
  vectors
vboxplot(Struc, [, graphics keyword phrases]), Struc a structure with
  REAL vector components
```

Keywords: distribution graphs

```
                                Usage
vboxplot(var1, var2, ... , vark [,graphics keyword phrases]) produces
vertically oriented parallel Tukey boxplots for the vectors var1 through
vark.  It is identical with boxplot(var1, var2, ..., vark, vertical:T
[,graphics keyword phrases]).

vboxplot(Struc [,graphics keyword phrases]) produces vertically oriented
parallel box plots for the components of structure Struc, all of which
must be vectors.  It is identical with boxplot(Struct, vertical:T
[,graphics keyword phrases]).

You can use all the graphics keyword phrases phrases that boxplot()
recognizes.

For more information including how to use split() to create a structure
argument, see boxplot().
```

Chapter 6

Mathematical Macros Help File

This Chapter contains help for the set of macros that do optimization, special functions, series manipulations, and other mathematical operations that are distributed with MacAnova in the file Math.mac.txt. The material here is a reformatting of the help in file Math.mac.txt.

6.1 bfs()

Usage:

```
bfs(x0, fun [,params:params]  [, goldsteps:ngold] [, maxit:maxiter]\
  [,minit:miniter] [,criteria:vector(nsigx,nsigfun,dgrad)]\
  [printwhen:d1] [,recordwhen:d2]), REAL vector x0, macro
  fun(x,i [,params]), integers ngold > 0, maxiter >= 0, miniter > 0,
  nsigx, nsigfun, d1 >= 0, d2 >= 0, dgrad REAL scalar
```

Keywords: minimize, quasi-newton, variable metric

```
                              Introduction
Macro bfs() uses the Broyden-Fletcher-Shanno variable metric algorithm
to minimize a function iteratively.  A golden mean line search is made
at each step.  See Dahlquist and Bjorck, Numerical methods, Prentice
Hall, 1974, p. 443.

bfs() is a "front-end" to macro minimizer() which it calls with
all the arguments to bfs() plus argument 'method:"bfs"'.

                                  Usage
result <- bfs(x0, fun [, params] [,optional keywords]) computes the
minimum of a real function F(x1,x2,...,xk) starting the Broyden-
Fletcher-Shanno iteration at x = x0 = vector(x01,x02,...,x0k),
a REAL vector with no MISSING elements.

See minimizer() for details on the arguments, keywords and the value.

                             Cross references
See also mnimizer(), dfp(), broyden(), and neldermead()
```

6.2 binom()

Usage:

```
binom(n,k), REAL n and k with non negative elements.  If both are non-
  scalars, they must have the same dimensions
```

Keywords: binomial coefficients, integers

```
                                Usage
binom(n,k), where n >= 0 and k >= 0 are REAL scalars with n >= k returns
a binomial coefficient.  n and k need not be integers, but when they are
binom(n,k) returns n!/(k!*(n-k)!) as an exact integer.  Otherwise it
returns gamma(n+1)/(gamma(k+1)*gamma(n-k+1)), where gamma(x) is the
Gamma function.

When just one of n and k is a scalar, binom(n,k) returns a REAL vector,
matrix or array consisting of binomial coefficients computed from the
scalar and each of the elements of the other argument.

When neither n or k is a scalar, both must have exactly the same
dimensions and the result is an array with the same dimesions
consisting of of binomial coefficients computed from corresponding
elements of n and k.

Examples:
  Cmd> binom(4,run(0,4)) # vector(binom(4,0),...,binom(4,4))
  (1)          1          4          6          4          1

  Cmd> binom(run(3,7),3) # vector(binom(3,3),...,binom(7,3))
  (1)          1          4         10         20         35

  Cmd> binom(run(3,7),run(0,4)) # vector(binom(3,0),...,binom(7,4))
  (1)          1          4         10         20         35

                            Cross reference
See also lgamma().
```

6.3 blockdmat()

Usage:

```
blockdmat(A1,A2,...,Ak), A1, A2, ... matrices, all of the same type,
  REAL, LOGICAL or CHARACTER
```

Keywords: matrices

```
                                Usage
B <- blockdmat(A1,A2,...,Ak), where A1, A2, ..., Ak are matrices,
creates a block diagonal matrix B with diagonal blocks A1, ..., Ak.  B
will have the same type as A1, ..., Ak which must all have the same
type.  Elements of B outside the blocks are 0 , F or "", depending on
the type.
```

```
If Aj is mj by nj, j = 1,...,k, then B is m1 + m2 + ... + mk by n1 + n2
+ ... + nk.

Example:                                                    [1 1 1 0 0]
           [1 1 1]       [2 2]                              [1 1 1 0 0]
When  A1 = [1 1 1], A2 = [2,2], then blockdmat(A1, A2) = [0 0 0 2 2]
                         [2,2]                              [0 0 0 2 2]
                                                            [0 0 0 2 2]

                              Cross references
See also dmat(), diag().
```

6.4 broyden()

Usage:

```
broyden(x0, fun [,params:params] [, maxit:maxiter] [,minit:miniter]\
  [,criteria:vector(nsigx,nsigfun,dgrad)] [printwhen:d1]\
  [,recordwhen:d2]), REAL vector x0, macro fun(x,i [,params]), integers
  ngold > 0, maxiter >= 0, miniter > 0, nsigx, nsigfun, d1 >= 0, d2 >=
  0, dgrad REAL scalar
```

Keywords: minimize, quasi-newton, variable metric

```
                               Introduction
Macro broyden() minimizes a function iteratively using a variable metric
algorithm due to Broyden.  It has no linear search step.  See Dahlquist
and Bjorck, Numerical methods, Prentice Hall, 1974, p. 443.

broyden() is a "front-end" to macro minimizer() which it calls with
all the arguments to minimizer() plus argument 'method:"broyden"'.

                                   Usage
result <- broyden(x0, fun [, params] [,optional keywords]) computes the
minimum of a real function F(x1,x2,...,xk) starting the Broyden
iteration at x = x0 = vector(x01,x02,...,x0k), a REAL vector with no
MISSING elements.

See minimizer() for details on the arguments, keywords and the value.
Keyword 'golden' is ignored by broyden().

                              Cross references
See also minimizer() bfs(), dfp(), and neldermead().
```

6.5 cdiag()

Usage:

```
d <- cdiag(a), REAL matrix a representing the fully complex form of a
  square complex matrix A
```

Keywords: complex matrices, matrices

```
                                  Usage
d <- cdiag(a), where a is a REAL matrix representing a square complex
matrix A in fully complex form.

d is a nrows(a) by 2 REAL matrix representing diag(A) in fully complex
form.

It is an error for A to not be square, that is for ncols(a) != 2*nrows(a)
and ncols(a) != 2*nrows(a) - 1.

                              Cross references
See also csubscr(), diag(), 'complex'.
```

6.6 ceigen()

Usage:

```
   eigs <- ceigen(a), REAL matrix a representing the fully complex form of
     a complex Hermitian matrix A (A' = conj(A)).  Result is
     structure(values:vals, vectors:vecs)
```

Keywords: complex matrices, matrices

```
                                  Usage
result <- ceigen(a) computes the real eigenvalues and complex eigen-
vectors of a complex matrix A with Hermitian symmetry (A' = conj(A)),
coded in fully complex form in REAL matrix a with no MISSING elements.

result is structure(values:V, vectors:U).  V is a length n vector of
eigen- values where n = nrows(a).  U is a n by 2*n REAL matrix; columns
2*i-1 and 2*i contain the real and imaginary parts of the i-th complex
eigenvector of A.

                                 Caution
When A has duplicate eigenvalues, some of the eigenvectors computed may
be linearly dependent.

Example:
  Cmd> a <- cmplx(matrix(vector(8,2,2,1),2),matrix(vector(0,-3,3,0),2))

  Cmd> a # 2 by 2 Hermitian symmetric complex matrix
  (1,1)            8            0            2            3
  (2,1)            2           -3            1            0

  Cmd> eigs <- ceigen(a)

  Cmd> eigs
  component: values
  (1)       9.5249     -0.52494
  component: vectors
```

```
  (1,1)     -0.70598     -0.59149     -0.31138     -0.23405
  (2,1)     -0.37378      0.10967      0.86883     -0.30561

  Cmd> cdivc(cmatmultc(a,eigs$vectors),eigs$vectors)
  (1,1)       9.5249   6.4749e-16     -0.52494  -2.4739e-16
  (2,1)       9.5249  -4.0029e-16     -0.52494   2.2808e-16

                            Cross references
See also cdivc(), cmatmultc(), eigen(), 'complex'.
```

6.7 chebcoefs()

Usage:

```
   chebcoefs(vector(a0,a1,...,an)), a0, a1, ..., an non MISSING REAL
     scalars
```

Keywords: expansions, power series, orthogonal polynomials

```
                                  Usage
Suppose Pn(x) = a0+a1*x+ ... + an*x^n is a polynomial in x of degree
n.  Then Pn(x) has a unique expansion, Pn(x) = b0 + b1*T1(x) + ... +
bn*Tn(x), where Tj(x) = the Chebyshev polynomial of degree j defined
for -1 <= x <= 1 as Tj(x) = cos(j*acos(x)).

chebcoefs(vector(a0,a1,...,an)), where a0, a1, ..., an are non MISSING
REAL scalars returns vector(b0,b1,...,bn), where the b's are the
coefficients in the Chebyshev expansion.

                            Cross reference
See also topic invchebcoefs().
```

6.8 cjtranspose()

Usage:

```
   b <- cjtranspose(a), REAL matrix a representing a complex matrix A in
     fully complex form.  b is the transpose conj(A)' in fully complex
     form.
```

Keywords: complex matrices, matrices

```
                                  Usage
b <- cjtranspose(a) is equivalent to b <- ctranspose(cconj(a)) and
computes the transpose of the complex conjugate of complex matrix a in
fully complex form.

                            Cross references
See also ctranspose(), transpose(), cconj(), 'complex'.
```

6.9 cmatmultc()

Usage:

```
c <- cmatmultc(a, b), a and b REAL matrices representing complex
  matrices A and B in fully complex form, with nrows(b) =
  floor((ncols(a) + 1)/2)
c <- cmatmultc(a, b, "%c%"), requires nrows(a) = nrows(b)
c <- cmatmultc(a, b, "%C%"), requires floor(ncols(a) + 1)/2) =
  floor(ncols(b) + 1)/2
```

Keywords: complex matrices, matrices

```
                                Usage
c <- cmatmultc(a, b), computes the matrix product of REAL matrices a and
b interpreted as complex matrices A and B in fully complex form (real
parts in odd columns, imaginary parts in even).

The result c is a REAL matrix interpreted as the complex matrix A %*% B,
in fully complex form.

c <- matmultc(a, b, "%*%") does the same.

It is required that nrows(b) = floor((ncols(a) + 1)/2).

c <- cmatmultc(a, b, "%c%") does the same except the matrix product
is A %c% B = A' %*% B and nrows(a) = nrows(b) is required.

c <- cmatmultc(a, b, "%C%") does the same except the matrix product
is A %C% A = A %*% B' and floor((ncols(a)+1)/2) = floor((ncols(b)+1)/2)
is required.

                            Cross references
See also 'matrices', 'complex'.
```

6.10 continfrac()

Usage:

```
continfrac(a, b), a and b REAL vectors or matrices with nrows(b) =
  nrows(a) or nrows(b) = nrows(a) + 1
```

Keywords: continued fractions, special functions

```
                                Usage
continfrac(a,b), where a and b are REAL vectors with no MISSING
elements, and nrows(a) = nrows(b) = m, evaluates the continued
continued fraction a[1]/(b[1] + a[2]/(b[2] + a[3]/(b[3] + a[4]/... +
a[m]/b[m])).

continfrac(a,vector(b0, b)) returns b0 + continfrac(a,b), when b0 is a
non MISSING real scalar.

a and b can also be matrices with the same shape.  In that case,
```

```
continfrac(a,b) returns hconcat(continfrac(a[,1], b[,1]),...,
continfrac(a[,m], b[,m]).

If either a or b is a vectir and the other has more than 1 column, the
vector is used in each column of the result.

continfrac(a,vconcat(b0',b)) returns b0 + continfact(a,b), where b0 is
is a vector of length ncols(b).
```

6.11 csolve()

Usage:

```
ainv <- csolve(a), REAL matrix a interpreted as a square complex matrix
  in fully complex form
```

Keywords: complex matrices, matrices

```
                                  Usage
ainv <- csolve(a) computes the complex inverse of REAL matrix a,
interpreted as a square complex matrix A in fully coplex form.

It is an error if A is singular.

                                 Caution
It is possible but unlikely that csolve() will report that A is singular
when that is not the case.

                                 Example
  Cmd> areal <- matrix(vector(0.57,-0.24,-0.33,-0.55),2)

  Cmd> aimag <- matrix(vector(0.43,-0.08,-0.16,0.26),2)

  Cmd> a <- cmplx(areal,aimag)

  Cmd> ainv <- csolve(a)

  Cmd> prd <- cmatmultc(ainv,a)

  Cmd> creal(prd)
  (1,1)            1  -2.7756e-17
  (2,1)   3.4694e-17            1

  Cmd> cimag(prd)
  (1,1)            0            0
  (2,1)   6.9389e-17   5.5511e-17

                            Cross references
See also cmplx(), cmatmultc(), creal(), cimag(), solve(), 'matrices',
'complex'.
```

6.12 csubscr()

Usage:

```
y <- csubscr(x,I), REAL matrix x containing complex matrix X in fully
  complex form, legal REAL or LOGICAL subscript or NULL I
y <- csubscr(x,I,J), legal REAL or LOGICAL subscripts or NULL, I and J
```

Keywords: complex matrices, matrices

```
                                  Usage
csubscr() simulates subscript extraction from a complex vector X or m by
n complex matrix X stored in REAL matrix x in fully complex form (real
parts in odd columns, imaginary parts in even columns).  The result y
contains a complex vector or matrix Y in fully complex form.

y <- csubscr(x,I) simulates Y <- X[I,] when I is a vector or Y <- X[I]
when I is a matrix with two columns.  When I is empty or NULL, it
simulates Y <- X[].

y <- csubscr(x,I,J) simulates Y <- X[I,J].  When I is empty or NULL it
simulates Y <- X[,J]; when J is empty or NULL it simulates Y <- X[I,].

y <- csubscr(x) is equivalent to cmplx(creal(x),cimag(y)).

                                  Caution
Unlike real subscripts you cannot assign to csubscr(x, I, J).

                                  Example
Example with 2 by 4 REAL a representing 2 by 2 complex A:
  Cmd> creal(a) # real part
  (1,1)        0.57       -0.33
  (2,1)       -0.24       -0.55

  Cmd> cimag(a) # imaginary part
  (1,1)        0.43       -0.16
  (2,1)       -0.08        0.26

  Cmd> csubscr(a,1,2) # simulates A[1,2]
  (1,1)       -0.33       -0.16

  Cmd> csubscr(a,hconcat(run(2),run(2))) # simulates cdiag(a)
  (1,1)        0.57        0.43
  (2,1)       -0.55        0.26

  Cmd> csubscr(a,,-1) # or csubscr(a,NULL,-1), simulates A[,-1]
  (1,1)       -0.33       -0.16
  (2,1)       -0.55        0.26

                             Cross references
See also cdiag(), 'complex', 'subscripts', creal(), cimag().
```

6.13 ctrace()

Usage:

```
b <- ctrace(a), REAL matrix a interpreted as a square complex matrix in
  fully complex form.
```

Keywords: complex matrices, matrices

```
                                Usage
c <- ctrace(a) computes the complex trace of REAL matrix a, interpreted
as a square complex matrix A in fully complex form.  c has value
cmplx(trace(creal(a)),trace(cimag(a))).  For A to be squre, nrows(a)
must be floor((ncols(a)+1)/2).'

Example:
  Cmd> areal <- matrix(vector(0.57,-0.24,-0.33,-0.55),2)

  Cmd> aimag <- matrix(vector(0.43,-0.08,-0.16,0.26),2)

  Cmd> ctrace(cmplx(areal,aimag))
  (1,1)        0.02        0.69

  Cmd> cmplx(trace(areal),trace(aimag)) # check
  (1,1)        0.02        0.69

                          Cross references
See also trace(), cmplx(), 'complex'.
```

6.14 ctranspose()

Usage:

```
b <- ctranspose(a), REAL matrix a representing a complex matrix in fully
  complex form
```

Keywords: complex matrices, matrices

```
b <- ctranspose(a) computes the complex transpose of the complex matrix
a in fully complex form.  That is creal(b) = creal(a)' and cimag(b) =
cimag(a)'.

                                Example
  Cmd> areal <- matrix(vector(0.57,-0.24,-0.33,-0.55),2)

  Cmd> aimag <- matrix(vector(0.43,-0.08,-0.16,0.26),2)

  Cmd> a <- cmplx(areal,aimag)

  Cmd> atrans <- ctranspose(a)

  Cmd> creal(atrans) # transpose of areal
  (1,1)        0.57       -0.24
```

```
  (2,1)       -0.33       -0.55

  Cmd> cimag(atrans) # transpose of aimag
  (1,1)        0.43       -0.08
  (2,1)       -0.16        0.26

                            Cross references
See also cjtranspose(), 'complex', 'transpose'.
```

6.15 dfp()

Usage:

```
   dfp(x0, fun [,params:params]  [, goldsteps:ngold] [, maxit:maxiter]\
     [,minit:miniter] [,criteria:vector(nsigx,nsigfun,dgrad)]\
     [printwhen:d1] [,recordwhen:d2]), REAL vector x0, macro
     fun(x,i [,params]), integers ngold > 0, maxiter >= 0, miniter > 0,
     nsigx, nsigfun, d1 >= 0, d2 >= 0, dgrad REAL scalar
```

Keywords: minimize, quasi-newton, variable metric

```
                              Introduction
Macro dfp() uses the Davidon-Fletcher-Powell variable metric algorithm
to minimize a function iteratively.  A golden mean line search is made
at each step.  See Dahlquist and Bjorck, Numerical methods, Prentice
Hall, 1974, p. 442.

dfp() is a "front-end" to macro minimizer() which it calls with
all the arguments to bfs() plus argument 'method:"dfp"'.

                                   Usage
result <- dfp(x0, fun [, params] [,optional keywords]) computes the
minimum of a real function F(x1,x2,...,xk) starting the Davidon-
Fletcher-Powell iteration at x = x0 = vector(x01,x02,...,x0k),
a REAL vector with no MISSING elements.

See minimizer() for details on the arguments, keywords and the value.

                              Cross references
See also minimizer(), bfs(), broyden(), and neldermead()
```

6.16 economize()

Usage:

```
   economize(vector(a0,a1,...,an),m), a0, ..., an REAL scalars, m > 0 an
     integer scalar
```

Keywords: expansions, power series

```
                              Introduction
```

```
Macro to "economize" a power series expansion on [-1, 1]

Suppose Fn(x) = a0 +a1*x + a2*x^2 + ... + an*x^n is the n-th partial
sum of the Taylor series expansion of a function F(x) defined on an
interval I contained in (-1, 1).  Then Fn(x) has a unique expansion
         Fn(x) = c0 + c1*T1(x) + c2*T2(x) + ... + cn*Tn(x)
where Tj(x) is the j-th Chebyshev polynomial defined as Tj(x) =
cos(acos(j*x)) for -1 <= x <= 1.

                                  Usage
b <- economize(vector(a0,a1,...,an),m) computes b = vector(b0,b1,...,bm)
where
  Bm(x) = b0 + b1*x + b2*x^2 + ... + bm*x^m =
                c0 + c1*T1(x) + c2*T2(x) + ... + cm*Tm(x)
is the Chebyshev series truncated at Tm.

When Fn(x) is a good approximation to F(x) on I, and, as is the case
with many functions, the coefficients c0, c1, ... converge to 0 much
faster than do a0, a1, ..., Bm(x) with m << n, may be a good
polynomial approximation for F(x) on I.
```

6.17 factorial()

Usage:

```
factorial(x), x REAL
```

Keywords: special functions, integers

```
                                  Usage
factorial(x) computes x! (x factorial), where x is a REAL scalar,
vector, matrix or array.  The result is the same size and shape as
as x.  If any element is MISSING, <= -1 or such that x! is too large
to be computed, the corresponding element of the result is MISSING.

Elements of x need not be integers.  x! is computed as exp(lgamma(x+1)),
except that if x is an integer <= 20 the value should be exact.

                             Cross references
See also binom() and lgamma().
```

6.18 factors()

Usage:

```
factors(vector(n1 [, n2, ...]), positive integer n1, n2, ...
```

Keywords: prime factors

```
                                  Usage
factors(n), where n > 0 is an integer returns a REAL scalar or vector
containing the prime factors of n.

result <- factors(N), where N = vector(n1,n2,...), integers n1 > 0, n2 >
0, ..., sets result to structure(factor(n1), factor(n2), ...), that is
result[j] contains the factors of N[j].  The name of component j is
'prime' if N[j] is prime and 'composite' otherwise.

Note: factors() has been superceded by function primefactors().

Example:
  Cmd> write(factors(2^2^run(5) + 1),format:"10.0f")
  STRUCTURE:
  component: prime
  (1)          5
  component: prime
  (1)         17
  component: prime
  (1)        257
  component: prime
  (1)      65537
  component: composite
  (1)        641    6700417

                              Cross references
See also printfactors() and primefactors().
```

6.19 i0()

Usage:

```
   i0(x), x a REAL scalar, vector, matrix, array or structure of REAL
     components.
```

Keywords: bessel functions, special functions

```
                                  Usage
i0(x) computes values of I-sub-0(x), the modified Bessel function of the
first kind, where x is a REAL scalar, vector, matrix or array.  The
result is REAL of the same shape as x.

When x1, x2, ... are REAL, i0(structure(x1,x2,...)) returns
structure(i0(x1),i0(x2), ...)

i0 is based on equations 9.8.1 and 9.8.2 in Abramowitz & Stegun,
Handbook of Mathematical functions.

                              Cross reference
See also i1().
```

6.20 i1()

Usage:

```
i1(x), x a REAL scalar, vector, matrix, array or structure of REAL
  components.
```

Keywords: bessel functions, special functions

```
                                Usage
i1(x) computes values of I-sub-1(x), the modified Bessel function of the
first kind, where x is a REAL scalar, vector, matrix or array.  The
result is REAL of the same shape as x.

When x1, x2, ... are REAL, i1(structure(x1,x2,...)) returns
structure(i1(x1),i1(x2), ...)

i1 is based on equations 9.8.3 and 9.8.4 in Abramowitz & Stegun,
Handbook of Mathematical functions.

                            Cross reference
See also i0().
```

6.21 invchebcoefs()

Usage:

```
invchebcoefs(vector(b0,b1,...,bn)), b0, b1, ... non MISSING REAL
  scalars
```

Keywords: expansions, power series

```
                                Usage
Suppose Pn(x) = b0+b1*T1(x)+...+bn*Tn(x), where Tj(x) is the Chebyshev
polynomial of degree j, defined as Tj(x) = cos(j*acos(x)) for -1 <= x
<= 1.  Then Pn(x) is a polynomial and can be expressed as Pn(x) =
a0+a1*x+ ... + an*xˆn.

invchebcoefs(vector(b0,b1,...,bn)), where b0, ..., bn are non MISSING
real scalars, returns vector(a0,a1,...,an), the coefficients of the
powers of x of Pn(x).

                            Cross reference
See also topic chebcoefs().
```

6.22 invertseries()

Usage:

```
invertseries(a), REAL vector a with non-MISSING elements
```

Keywords: expansions, power series

```
                                Usage
Suppose A(x) is a function with A(0) = 0 and with Taylor series a1*x +
a2*x^2 + a3^x^3 + ..., where a1 != 0.  Then in a neighborhood of 0
there is an inverse function B(y) is defined such that B(A(x)) = x for
y near 0.  Let b1*y + b2*y^2 + b3*y^3 ... be the Taylor series for
B(y).  The coefficients bj are unique functions of a1, a2, ..., aj; in
particular, b1 = 1/a1.

b <- invertseries(a), where a = vector(a1,a2,...,an) is a REAL vector,
computes the coefficients b = vector(b1,b2,...,bn) of the Taylor series
for B(y).

Because of numerical instability, higher order coefficients in b may
not be accurate.

Example:
Let A(x) = log(1-x) = -x - x^2/2 - x^3/3 - ... . Then B(y) = 1 - exp(y)
= -y - y^2/2 - y^3/6 - y^4/24 - y^5/120 - ... satisfies B(A(x)) = 1 -
exp(log(1 - x)) = 1 - (1 - x) = x.

  Cmd> a <- -1/run(10); a # coefficients of -log(1-x)
  (1)          -1        -0.5    -0.33333       -0.25        -0.2
  (6)    -0.16667    -0.14286      -0.125    -0.11111        -0.1

  Cmd> b <- invertseries(a); b # coefficients of 1 - exp(y)
  (1)          -1        -0.5    -0.16667   -0.041667  -0.0083333
  (6)  -0.0013889 -0.00019841 -2.4802e-05 -2.7557e-06 -2.7557e-07
```

6.23 kronecker()

Usage:

```
kronecker(A,B), A, B REAL matrices with no MISSING values
```

Keywords: matrices

```
                                Usage
C <- kronecker(A, B), where A and B are REAL matrices with no missing
values, computes the Kronecker product of A and B.

C is a nrows(A)*nrows(B) by ncols(A)*ncols(B) made of nrows(a)*ncols(a)
blocks of the form a[i,j]*B.

                            Cross reference
See also topic 'matrices'.
```

6.24 mathhelp()

Usage:

```
mathhelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
mathhelp(topic, subtopic:Subtopics), CHARACTER scalar or vector
  Subtopics
mathhelp(topic1:Subtopics1 [,topic2:Subtopics2 ...])
mathhelp(key:Key), CHARACTER scalar Key
mathhelp(index:T [,scrollback:T])
```

Keywords: general

```
                                Usage
mathhelp(Topic1 [, Topic2, ...]) prints help on topics Topic1, Topic2,
... related to macros in file math.mac.  The help is taken from file
math.mac.

mathhelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these macros.

mathhelp(index:T) or simply mathhelp() prints an index of the topics
available using mathhelp().

mathhelp(Topic, subtopic:Subtopic), where Subtopic is a CHARACTER scalar
or vector, prints subtopics of topic Topic.  With subtopic:"?", a list
of subtopics is printed.

mathhelp(Topic1:Subtopics1 [,Topic2:Subtopics2], ...), where Suptopics1
and Subtopics2 are CHARACTER scalars or vectors, prints the specified
subtopics.  You can't use any other keywords with this usage.

In all the first 4 of these usages, you can also include help() keyword
phrase 'scrollback:T' as an argument to mathhelp().  In windowed
versions, this directs the output/command window will be automatically
scrolled back to the start of the help output.

                             Keyword key
mathhelp(key:key) where key is a quoted string or CHARACTER scalar lists
all topics cross referenced under Key.  mathhelp(key:"?") prints a list
of available cross reference keys for topics in the file.

mathhelp() is implemented as a predefined macro.

                           Cross reference
See help() for information on direct use of help() to retrieve
information from math.mac.
```

6.25 matsqrt()

Usage:

```
matsqrt(A [, symmetric:T or [lower:T]), square positive semi-definite
   matrix A with no MISSING values
```

Keywords: matrices

```
                                Usage
B <- matsqrt(A) computes a matrix square root B of a positive
semi-definite REAL matrix A with no MISSING values.  B satisfies
B' %*% B = A.  B can also be computed by cholesky(A).

B <- matsqrt(A, lower:T) returns the lower triangular matrix square
root of A.

B <- matsqrt(A, symmetric:T) returns the symmetric matrix square
root of A.  B satisfies B %*% B = A

                            Cross reference
See also cholesky().
```

6.26 minimizer()

Usage:

```
minimizer(x0, fun [,params:params] [, method:M] [, goldsteps:ngold] \
  [, maxit:maxiter] [,minit:miniter] [,criteria:vector(nsigx, \
  nsigfun,dgrad)] [printwhen:d1] [,recordwhen:d2]), REAL vector x0,
  macro fun(x,i [,params]), CHARACTER scalar M (one of "bfs", "dfp",
  "broyden", integers ngold > 0, maxiter >= 0, miniter > 0, nsigx,
  nsigfun, d1 >= 0, d2 >= 0, dgrad REAL scalar
```

Keywords: minimize

```
                            Introduction
Macro minimizer() uses a quasi-Newton variable metric algorithm to
minimize a function iteratively.  There is a choice of three methods,
Broyden-Fletcher-Shanno (method:"bfs"), Davidon-Fletcher-Powell
(method:"dfp") and Broyden's (method:"broyden").  For the first two a
golden mean line search is made at each step.  See Dahlquist and
Bjorck, Numerical methods, Prentice Hall, 1974, p. 441-444.

                                Usage
result <- minimizer(x0, fun [, params] [,optional keywords]) computes
the minimum of a real function F(x1,x2,...,xk) starting iteration at x
= x0 = vector(x01,x02,...,x0k), a REAL vector with no MISSING elements.
The Broyden-Fletcher-Shanno updating is used by default.

result <- minimizer(x0, fun [, params] , method:M [,optional
keywords]), where M is one of "bfs", "dfp", or "broyden", does the
same, using the Broyden-Fletcher-Shanno, Davidon-Fletcher-Powell or
Broyden update methods.

Optional argument params is a variable, possibly a structure, with
additional constant information used to compute F(x) and its gradient
vector.
```

```
fun is a macro such that fun(x,0 [,params]) returns F(x[1],x[2],
...,x[k]) and fun(x,1 [,params]) returns a length k REAL gradient
vector (vector of partial derivatives of F(x) with respect to the
elements of x).  fun() should ignore its third argument if not needed.

fun(x,-1 [,params]) should carry out any initialization needed.  If
starting values or elements of param are not appropriate, fun(x, -1,
[params]) should return MISSING.  Otherwise, any non-MISSING value
should be returned.

fun() can use either a formula for derivatives, when one is known, or
compute them by numerical differentiation.

                                  Result
result is structure(x:xmin, f:minVal, gradient:gradient, h:invhessian,
iterations:niter, status:N) where xmin is a REAL vector such that
minVal = F(xmin) is a local minimum, gradient is the length k gradient
vector at xmin, invhessian is a k by k approximation to the inverse of
the Hessian matrix (matrix of second order derivatives of F), niter is
the number of iterations taken and N is an integer indicating
convergence status; see below.

                            Convergence control
Optional keyword phrase criterion:vector(nsigx, nsigfun, dgrad) allows
control over how convergence is determined.  nsigx and nsigfun must be
integers and dgrad a small REAL scalar.  At least one of nsigx,
nsigfun and dgrad must be positive.  A value <= 0 is ignored.

Iteration ends when any of the following occurs
  1.  nsigx > 0 and all elements of x have relative change <=
  10^-nsigx.  For any x[i] with abs(x[i]) < .5, change is relative to
  .5.

  2.  nsigfun > 0 and relative change in F(x) <= 10^-nsigfun.  When
  abs(f(x)) < .5, change is relative to .5.

  3.  dgrad > 0 and ||gradient|| < dgrad

The default value for 'criterion' is vector(8,5,-1).

Keyword phrases minit:miniter and maxit:maxiter specify no check for
convergence is made until iteration miniter and no more than maxiter
iterations will be done.  miniter >= 0 (default 0) and maxiter > 0
(default 30) are integers.

                            Convergence status
When N = 0 (value of component 'status' of the result), no convergence
criterion was satisfied.

When N < 0, iteration was terminated because an illegal value was
encountered.
```

```
When N = 1 iteration ended by test using nsigx.

When N = 2 iteration ended by test using nsigfun.

When N = 3, iteration ended by test using dgrad.

After
  Cmd> result <- minimizer(x0, fun ..., maxit:n)

You can restart the iteration where it stopped by
  Cmd> result <- minimizer(result$x, fun, ..., h:result$h)

                            Other keywords
There other optional keyword phrases that can be arguments.

  h:invhessian      invhessian is a k by k REAL symmetric matrix
                    (default dmat(k,1)) used as starting approximation
                    to inverse Hessian matrix
  goldsteps:m       integer m >= 0 (default 5), the number of cycles
                    to be used in the golden mean linear search; with
                    m == 0 no linear search is done; ignored with
                    method:"broyden"
  printwhen:d1      Integer d1 >= 0.  When d1 > 0, current values of x,
                    F(x), and the gradient are printed on iterations
                    d1, 2*d1, 3*d1, ...
  recordwhen:d2     Integer d2 >= 0.  When d2 > 0, current values of
                    x, F(x) and the gradient on saved in components
                    'xvalx', 'funvals' and 'gradients' of side-
                    effect structure BFSRECORD, DJPRECORD or
                    BROYDNRECORD, depending on the method used.

                           Cross references
See also bfs(), dfp(), broyden(), and neldermead()
```

6.27 moorepenrose()

Usage:

```
moorepenrose(A), A a real matrix with no MISSING values
```

Keywords: matrices, generalized inverse

```
                                  Usage
B <- moorepenrose(A), where A is a real matrix with no MISSING values
computes the Moore-Penrose inverse of a A.

B satisfies A %*% B %*% A = A and B %*% A %*% B = B.

When A is square and non-singular B = solve(A).

When A is m by n, B is n by m.  When m >= n and a is full rank b is
```

```
solve(a %c% a, a').

The result is computed from the singular value decomposition of A.

                              Cross references
See also solve(), svd().
```

6.28 levmar()

Usage:

```
   levmar(b,x,y,f,param [,deriv:deriv,crit:crvec,active:active,\
     maxit:itmax,minit:itmin,print:T])
   or
   levmar(b,x,y,param ,resid:res [,deriv:deriv,crit:crvec,active:active,\
     maxit:itmax,minit:itmin,print:T])
     b        REAL vector of starting values for coefficients
     x        REAL variable.   Without 'resid:res'  a vector or matrix
              with nrows(x) = nrows(y); with 'resid:res' nrows(x) =
              norows(y) is not required
     y        REAL vector of data to be fit
     f        Macro:  fit <- f(b,x,param) returns a vector of fitted values
              with length nrows(x) = nrows(y); not allowed with 'resid:res'
     param    vector or structure of additional parameters for f or NULL
     res      Macro: res(b, x, y, param) computes a vector of residuals of
              length nrows(y).  'resid:res' is required when argument f is
              omitted and is not allowed when f is an argument.
     crvect   vector(numsig, nsigsq, delta), 3 criteria for convergence
     deriv    optional macro: deriv(b,x,y,param,j) computes derivative of
              f(b,x,param) or -res(b,x,y,param)  with respect to b[j]
     active   LOGICAL vector the same length as b
     itmax    integer >= 0, maximum number of iterations permitted (default
              = 30)
     itmin    0 <= minimum <= itmax = number of iterations performed (default
              = 1)
     print    if T, partial results are printed on each iteration

     Returned value is structure(coefs:b_hat,hessian:hes,jacobian:jac,
       gradient:g,rss:Rssmin,residuals:resids,nobs:n,iter:niter,
       iconv:convflag)
```

Keywords: nonlinear least squares, minimize

```
                               Introduction
levmar() uses an analogue of the Levenberg-Marquardt and Gauss algorithms
to minimize a sum of squares.  Specifically, the criterion minimized is
sum(resids^2) where resids is computed as
    resids <- y - f(b, x, param)
or as
    resids <- res(b, x, y, param)
where f() or res() is a macro provided by the user
```

```
You can supply a macro to compute derivatives with respect to the elements
of b analytically or rely on difference-based numerical differentiation.

levmar() is intended for primarily use in higher level macros such as
nlreg() (non-linear regression) and arima() (estimation of ARIMA time
series models).

levmar() is based on a Fortran program for nonlinear least squares of Ken
Brown.  See below for references.

                            Usage and arguments
levmar(b,x,y,f,param) uses an iterative algorithm to find a REAL vector
b_hat which minimizes Rss = sum((y - f(b_hat,x,param))^2), a sum of
squared residuals.

levmar(b,x,y,param, resid:res) does the same except the quantity minimized
is Rss = sum(res(b, x, y, param)^2).

In these usages, derivatives are computed by differences.

levmar(b,x,y,f,param, deriv:der) and levmar(b,x,y,param,resid:res,
deriv:der) do the same, except derivatives are computed by macro der()
instead of by differencing.

The arguments to levmar() are as follows
  b        REAL vector of starting values for b_hat
  x        REAL variable.  When f is an argument, x must be a vector or
           matrix with nrows(x) = nrows(y).  When 'resid:res' is an
           argument, nrows(x) is not required; if x is not used, it should
           be 0
  y        REAL vector of data to be fit
  param    vector, matrix or structure of additional fixed parameters or
           NULL
  f        macro called as fit <- f(b,x,param).  fit is a vector of
           containing nrows(x) = nrows(y) function values
  res      macro called as r <- res(b, x, y, param).  r is a vector
           of nrows(y) residuals
  der      macro called as derivs_j <- der(x,y,param,j).  derivs_j is
           vector containing the nrows(y) derivatives with respect to
           b[j] of the elements of f(b,x,param) or -res(b,x,y,param).

                               Return value
levmar() returns structure(coefs:b_hat,hessian:hes,jacobian:jac,
gradient:g,rss:Rssmin,residuals:resids,nobs:n,iter:niter,iconv:convflag)
where component values are as follows:

  b_hat     REAL vector which minimizes Rss
  hes       jac' %*% jac, an approximation to the Hessian matrix H,
            where H[i,j] = 2nd order partial derivative of Rss/2
            with respect to b_hat[i] and. b_hat[j]
  jac       the nrows(y) by nrows(b) Jacobian matrix; -jac[,j] = vector
            of partial derivatives of the elements of the residual
            vector with respect to b_hat[j].
```

g	the nrows(b) REAL gradient vector with g[j] = partial derivative of Rss/2 with respect to b_hat[j]. g should be close to a vector of zeros.
Rssmin	the minimized value of Rss
resids	vector of residuals of length nrows(y)
n	positive integer = nrows(y) = nrows(resids) = nrows(jac)
niter	positive integer = the number of iterations
conflag	Convergence status flag; 0 = not converged, 1 = met relative change in b_hat criterion, 2 = met relative change in Rss criterion, 3 = met norm of gradient vector criterion, 4 = failed to reduce Rss on a step of the iteration.

When any parameters are inactive as specified by keyword 'active' (see below), jac is nrows(y) by p, hes is p by p and g has length p where p = number of active parameters.

Keyword phrase arguments

There are several of keywords that can be used to control the iteration and hold parameters at fixed values.

Keyword Phrase	Value and Explanation
crit:crvec	vector(numsig, nsigsq, delta), 3 criteria for convergence (default = vector(8,5,-1) numsig = desired number of significant digits in the elements of b_hat (conflag = 1 when met) nsigsq = desired number of significant digits in the minimized Rss (conflag = 2 when met) delta is a threshold for ||g||. Iteration is stopped when ||g|| <= delta (conflag = 3 when met) A negative criterion is not used
active:act	LOGICAL vector the same length as b. b[j] "participates" in the iteration only if act[j] is True (default = rep(T,length(b))). When act[j] is False, b_hat[j] remains at the starting value
maxit:itmax	Non-negative integer specifying the maximum number of iterations (default = 30). When itmax = 0, no iterations are done and the quantities returned are computed at the starting value b
minit:itmin	Non-negative integer < itmax specifies the mininum number of iterations
print:T	When T, partial results are printed on each iteration

Iteration stops when (i) itmax is exceeded, (ii) any of the three convergence criteria are satisfied, or (iii) when there has been no reduction in Rss on an iteration after 10 halvings of the initial step size.

Convergence criteria

There are 3 possible convergence criteria, at least one of which must be enabled. levmar() terminates iteration when at least itmin iterations have been completed and any convergence criterion is satisfied or when itmax iterations have been completed.

```
The criteria are specified by optional argument crit:vec, where vec is
vector(numsig [, nsigsq [, delta]]) with length <= 3.  The default values
for numsig, nsiqsq and delta are 8, 5 and -1, respectively.

A negative value for a criterion means it is not active.

  numsig   Desired number of significant digits in every active
           coefficient.  Specifically, the criterion is satisfied if,
           for every active coefficient b[j], the change d_j satisfies
           abs(d_j) < 10^-numsig*max(.5,abs(b_j)) where b_j is the
           updated value of b[j].  Component 'iconv' of the return value
           is 1 when satisfied.

  nsigsq   Desired number of significant digits in Rss = the residual
           sum of squares.  Specifically, the criterion is satisfied
           when abs(Rss_new - Rss_old) < 10^-nsigsq*max(.5,Rss_new).
           Component 'iconv' of the return value is 2 when satisfied.

  delta    Desired maximum norm ||g|| of the gradient vector g.
           Specifically, the criterion is satisfied when
           sqrt(sum(g^2)) <= delta.  Component 'iconv' of the return
           value is 3 when satisfied.

For numsig and nsigsq, the returned coefficients are the values updated
on that iteration.  For delta, the returned coefficients are the values
found on the previous iteration.

Criteria are checked in the order delta, numsig and nsigsq.

Example:
Fit the function b1 + b2*b3^x to data from Snedecor and Cochran with
starting values b1 = b2 = 40 and b3 = 1.

  Cmd> x <- vector(0,1,2,3,4,5)

  Cmd> y <- vector(57.5, 45.7, 38.7, 35.3, 33.1, 32.2)

  Cmd> func <- macro("@b <- $1; @b[1]+@b[2]*@b[3]^($2)", dollars:T)

  Cmd> startVal <- vector(40,40,1) # starting values for iteration

  Cmd> stuff <- levmar(startVal,x,y,func,NULL); stuff
  component: coefs
  (1)      30.724      26.821     0.55184
  component: hessian
  (1,1)           6      2.1683      111.39
  (2,1)      2.1683      1.4367      30.242
  (3,1)      111.39      30.242      2675.8
  component: jacobian
  (1,1)           1           1           0
  (2,1)           1     0.55184      26.821
  (3,1)           1     0.30453      29.602
  (4,1)           1     0.16805      24.503
```

```
(5,1)           1    0.092736      18.029
(6,1)           1    0.051176      12.436
component: gradient
(1)  5.5896e-08 -8.4145e-09  2.4503e-05
component: rss
(1)    0.097248
component: residuals
(1)   -0.044919     0.17523    -0.19159    0.068869    -0.11115
(6)     0.10356
component: nobs
(1)           6
component: iter
(1)           7
component: iconv
(1)           1

Cmd> # compute MSE and approximate standard errors

Cmd> mse <- stuff$rss/(stuff$nobs - length(stuff$coefs)); mse
(1)     0.032416

Cmd> sqrt(mse*diag(solve(stuff$hessian)))
(1)      0.23099       0.2577     0.008448

Cmd> # don't iterate over coefficient 1

Cmd> startVal1 <- vector(30,40,1)

Cmd> levmar(startVal1,x,y,func,NULL,active:vector(F,T,T))
component: coefs
(1)          30      27.418     0.57447
component: hessian
(1,1)      1.4907      34.492
(2,1)      34.492      3136.2
component: jacobian
(1,1)           1           0
(2,1)     0.57447      27.418
(3,1)     0.33002      31.502
(4,1)     0.18959      27.145
(5,1)     0.10891      20.792
(6,1)    0.062567      14.931
component: gradient
(1)  3.5552e-09  0.00067866
component: rss
(1)     0.38885
component: residuals
(1)     0.08214    -0.05082    -0.34842     0.10193     0.11385
(6)     0.48454
component: nobs
(1)           6
component: iter
(1)           7
component: iconv
```

```
  (1)          1
```

```
                                References
For information on the algorithm, see K M Brown and J E Dennis,
Derivative free analogues of the Levenberg-Marquardt and Gauss
algorithms for nonlinear least squares approximation, Numerische
Mathematik, Vol. 18, pp. 289-297 (1972), and K M Brown, Computer
oriented methods for fitting tabular data in the linear and nonlinear
least squares sense, Technical Report No. 72-13, University of
Minnesota Department of Computer and Information Sciences.
```

6.29 neldermead()

Usage:

```
   neldermead(fun, xstart, steps [,data] [, maxeval:maxeval, print:ip,\
     stopcrit:eps, nloop:nloop, quad:F or T, simpcrit:s]),
     fun a macro, xstart and steps REAL vectors with no MISSING elements,
     data a variable as required by fun(), integers maxeval > 0, nloop > 0
     and ip, REAL scalars eps > 0 and s > 0.  fun() specifies a
     function to be minimized and is called as fun(x) or fun(x,data),
     where x is a REAL vector the same length as xstart.
```

Keywords: minimize, direct search

```
                                Introduction
neldermead() is a macro implementing function minimization by direct
search using the simplex method.  The minimum found may be a local,
not a global, minimum.  For details, see Nelder & Mead, The Computer
Journal, January 1965.

                                    Usage
result <- neldermead(fun, xstart, steps) attempts to find a minimum of
the function F(x) computed by macro fun().

xstart, a REAL vector with no MISSING values, contains starting values
for x.

Macro fun() will be called as f <- fun(x, NULL) to evaluate
F(x), with argument 2 to be ignored.

steps is a REAL vector of non-negative numbers the same length as
xstart.  When steps[i] = 0, x[i] will remain fixed at xstart[i].  When
steps[i] > 0, it specifies an initial step size for building the
simplex.

result <- neldermead(fun, xstart, steps, data), where data is an
arbitrary variable, possibly a structure, does the same, except that
fun() will be called as f <- fun(x, data).  Argument data can contain
data or other constant information such as a fixed parameter.

result <- neldermead(fun, xstart, steps [,data], quad:T) does the same,
```

```
except a quadratic approximation is fitted to F(x) near the minimum
found by searching and then the minimum of that approximation is
found.

The value returned has the form structure(x:xmin, f:minval,
invhessian:v, neval:n, status:s).

                            Returned value
The components of the structure returned are as follows:
  x:xmin        REAL vector contains the location of the minimum found
  f:minval      REAL scalar = F(xmin), the minimum attained
  invhessian:V  NULL or REAL square matrix describing the quadratic surface
                fitted with quad:T.
  neval:n       Integer > 0, the number of function evaluations required
  status:s      Integer >= 0 specifying the termination status. s = 0
                means successful termination at the apparent minimum; s
                = 1 means termination because of excessive function
                evaluations; s = 2 means v is not positive semidefinite
                (only with quad:T), indicating stationary point is not a
                minimum.

When F(x) = -log L(x), where x is a vector of parameters and L(x) is a
likelihood function, V is the inverse of the observed information matrix
and can be used as a variance-covariance matrix of the maximum
likelihood estimates.

When F(x) = sum((y - yhat(x))^2) is a sum of squared residuals, 2*MSE*V
is an estimate of the variance-covariance matrix of the least squares
estimates, where MSE = minval/edf, edf = error degrees of freedom.

                         Keyword phrase arguments
neldermead recognizes a number of additional keyword phrases.

  maxeval:m   Integer m > 0, the maximum number of function evaluations
              allowed; default is m = 1000
  crit:eps    REAL scalar eps > 0.  Search will terminate when SD <= eps,
              where SD = standard deviation of values of F(x) on a
              simplex.  The default is eps = 1e-5
  checkwhen:n Integer n > 0; the stopping rule is applied after every
              n function evaluations. The default is n = 10.
  simp:s      Small REAL scalar s > 0, a criterion for expanding the
              final simplex to overcome rounding errors before fitting
              the quadratic surface with quad:T.  The default is s =
              1e-8.  See below for a guideline.
  print:ip    Integer scalar ip controlling printing.  With ip < 0 (the
              default), nothing will be printed unless an error is
              found.  With ip = 0, the found minimum value and its
              location will be printed as well as warning messages.
              With ip > 0, the current value of x and F(x) will be
              printed every ip function evaluations.  When ip >= 0
              the returned structure is "invisible"; it can be
              assigned but won't be printed automatically.
```

```
                          Advice on usage
When the function minimized can be expected to be smooth in the vicinity
of the minimum, you are are strongly urged to use 'quad:T' to specify
the quadratic-surface fitting option.  This is the only satisfactory
way of testing that the minimum has been found.  When the fitted
quadratic surface is not positive definite (value of status = 2), it
probably means that the search terminated prematurely and you have not
found the minimum.

You should use simp:s, where s >= 10000 * E, where E = the rounding
error in calculating F(x). For example, the default simp:1e-8 is
appropriate when the rounding error in calculating F(x) may be of the
order of 1e-12.  If, say, the F(x) is computed by numerical integration
with accuracy on the order of 1e-05, simp:0.1 would be appropriate..

This advice is derived from the comments in the Fortran source on which
neldermead() was modeled.

                         History and references
neldermead() is based on a Fortran program with the following history
  Programmed by D.E.Shaw, CSIRO,  Division of Mathematics & Statistics
     P.O. Box 218,  Lindfield,  N.S.W. 2070
  With amendments by R.W.M.Wedderburn,Rothamsted Experimental Station,
     Harpenden,  Hertfordshire,  England
  Further amended by Alan Miller, CSIRO, Division of Mathematics &
     Statistics, Private Bag 10,  Clayton,  Vic. 3168

                           Cross references
For other macros to minimize functions, see minimizer(), bfs(), dfp()
and broyden().
```

6.30 orthopoly()

Usage:

```
   orthopoly(x, n, [,polycode [,parameters]]), x a REAL vector with no
     MISSING values, n >= 0 an integer, polycode one of p, j, g, t, u,l, h
     and d, parameters a REAL scalar or vector
```

Keywords: orthogonal polynomials

```
                                  Usage
orthopoly() is a macro which computes values of several of the standard
orthogonal polynomials.

P <- orthopoly(x, n), where x is a REAL vector with no MISSING values
and n > 0 is an integer, computes the matrix
  P = hconcat(P0(x), P1(x), ..., Pn(x))
where Pj(x) = vector(Pj(x[1], Pj(x[2]), ...) and Pj is the j-th
Legendre polynomial.  When x is a scalar, P is vector(P0(x),..., Pn(x)).
```

```
P <- orthopoly(x, n, polycode [,parameters]) does the same except
the type of polynomials is determined by polycode, an unquoted letter
which must be one of p, j, g, t, T, u, U, l, h and d.

parameters is required when polycode is j, G, or l, is optional when
polycode is d and should not be an argument otherwise.

                          Available polynomials
The following table summarizes the options
  Polynomial type      polycode  parameters
  Legendre                p       none
  Jacobi                  j       vector(alpha,beta)
  Gegenbauer              g       alpha
  Chebyshev 1             t       none
  Shifted Chebyshev 1     T       none
  Chebyshev 2             u       none
  Shifted Chebyshev 2     U       none
  Laguerre                l       alpha
  Hermite                 h       none
  Discrete                d       vector w of with w[i] > 0; default
                                  is w = rep(1,length(x)).

                          Discrete polynomials
The discrete polynomials are defined by both vector x and the vector of
weights.  They are orthogonal on the discrete set x[1], ..., x[m],
where m = length(x).  That is, they satisfy
  w[1]*Pj(x[1])*Pk(x[1]) + w[2]*Pj(x[2])*Pk(x[2]) + . . . +
     w[m]*Pj(x[m])*Pk(x[m]) = 0, j != k

                          Recurrence relations
All but the discrete polynomials are computed from the recurrence
relations in Table 22.7 of Handbook of Mathematical Functions by
Abramowitz and Stegun and satisfy the normalizations in Table 22.4.
The discrete polynomials are also computed by recursion and are
standardized so that sum(wj*poly(x[j])^2)/sum(wj) = 1
```

6.31 partitions()

Usage:

```
partitions(n [,all:T])
```

Keywords: integers

```
                                    Usage
partitions(n), where n > 0 is an integer, computes a matrix with n
columns whose rows, possibly padded with 0, are the partitions of n.  A
partition of n is a non-increasing set of integers summing to n.

partitions(n,all:T) returns a structure R with n components, such that
R[j] is a matrix with j columns whose rows are the partitions of j
```

```
(padded with 0).

The number N of partitions of n grows fairly rapidly:
  n     N     n     N     n     N     n     N
  1     1     7    15    13   101    19   490
  2     2     8    22    14   135    20   627
  3     3     9    30    15   176    21   792
  4     5    10    42    16   231    22  1002
  5     7    11    56    17   297    23  1255
  6    11    12    77    18   385    24  1575

Examples:
  Cmd> partitions(4) # all 5 partitions of 4; rows add to 4
  (1,1)          4          0          0          0
  (2,1)          3          1          0          0
  (3,1)          2          2          0          0
  (4,1)          2          1          1          0
  (5,1)          1          1          1          1

  Cmd> partitions(3,all:T) # partitions of 1, 2 and 3
  component: Parts_of_1
  (1,1)          1
  component: Parts_of_2
  (1,1)          2          0
  (2,1)          1          1
  component: Parts_of_3
  (1,1)          3          0          0
  (2,1)          2          1          0
  (3,1)          1          1          1
```

6.32 printfactors()

Usage:

```
printfactors(vector(n1 [n2, ...,]), integers n1 > 0, n2 > 0, ...
```

Keywords: prime factors

```
                                  Usage
printfactors(vector(n), where n > 0 is an integer prints n and its prime
factors, if any.  It does not return any values.

printfactors(N), where N = vector(n1,n2,...), integers n1 > 0, n2 > 0,
..., prints each element of N together with its prime factors, if any.

Example:
  Cmd> printfactors(2^2^run(5) + 1)
  5 is prime
  17 is prime
  257 is prime
  65537 is prime
```

```
  4294967297 = 641*6700417

                            Cross references
See also factors() and primefactors().
```

6.33 qrdcomp()

Usage:

```
   qrdcomp(x) or qrdcomp(qr(x)), where x is a REAL matrix with no MISSING
     values
```

Keywords: matrices, qr decomposition

```
                                    Usage
qrdcomp(X) computes a QR decomposition without pivoting of M by N REAL
matrix X with no MISSING values.  It returns structure(q:Q,r:R), where
Q and R are REAL matrices satisfying X = Q %*% R.

When M >= N, Q is M by N orthonormal (Q' %*% Q = N by N identity
matrix) and R is N by N upper triangular.  When M < N, Q is M by M
orthonormal and R is M by N with the first M columns an upper
triangular matrix.

qrdcomp() also works with the output from function qr().  That is
qrdcomp(qr(X)) is the same as qrdcomp(X).

Note:  Computation of the QR decomposition without pivoting is not
reliable when X is not of full rank.

                                 Keyword pivot
qrdcomp(X, pivot:T) or qrdcomp(qr(X, pivot:T)) a QR decomposition or X
with pivoting and return a structure(q:Q,r:R,pivot:J), where J is a
permutation of run(N), such that X[,J] = Q %*% R, where Q and R are as
before.  This usage is accurate even when X is not of full rank.

                                Cross reference
See also qr()
```

Chapter 7

Multivariate Macros Help File

This Chapter contains help for the set of macros that do multivariate analysis distributed with MacAnova in the file Mulvar.mac.txt. The material here is a reformatting of the help in file Mulvar.mac.txt.

7.1 backstep()

Usage:

```
backstep(H,E,fh,fe), H and E symmetric REAL matrices of the same size
  with no MISSING values
```

Keywords: classification, discrimination, stepwise

```
NOTE: This macro is OBSOLETE and is retained only for backward
compatibility because it was in file MacAnova.mac in earlier versions of
MacAnova.  For doing stepwise variable selection in discriminant
analysis you should use newer macros dastepsetup(), daentervar(),
daremovevar(), dastepstatus() and dasteplook().

                          When backstep() is used
You can use macro backstep() to perform a variable elimination step in
backwards stepwise variable selection in linear discriminant analysis.

Macro backstep() is intended to be used after you have used manova() to
compute hypothesis and error matrices H and E, with fh and fe degrees of
freedom respectively.

                            Status information
Status information about the variables currently "in" and "out" is
maintained in integer vectors INS and OUTS containing numbers of
variables currently included and currently excluded.  When no variables
are "in", INS = NULL (INS = 0 means the same thing); when all variables
are "in", OUTS = NULL.  INS must be initialized, usually to run(p),
before backstep() can be used.

                                  Usage
backstep(H,E,fh,fe) computes F-to-delete for all variables currently
```

```
included in vector INS.  It then updates INS by removing the index of
the variable with the smallest F-to-delete, setting INS to 0 if no
variable is left "in".  OUTS is computed as run(p) when there are no
variables "in" and as run(p)[-INS] otherwise.

                              Value returned
The value returned is structure(f:F_to_delete, df:vector(fh,fe-k+1),
ins:INS,outs:OUTS), where F_to_delete is the vector of F-to-delete
statistics and k = length(INS) before deletion.  INS and OUTS are
copies of the updated INS and OUTS vectors.  The F-to-delete statistics
have nominal degrees of freedom fh an fe - k + 1.

The variable that is deleted is the one with the smallest F-to-delete
statistic.  If this is large enough, you may want to reenter the
variable using forstep().  See forstep().

                                   Example
You can initialize things by
  Cmd> manova("y = groups",silent:T) # response matrix y, factor groups

  Cmd> H <- matrix(SS[2,,]); E <- matrix(SS[3,,])

  Cmd> fh <- DF[2]; fe <- DF[3]

  Cmd> INS <- run(nrows(H)) # all variables "in"

Macro forstep() is available for doing a forward step (variable
inclusion).  One difference between backstep() and forstep() is that
backstep() determines the variable to eliminate, and then updates INS
and OUTS; you must tell forstep() which variable to include.  See
forstep() for details.  See also compf() which computes F-to-enter for
variables not in INS.  compf() does not compute F-to-delete.

Both forstep() and compf() are OBSOLETE and are retained only for
backward compatibility.

                              Cross references
See also manova(), daentervar(), daremovevar(), dastepsetup(),
dastepstatus() and dasteplook().
```

7.2 chiqqplot()

Usage:

```
chiqqplot(y [,sqrt:T] [,graphics keywords phrases]), REAL matrix y
chiqqplot(dsq, df [,sqrt:T] [, graphic keyword phrases]), REAL vector
  or matrix dsq of squared distances, REAL scalar df > 0
```

Keywords: plotting, diagnostics

```
                                   Usage
You use chiqqplot() to make a chisquare or sqrt(chisquare) quantile-
```

```
quantile (Q-Q) plot of ordered generalized distances against chisquare
or sqrt(chisquare) quantiles.

chiqqplot(y [, graphics keyword phrases]), where y is a REAL matrix
with no MISSING elements, plots the ordered values of generalized
distances against x[i] = invchi((i - 1/2)/n, df), where df = p =
ncols(y).  Common graphics keywords are 'xlab', 'ylab', 'title' and
'symbols'.  'symbols' may be used as with chplot(), except that
'symbols:0' labels the points by row number.

The generalized distance of y[i,] from ybar, the sample mean row vector,
= sum(y)/nrows(y) is
      dsq[i] = (y[i,] - ybar) %*% solve(s) %*% (y[i,] - ybar)'
with s the sample variance-covariance matrix with divisor n-1.

                          Assessment of normality
When the rows of y are a random sample from a multivariate normal
distribution, d[i] is distributed approximately as chi-squared on p
degrees of freedom and the plot should approximate a straight line with
slope 1.

Very commonly, the "data" matrix is RESIDUALS, the matrix of residuals
computed by manova().  The Q-Q plot is a way to assess the multivariate
normality of the errors.

                                Keyword sqrt
chiqqplot(y, sqrt:T ...) does the same, except the ordered values of
sqrt(dsq[i]) are plotted against sqrt(invchi((i - 1/2)/n, df)).
Again the plot should be linear when y is multivariate normal.

                             Distance argument
chiqqplot(dsq, df, [,sqrt:T] ...) does the same except the ordered
columns of REAL vector or matrix dsq or sqrt(dsq) are plotted against
quantiles of chisquare or sqrt(chisquare) with df degrees of freedom.
When dsq is computed by distcomp(y) and df = ncols(y), this usage is
equivalent to chiqqplot(y, [,sqrt:T] ...)

When dsq is a matrix, the columns are separately ordered and plotted
with different symbols.  With symbol:0, the symbols are the row numbers
when ncols(dsq) = 1 and are the column numbers otherwise.

chiqqplot() uses macros covar() and distcomp().
```

7.3 compf()

Usage:

```
f <- compf(h,e,fh,fe), REAL symmetric matrices h and e, positive
  integers fh and fe; integer vector INS must be defined
```

Keywords:

```
NOTE: This macro is OBSOLETE and is retained only for backward
compatibility because it was in file MacAnova.mac in earlier versions of
MacAnova.  For doing stepwise variable selection in discriminant
analysis you should use newer macros dastepsetup(), daentervar(),
daremovevar(), dastepstatus() and dasteplook().

                          When compf() is used
Macro compf() computes Fs-to enter at any stage in stepwise variable
selection in linear discriminant analysis.

You can use compf() after manova() has computed hypothesis and error
matrices H and E, with fh and fe degrees of freedom respectively.

                                Setup
Integer vector INS must be defined, containing the variable numbers
that are "in".  INS = 0 means no variables are in and the Fs-to-enter
are simply the separate ANOVA Fs.  When INS != 0, the Fs-to-enter are
the analysis of covariance Fs for each "out" variable, with the "in"
variables being used as covariates.

                                Usage
compf(H,E,fh,fe) returns structure(f:f_to_enter,df:vector(fh,fe-k),
ins:INS, outs:OUTS), where OUTS is run(p) when INS = 0, is NULL when
INS contains all integers 1, ..., p and is run(p)[-J] otherwise, where
p = ncols(H).  k is the number of variables "in".  The F-to-enter
statistics have nominal degrees of freedom fh and fe - k.

                               Example
Here is an example of starting forward stepwise variable selection.

  Cmd> manova("y = groups",silent:T)#, response matrix y, factor groups

  Cmd> H <- matrix(SS[2,,]); E <- matrix(SS[3,,])

  Cmd> fh <- DF[2]; fe <- DF[3]

  Cmd> INS <- 0 # no variables in

  Cmd> results <- compf(H,E,fh,fe)

  Cmd> j <- grade(results$f,down:T)[1] # index of largest F

  Cmd> # now continue with forstep(j,H,E,fh,fe)
```

7.4 covar()

Usage:

```
  covar(x), REAL matrix x with no MISSING values, returns
    structure(n:sampleSize,mean:xbar,covariance:s)
```

Keywords: covariance, descriptive statistics

```
                                  Usage
covar(x), where x is a REAL matrix with no MISSING values computes the
sample mean and variance-covariance matrix of x.  It returns
         structure(n:N, mean:xbar, covariance:s)
where N = nrows(x) is the sample size, xbar is a row vector of the
sample means of the columns, and s is the sample variance-covariance
matrix (with divisor N - 1).

Macro covar() is obsolete but is retained for backward compatibility.

                          Comparison with tabs()
Essentially the same output can be obtained from tabs(y, count:T,
covar:T, mean:T) which returns structure(mean:means, covar:s,count:n).
The components of covar() output differ from those of tabs() in three
ways:
 (a) Component names ('covariance' instead of 'covar' and 'n'
     instead of 'count')
 (b) The value of 'mean' is a row vector (1 by ncols(x)) for covar() and
     a (column) vector for tabs()
 (c) The value of covar() component 'n' is the scalar N, while tabs()
     component 'count' is rep(N,ncols(x)), with a count for each column
     of x

For both covar() and tabs() with covar:T, it is an error if y has any
MISSING elements.

                            Cross references
See also tabs() and groupcovar().
```

7.5 daentervar()

Usage:

```
daentervar(var1 [,var2 ...] [,silent:T]), var1, var2 ... names
  or numbers of variables to be entered
```

Keywords: classification, discrimination, stepwise

```
                                  Usage
daentervar(j), where j is a positive integer, enters dependent variable
j as part of a stepwise dependent variable selection, usually as one
stage in stepwise discriminant analysis.  This is what is sometimes
called a "forward" step.  It is an error if variable j is already an
"in" variable.

daentervar() updates variable _DASTEPSTATE which encapsulates the
current state of the variable selection process and prints a report with
the new values of F-to-enter or F-to-remove, and their P-values.  See
topic '_DASTEPSTATE' and dastepsetup() for more details.  It returns a
copy of the updated _DASTEPSTATE as an invisible variable that can be
assigned but is not normally printed.
```

```
You normally choose the variable to enter as the variable with the
largest value of F-to-enter as printed by macro dastepsetup(),
dastepstatus() or a preceding use of daentervar()

                          Entering named variable
daentervar(varname), where varname is the quoted or unquoted name of
a variable in the model, does the same.  Thus if the original data
matrix had column labels "SepLen", "SepWid", "PetLen" and "PetWid",
either daentervar(SepWid) or daentervar("SepWid") would be equivalent
to daentervar(2).

                        Entering several variables
daentervar(j1, j2, ...) and daentervar(varname1, varname2, ...)
successively enter several variables, printing a report after each is
entered.  The value returned is _DASTEPSTATE after all the variables
have been entered.

                             Keyword silent
daentervar(j1 [,j2, ...], silent:T) and daentervar(varname1 [,varname2,
...], silent:T) do the same, except no report is printed.  This would
normally be followed by dastepstatus() to print a report after all the
variables have been entered.

                            Cross references
See also dastepstatus(), daremovevar()  and dasteplook().
```

7.6 daremovevar()

Usage:

```
daremovevar(var1 [,var2 ...] [,silent:T]), var1, var2 ... names or
  numbers of variables to be removed
```

Keywords: classification, discrimination, stepwise

```
                                    Usage
daremovevar(j), where j is a positive integer, removes dependent
variable j as part of a stepwise dependent variable selection, usually
as one stage in stepwise discriminant analysis.  This is what is
sometimes called a "backward" step.  It is an error if variable j is
not an "in" variable (is already an "out" variable).

daremovevar() updates variable _DASTEPSTATE which encapsulates the
current state of the variable selection process and prints a report with
the new values of F-to-remove or F-to-remove, and their P-values.  See
topic '_DASTEPSTATE' and dastepsetup() for more details.  It returns a
copy of the updated _DASTEPSTATE as an invisible variable that can be
assigned but is not normally printed.

You normally choose which variable to remove as the variable with the
```

```
smallest value of F-to-remove as printed by macro dastepsetup(),
dastepstatus() or a preceding use of daremovevar().

                          Removing named variable
daremovevar(varname), where varname is the quoted or unquoted name of a
variable in the model, does the same.  Thus if the original data matrix
had column labels "SepLen", "SepWid", "PetLen" and "PetWid", either
daremovevar(SepWid) or daremovevar("SepWid") would be equivalent to
daremovevar(2).

                        Removing several variables
daremovevar(j1, j2, ...) and daremovevar(varname1, varname2, ...)
successively remove several variables, printing a report after each is
removed.  The value returned is _DASTEPSTATE after all the variables
have been removed

                             Keyword silent
daremovevar(j1 [,j2, ...], silent:T) and daremovevar(varname1
[,varname2,...], silent:T) do the same, except no report is printed.
This would normally be followed by dastepstatus() to print a report
after all the variables have been removed.

                            Cross references
See also dastepstatus(), daentervar()  and dasteplook().
```

7.7 dasteplook()

Usage:

```
   dasteplook(name1, name2, ...), names selected from  'model', 'hpluse',
     'e', 'in', 'F', 'fh', 'fe', and 'history' dasteplook(all)
```

Keywords: classification, discrimination, stepwise

```
                                    Usage
dasteplook(compname), where compname is the name of a component of
variable _DASTEPSTATE, that is one of 'model', 'hpluse', 'e', 'in',
'F', 'fh', 'fe', and 'history', returns that component of
_DASTEPSTATE.  See topic '_DASTEPSTATE' for details on these components

compname can either be quoted, as in dasteplook("hpluse"), or unquoted,
as in dasteplook(hpluse).

                        Viewing several components
dasteplook(compname1, compname2, ...), where the compnames are quoted
or unquoted _DASTEPSTATE component names, returns a structure
consisting of those components.

dasteplook(all) or dasteplook("all") return all of _DASTEPSTATE as
value.

                          Purpose of dasteplook()
```

```
dasteplook() is designed for use in a macro which controls the use of
macros dastepsetup(), daentervar(), daremovevar() to carry out an entire
stepwise dependent variable selection.

                              Cross references
See also topics dastepsetup(), daentervar(), daremovevar() and
dastepstatus().
```

7.8 dastepsetup()

Usage:

```
   dastepsetup([Model] [,allin:T or in:logvec] [,silent:T]), Model a
     CHARACTER scalar glm model, usually of the form "y = groups"
```

Keywords: classification, discrimination, stepwise

```
                           Purpose of dastepsetup()
You use macro dastepsetup() at the start of a forward or backward
stepwise selection of dependent variables in a discriminant analysis or
more generally in a multivariate linear model.

What it actually does is create and initialize invisible variable
_DASTEPSTATE which encapsulates information on which dependent
variables are "in" and which are "out" at any stage of the variable
selection process. See topic '_DASTEPSTATE'.

                             Linear discrimination
The most common use is in stepwise linear discriminant analysis where
you are trying to select a subset of reponse variables that effectively
discriminate among two or more groups.  It can also be used in any
linear model when you are trying to select a subset of reponse
variables that are responsible for any violation of the overall null
hypothesis H0: all model coefficients except constant term are 0.

                                    Usage
dastepsetup(Model), where Model is a CHARACTER scalar specifying a GLM
model, initializes _DASTEPSTATE so that no variables are "in" and all
are "out".  This is appropriate at the start of forward stepwise
dependent variable selection.  In linear discrimination analysis, Model
has the form "y = groups", where groups is a factor defining the groups
to be discriminated.

                                Printed output
A report of the current status is printed.  This includes all the
F-to-enter statistics and their P-values.

                                Value returned
A copy of _DASTEPSTATE is returned as an "invisible" variable which can
be assigned but is not automatically printed.

                                Keyword silent
```

```
dastepsetup(Model, silent:T) does the same, except the printed report
is suppressed.

                              Default model
dastepsetup([,silent:T]) does the same, except variable STRMODEL,
usually the most recent GLM model used, is taken as Model.

                              Keyword allin
dastepsetup([Model], allin:T [,silent:T]) does the same, except that
all response variables are "in" and no variables are "out".  Component
'history' of _DASTEPSTATE is initialized to run(p), where p is the
number of variables.

                               Keyword ins
dastepsetup([Model], in:ins [,silent:T]), where ins is a LOGICAL vector
of length p, does the same, except only variables j1, j2 , ... are "in"
where ins[j1], ins[j2] ... are T and the remaining elements are F.
Component 'history' is initialized to vector(j1,j2,...).

                            What you do next
After dastepsetup(), your next step is to use daentervar() to enter a
new variable or daremovevar() to remove a variable.  The choice of which
variable to enter or remove is usually made on the basis of the
F-to-enter and/or F-to-remove statistics in the printed report.

                            Cross references
See also topics daentervar(), daremovevar(), dastepstatus() and
dasteplook().
```

7.9 _DASTEPSTATE

Keywords: classification, discrimination, stepwise

```
                              Description
_DASTEPSTATE is an invisible variable which encapsulates the current
state of a process of stepwise response variable selection for a
multivariate linear model.  The model is most commonly of the form "y =
groups", groups a factor, and the stepwise process corresponds to
stepwise linear discriminant analysis.  You normally don't need to be
concerned with _DASTEPSTATE itself, since all interaction with it can
be done using macros.

In the following, p = number of variables, E = p by p error matrix from
manova() and H = p by p hypothesis matrix.  For a model of the form "y
= groups", H = matrix(SS[2,,]) and E = matrix(SS[3,,]).  H excludes
contributions from a constant term.  See topic 'models'

At a particular stage there are from 0 to p "in" variables, variables
tentatively considered important in rejecting the null hypothesis H0
associated with H; the remainder are "out" and either have not yet been
considered or are tentatively considered unimportant in rejecting H0.
```

```
At each stage there is an F-to-enter statistics for each "out"
variable, if any.  This is the F-statistic in an analysis of covariance
of the variable with the "in" variables as covariates.  When no
variables are "in", this is just the usual ANOVA F-statistic for the
variable.

Similarly, at each stage, there is an F-to-remove statistic for each
"in" variable, if any.  This is the F-to-enter statistic for the
variable that would be computed if it were to be removed and turned
into an "out" variable.

                        Contents of _DASTEPSTATUS
_DASTEPSTATUS has the form
 structure(model:glmModel, hpluse:(H+E)swept, e:Eswept, in:ins,\
     F:F_stats, fh:hypDF, fe:errDF, history:hist)

  glmModel      CHARACTER scalar specifying a GLM model, usually of the
                form "y = groups", groups a factor
  ins           LOGICAL vector of length p with ins[j] = T when variable
                j is "in"
  (H+E)swept    H+E when no variables are "in"; swp(H+E,run(p)[ins]),
                otherwise;  that is any "in" variables have been swept
  Eswept        E when no variables are "in"; swp(E,run(p)[ins])
                otherwise
  F_statistics  A REAL vector of F-statistics, with F[ins] being values
                of F-to-remove and F[!ins] being values of F-to-enter
  hypDF         Numerator d.f. of F-to-enter and F-to-remove = fh, the
                degrees of freedom associated with H
  errDF         Denominator d.f of F_statistics = fe - k - 1 for "in"
                and fe - k otherwise, where fe are the error d.f. and
                k = sum(ins) = number of "in variables"
  hist          An integer vector summarizing the path followed to
                reach the current state.  when hist[i] = j > 0, variable
                j was entered at step i; when hist[i] = -j < 0, variable
                j was removed at step i.

                        When to use dastepsetup()
You normally use macro dastepsetup() to initialize _DASTEPSTATUS.
dastepsetup() uses manova() to find E, H, fh and fe.  When variables j1,
j2, ... jk are specified as being "in", hist is initialized to
vector(j1,...,jk); when no variables are "in" at the start, hist is
NULL.  dastepsetup() calls stepstatus to compute component 'F' and
optionally report on the initial status.

                           What you do next
You use macros daentervar() and daremovevar() to update _DASTEPSTATUS by
changing an "out" variable to an "in" or vice versa.  These optionally
print a report of the new status.

You use macro dastepstatus() to compute component F and optionally print
a report of the information in _DASTEPSTATUS.
```

```
You use macro dasteplook() to extract components of _DASTEPSTATUS
without changing it.

Macros daentervar(), daremovevar(), dastepsetup() and dasetupstatus()
return the new value of _DASTEPSTATUS as an "invisible" result which can
be assigned but is not printed.

                            Cross references
See also topics dastepsetup(), daentervar(), daremovevar(),
dasteplook() and dastepstatus().
```

7.10 dastepstatus()

Usage:

```
dastepstatus([silent:T])
```

Keywords: classification, discrimination, stepwise

```
                                  Usage
dastepstatus() computes and prints out the values of F-to-enter or
F-to-remove, and their P-values at the current stage of stepwise
dependent variable selection.  You can use this information to select
the next variable to be entered (the one with the largest F-to-enter)
or remove (the one with the smallest F-to remove).  Component
'F_statistics' of variable _DASTEPSTATE is updated.

                              Value returned
dastepstatus() returns as value an "invisible" copy of variable
_DASTEPSTATE.  This can be assigned but is not normally printed.
_DASTEPSTATE must have been previously initialized or updated by macros
dastepsetup(), daentervar() or daremovevar().

                              Keyword silent
dastepstatus(silent:T) sets component 'F_statistics' of _DASTEPSTATE
and returns an invisible copy of _DASTEPSTATE but does not print the
F-statistics.

You ordinarily don't need to use dastepstatus() directly since macros
daentervar(), daremovevar() and dastepstatus() all use dastepstatus() to
report F-statistic values and their P-values.  An exception is when you
want to enter or remove several variables at once and want to see a
report only at the end.  Suppose, for example, that you want to add
variables 2 and 4 together, but don't want to see a report after
variable 2 is entered.

                                  Example
  Cmd> daentervar(2,4,silent:T) # silently enter variables 2 and 4

  Cmd> dastepstatus() # print report
```

```
                          Cross references
See also topics dastepsetup(), daentervar(), daremovevar(), dasteplook()
and '_DASTEPSTATE'.
```

7.11 discrim()

Usage:

```
   discrim(groups, y), vector of positive integers groups, REAL matrix y
     with no MISSING values
```

Keywords: classification, discrimination

```
                                   Usage
discrim(groups, y), where groups is a factor or an integer vector, and
y is a REAL data matrix with no MISSING elements, computes the
coefficients of linear discriminant functions that can be used to
classify an observation into one of the populations specified by
argument groups.

The functions being estimated are optimal in the case when the
distribution in each population is multivariate normal and the
variance-covariance matrices are the same for all populations.

When there are g = max(groups) populations, and p = ncols(y) variables,
the value returned is structure(coefs:L, add:C) where L is a REAL p by
g matrix and C is a 1 by g row vector.

If y is a length p vector of data to be classified to one of the
populations, then f = L' %*% y + C' is the vector of discriminant
function values (scores) for the g populations.

If f[j] = max(f) is the largest element of f, then, assuming the g
populations are equally probable (each have prior probability 1/g), then
population j is the most probable population based on y.

                    Prior and posterior probabilities
If P is a length g vector such that P[j] = prior probability a randomly
selected case belongs to population j, then the estimated
posterior probability that y belongs to population k is
   P[k]*exp(f[k])/sum(P * exp(f)) =
            P[k]*exp(f[k] - f[1])/sum(P * exp(f - f[1]))
The second form is preferred since exp(f[k]) can be too large to
compute.

                       Classifying rows of matrix
When Y is a m by p data matrix whose rows are to be classified, F = Y
%*% L + C is m by g matrix, with F[i,j] containing the value of the
discriminant function for population j evaluated with the data in row i
of Y.  A m by g matrix of posterior probabilities for each group and
case can be computed by
       P * exp(F  - F[,1])/((P * exp(F - F[,1])) %*% rep(1,g))
```

```
                Bias in estimating posterior probabilities
It is well known that posterior probabilities computed for a case that
is in "training set", the data set from which a classification method
was estimated, are biased in an "optimistic" direction: The estimated
posterior probability for its actual population is biased upward.  For
this reason posterior probabilities should be estimated only for cases
that are not in the training set.  See macro jackknife() for a partial
remedy.
```

7.12 discrimquad()

Usage:

```
   discrimquad(groups, y), factor or vector of positive integers groups,
     REAL matrix y with nrows(y) = length(groups)
```

Keywords: classification, discrimination

```
                                      Usage
discrimquad(groups, y), where groups is a factor or an integer vector,
and y is a REAL data matrix, computes the coefficients of quadratic
discriminant functions that can be used to classify an observation into
one of the populations specified by argument groups.

It is an error if the smallest group has p or fewer members or if y has
any MISSING elements.

                                 Value returned
When there are g = max(groups) populations, and p = ncols(y) variables,
the value returned is structure(Q:q, L:l, addcon:c, grandmean:ybar),
where the components are as follows:
   q     structure(Q1,Q2,...Qg), each Qj a  REAL p by p matrix
   l     structure(L1,L2,...Lg), each L2 a REAL vector of length p
   c     vector(c1,c2,...cg), cj REAL scalars
   ybar  vector(ybar1,...ybarp), the vector of column means

                                 Quadratic score
When x is a vector of length p to be classified, the quadratic score
for group j is
    qs[j] = (x-ybar)' %*% q[j] %*% (x-ybar) + (x-ybar)' %*% l[j] + c[j]

The functions are optimal in the case when the distribution in each
population is multivariate normal with no assumption that the variance-
covariance matrices are the same for all populations.

                       Prior and posterior probabilities
When P = vector(P1,P2,...,Pg) is a vector of prior probabilities a
randomly selected case comes from the various populations, then the
posterior probabilities the elements of the vector
   P*exp(qs)/sum(P*exp(qs)) = P*exp(qs - qs[1])/sum(P*exp(qs - qs[1])
```

```
The latter form is usually preferable since it is possible for
exp(qs[1]) to be so large as to be uncomputable.  These probabilities
can be computed using macro probsquad().

               Bias in estimating posterior probabilities
It is well known that posterior probabilities computed for a case that
is in "training set", the data set from which a classification method
was estimated, are biased in an "optimistic" direction: The estimated
posterior probability for its actual population is biased upward.  For
this reason posterior probabilities should be estimated only for cases
that are not in the training set.

                              Cross references
See also discrim() and probsquad().
```

7.13 distcomp()

Usage:

```
distcomp(y), REAL matrix y with no MISSING values
```

Keywords: covariance, descriptive statistics

```
                                    Usage
distcomp(y) computes the generalized distances of each row of REAL
matrix y from the mean of all the rows.  y must not have any missing
elements.

When y is n by p, the result is the length nrows(y) vector
        d = diag((y - ybar') ' %*% solve(s) %*% (y - ybar'),
where ybar is the vector of column means and s is the sample covariance
matrix of y.  The individual elements of d[i] are

   d[i] = (y[i,]' - ybar)' %*% solve(s) %*% (y[i,]' - ybar),

Very commonly matrix y is the matrix RESIDUALS computed by manova().

                              Cross references
See also chiqqplot().
```

7.14 facanal()

Usage:

```
facanal(s, m [, method:Meth] [,start:psi0] [,rotate:Rotmeth] \
  [,maxit:nmax] [,minit:nmin] [,crit:vector(nsig1,nsig2,dg)]\
  [,minimizer:M] [,gold:ngold] [,quiet:T] [,silent:T] \ [,recordwhen:d1]
  [,printwhen:d2]), REAL p.d. symmetric matrix s, integer m > 0, psi0
  positive vector of starting uniquenesses, Rotmeth one of "varimax",
  quartimax", "equimax", "none" (default), nmin >= 0, nmax > 0, ngold >
  0, d1 >= 0, d2 >= 0, nsig1, nsig2 integers, dg REAL scalar, M one of
  "dfp", "bfs" (default) or "broyden",
```

Keywords: factor analysis

```
                              Introduction
facanal() uses an optimization macro (dfp(), bfs() or broyden()) to find
ULS, GLS or ML estimates of uniquenesses and loadings.  The optimizer
minimizes a criterion as a function of log(psi), psi = vector of
uniquenesses.

You can specify a rotation method and starting values.

facanal() is to be preferred to other macros, including ulsfactor() and
glsfactor(), for factor extraction.

By default, minimizer bfs() is used but you can specify another
minimizer.

                                  Usage
facanal(r, m, method:Meth) computes estimated uniquenesses and unrotated
estimated loadings for a orthogonal factor analysis based a p by p
correlation or covariance matix r which must be symmetric and positive
definite.

Integer m > 0 is the number of factors assumed.  It must satisfy m <
(2*p + 1 - sqrt(8*p+1))/2

Meth is must be one of "mle" or "ml" (ML or maximum likelihood method),
"uls" (ULS or unweighted least squares method) or "gls" (GLS or
generalized least squares method).  If you omit method:Meth, the default
is method:"mle".

In addition to the uniquenesses and loadings, facanal() prints the
minimized value of the criterion being minimized.  This can be used in a
goodness of fit test.

See below for the value returned by facanal().  It is "invisible" unless
'quiet:T' or 'silent:T' is an argument.

                             Starting values
facanal(r, m, method:Meth, start:psi0), does the same except that the
minimization iteration is started with uniquenesses psi0.  The default
starting values are psi0 = 1/diag(solve(r)).
```

```
                                Rotation
You control rotation of the estimated loadings using keywords 'rotate'
and 'kaiser'.

facanal(r, m, method:Meth, rotate:Rot), where Rot is one of "varimax",
"quartimax", "equimax" or "none" (the default), carries out the
indicated rotation of the factor loadings using Kaiser normalization.
It does not affect the values of the uniquenesses or criterion.

facanal(r, m, method:Meth, rotate:Rot,kaiser:F), does the same except
Kaiser normalization is not done.

                          Optimization control
There are several keywords you can use to control the actual
minimization of the criterion.  Default values are in [..]

  nmin:n1     integer n1 >= 0 = minimum number of interations performed
              [0]
  nmax:n2     integer n2 > 0 = maximum number of interations performed
              [30]
  crit:vector(nsig1,nsig2,dg)
     nsig1    target number of significant digits in log uniquenesses
              [5]
     nsig2    target number of significant digits in the criterion [8]
     dg       target upper limit for ||gradient|| [-1]
              Negative values of nsig1, nsig2 or dg are ignored.
  minimizer:M M = name of minimizer, one of "dfp" (Davidon-Fletcher-
              Powell), "bfs" (Broyden-Fletcher-Shanno) or "broyden"
              ["bfs"]
  ngold:n     integer n >= 0 = number of steps in golden mean linear
              search by minimizer [1]; ignored with minimizer:"broyden"

                             Macros needed
facanal() requires and loads automatically one of the macros ulscrit(),
glscrit() or mlcrit() to compute the objective function, gradient vector
and loading matrix.

                          Printing control
There are three keywords that you can use to control what gets printed.

   quiet:T     suppresses printing of results;  the return value is
               visible
   silent:T    same as quiet:T except warning messages are also
               suppressed
   printwhen:d1  d1 >= 0 integer specifies (when d1 > 0) that partial
               results are printed at iterations d1, 2*d1, 3*d1.  These
               are the current values of x = log(uniquenesses), the F(x)
               = criterion and the gradient vector dF(x)/dx

                          Side effect variables
In addition to returning them, facanal() saves the estimated
uniquenesses, rotated loadings, minimized criterion, eigenvalues and
gradient vector in side-effect variables PSI, LOADINGS, CRITERION,
```

```
EIGENVALUES and GRADIENT, respectively.

Each time they are entered, ulscrit(), glscrit() or mlcrit() save a copy
of their argument psi in invisible variable _PSIULS, _PSIGLS or _PSIML.

You can use keyword 'recordwhen' to specify that partial results are
saved in side-effect structure BFSRECORD, DFPRECORD or BROYDNRECORD on
some or all iterations:

   recordwhen:d2  d2 >= 0 integer specifies (when d2 > 0) that partial
               results are saved at iterations d2, 2*d2, 3*d2, ...
               Values of x = log(uniquenesses), F(x) = criterion and
               dF(x)/df go in components 'xvalx', 'funval' and
               'gradients' of the structure.

                              Value returned
facanal() always returns as structure a value which can be assigned.
Unless 'silent:T' or 'quiet:T' is an argument, the return value is
"invisible" and won't be automatically printed.

The result has the form structure(psihat:psi,loadings:l,criterion:crit,
eigenvals:vals, gradient:g, method:Meth, rotation:Rot, iter:n, status:k)
where the values of the components are as follows:

   psi         REAL vector of estimated uniquenesses
   l           REAL p by m matrix of loadings
   crit        Minimized criterion
   vals        eigenvalues s relative to psi
   g           REAL gradient vector at optimum
   n > 0       integer = number of iterations
   k >= 0      integer specifying which (1, 2 or 3) of the convergence
               criterion signalled convergence; k = 0 means not
               converged

                                 Examples
  Cmd> facanal(cor(y),2,rotate:"varimax") # y a 50 by 5 matrix
  Convergence in 26 iterations by criterion 2
  estimated uniquenesses:
  (1)     0.39881  5.4107e-07     0.20203     0.32915     0.63907
  varimax rotated estimated loadings:
  (1,1)     0.71121     0.30883
  (2,1)     0.13876     0.99033
  (3,1)    -0.84817    -0.28032
  (4,1)    -0.80203    -0.16613
  (5,1)     0.59678   -0.069151
  minimized ml criterion:
  (1)    0.013439

  Cmd> result <- facanal(cor(y),2,rotate:"varimax",method:"uls",\
        silent:T)

  Cmd> compnames(result)
  (1) "psihat"
```

```
  (2) "loadings"
  (3) "criterion"
  (4) "eigenvals"
  (5) "gradient"
  (6) "method"
  (7) "rotation"
  (8) "iter"
  (9) "status"

  Cmd> result[vector(8,9)] # status = 0 => did not converge
  component: iter
  (1)          30
  component: status
  (1)           0

                            Cross references
See also ulsfactor(), glsfactor(), ulscrit(), glscrit(), mlcrit(),
minimizer().
```

7.15 forstep()

Usage:

```
   forstep(i,H,E,fh,fe), integer i > 0, fh > 0, fe > 0, REAL symmetric
     matrices H and E with no MISSING values
```

Keywords: factor analysis, iteration

```
NOTE: This macro is OBSOLETE and is retained only for backward
compatibility because it was in file MacAnova.mac in earlier versions of
MacAnova.  For doing stepwise variable selection in discriminant
analysis you should use newer macros dastepsetup(), daentervar(),
daremovevar(), dastepstatus() and dasteplook().

                            What forstep() does
Macro forstep() performs a variable inclusion step in forward stepwise
variable selection in linear discriminant analysis.

forstep() is intended to be used after you have used manova("y =
groups"), where y is a data matrix and groups is a factor, to compute
hypothesis and error matrices H = matrix(SS[2,,]) and E =
matrix(SS[3,,]), with fh = DF[2] and fe = DF[3] degrees of freedom
respectively.

                            Status information
Status information about the variables currently "in" and "out' is
maintained in integer vectors INS and OUTS containing numbers of
variables currently included and currently excluded.  When no variables
are "in", INS = 0; when all variables are "in", OUTS = NULL.  INS must
be initialized, usually to 0, before forstep() can be used.

                                    Usage
```

forstep(j,H,E,fh,fe), where j is the number of a variable not currently "in", adds j to INS and removes it from OUTS, and then uses macro compf() to compute F-to-enter for all variables included in the updated INS. The Fs-to-enter are the analysis of covariance Fs for each "out" variable, with the "in" variables being used as covariates. See topic compf(). When no variables are "in", the Fs-to-enter are the ordinary ANOVA F-statistics for each variable.

Value returned

The value returned (which will normally be printed if not assigned) is structure(f:F_to_enter, df:vector(fh,fe-k), ins:INS,outs:OUTS), where F_to_enter is the vector of F-to-enter statistics, one for each variable not in INS, INS and OUTs are copies of the status vectors INS and OUTS. k is the number of variables currently "in".

The F-to-enter statistics have nominal degrees of freedom fh and fe - k. The next variable to be entered, if any, is normally the variable with the largest F-to-enter. The decision to enter it is based on the size of F-to-enter.

Example

You can somewhat automate the start of this process as follows:

```
Cmd> manova("y = groups", silent:T) # response matrix y, factor groups

Cmd> H <- matrix(SS[2,,]); E <- matrix(SS[3,,])

Cmd> fh <- DF[2]; fe <-  DF[3]

Cmd> INS <- 0; stuff <- compf(H,E,fh,fe)

Cmd> stuff <- forstep(OUTS[grade(stuff$f,down:T)[1]],H,E,fh,fe);
```

The last step can be repeated to bring "in" variables. Of course, in practice, you want to examine the computed F-to-enter statistics to see if another variable *should* be entered.

Doing a backward step

You can do a backward step (variable deletion) using macro backstep(). One difference between backstep() and forstep() is that backstep() determines the variable to eliminate, and then updates INS and OUTS; you must tell forstep() which variable to include. See backstep() for details. See also compf() which computes F-to-enter for variables not in INS.

Both backstep() and compf() are OBSOLETE and are retained only for backward compatibility.

Cross references

See also manova(), daentervar(), daremovevar(), dastepsetup(), dastepstatus() and dasteplook().

7.16 glscrit()

Usage:

```
result <- glscrit(logpsi, k, structure(r, nfact [, del])), REAL vector
  logpsi, integer scalar k, integer nfact > 0, small REAL scalar del
```

Keywords: factor analysis

```
                                Usage
glscrit() is used by facanal() with method:"gls" to compute the
generalized least squares (GLS) criterion F_gls(x), x = log(psi) being
minimized, its gradient dF_gls(x)/dx and the current loadings and
eigenvalues.

trace((I - solve(r) %*% rhat) %*% (I - solve(r) %*% rhat))

result <- glscrit(logpsi, k, structure(r, nfact [,del]) computes a
scalar, vector or structure, depending on the value of integer k.

r is a REAL symmetric covariance or correlation matrix, nfact > 0 is the
number of factors, and del >= 0 is a small REAL scalar (default is 1e-5)
controlling how the gradient is computed.

When del = 0 (value used by facanal()), the gradient is computed
analytically.

When del > 0, the gradient vector has elements gradient[i] = (F_gls(x) +
del*(run(p)==i)) - F_gls(x))/del.

                               Result
The result computed is as follows:
   k           result
   0           scalar F_gls(x), x = logpsi
   1           vector gradient dF_gls(x)/dx
  -2           structure(loadings, eigenvals)

When r is not positive definite, MISSING (k = 0) or rep(MISSING,
nrows(s)) (k = 1) is returned.

No argument checking is done.

                           Cross references
See also facanal(), mlcrit(), ulscrit()
```

7.17 glsfactor()

Usage:

```
glsfactor(s, m [, quiet:T,silent:T,start:psi0,uselogs:T], maxit:nmax,
  minit:nmin, print:T,crit:crvec]), REAL symmetric matrix s with no
  MISSING values, integers nmax > 0 and nmin > 0, crvec =
  vector(numsig, nsigsq, delta)
```

Keywords: factor analysis

```
                                  Usage
You use glsfactor() to find GLS (generalized least squares) estimates of
the vector psi of factor analysis uniquenesses and loading matrix L by
means of nonlinear least squares.  It uses macro glsresids() to compute
residuals used by non-linear least squares macro levmar().

glsfactor(r, m) estimates uniquenesses and loadings for a m-factor
analysis based on p by p symmetric correlation or variance-covariance
matrix r.  m > 0 must be an integer < (2*p + 1 - sqrt(8*p+1))/2.  The
default starting value for the uniquenesses is psi0 = 1/diag(solve(r)).

glsfactor() prints the estimated uniquenesses psihat and loading matrix,
the minimized criterion value, and vals, a vector of numbers computed
from certain eigenvalues; see below.

                              Value returned
The result is an "invisible" (assignable but not automatically printed)
variable of the form
   structure(psihat:psi,loadings:L,criterion:crit,eigenvals:vals,\
      status:iconv,iter:n)
The values of these components are as follows

   psi     length p vector of estimated uniquenesses
   L       p by m matrix of estimated loadings satisfying
           L' %*% dmat(1/psi) %*% L is diagonal
   crit    sum of squared "residuals" of r from rhat
   vals    1/v - 1, where v is the vector of eigenvalues of
           dmat(psi) %*% solve(r) in increasing order
   iconv   integer >= 0 indicating convergence status, with 0 and 4
           indicating non-convergence
   n       number of iterations

                         Keyword phrase arguments
There are several keyword phrases that can be used as arguments to
control the behavior of glsfactor().
  start:psi0     psi0 a REAL vector of length p with min(psi0) > 0
                 to be used as starting values for the iteration
  uselogs:T      log(psi) is used in the iteration instead of psi;
                 this ensures psi does not become negative and may
                 speed convergence
  maxit:nmax     No more than nmax iterations are to be used; the
                 default is 30
  minit:nmin     At least nmin iterations will be performed; the
                 default is 1
  crit:crvec     crvec = vector(numsig, nsigsq, delta), 3 criteria
                 for convergence; a negative criterion is ignored;
                 see macro levmar() for details
  quiet:T        uniquenesses, loadings and vals are not printed;
                 only certain warning messages are printed
  silent:T       nothing is printed except error messages
```

```
  print:T        macro levmar() will print a status report on every
                 iteration

                      Convergence status indicator
Values of iconv indicate the following situations
 iconv  Meaning
   0    Not converged
   1    Converged with relative change in psi or log(psi) < 10^-numsig
   2    Converged with relative change in rss <= 10^-nsigsq
   3    Converged with ||gradient|| < delta
   4    Interation stopped; could not reduce rss.

                                  Caveats
Without uselogs:T, there is no guarantee that min(psihat) > = 0.

Even with uselogs:T, there is no guarantee that the minimum is inside
the permissible range (r - dmat(psihat) positive semi-definite);
however, this often leads to an 'argument to solve() singular' error
message.

                        Invisible variable _PSIGLS
Everytime it is called by levmar(), macro glsresids() saves a copy of
the current value of psi in invisible variable _PSIGLS.  Type
'print(_PSIGLS)' to see this value.

glsfactor() is very much a work in progress, that may in fact be
abandoned in favor of other methods using optimization macros in macro
file Math.mac distributed with MacAnova.

                             Cross references
See also ulsfactor() and glsresids().
```

7.18 glsresids()

Usage:

glsresids() is used by macro glsfactor() to do GLS factor extraction by a nonlinear least squares method.

Keywords: factor analysis

```
                                  Usage
glsresid(theta,0,vector(r),m) returns a length p^2 "residual vector"
computed from but not identical with dmat(p,1) - solve(r) %*% rhat,
where r is a REAL p by p correlation or variance-covariance matrix.  m >
0 is an integer specifying the number of factors to extract.

When __USELOGSGLS, a LOGICAL scalar defined by glsfactor(), has value
False, argument theta is interpreted as psi, the length(p) REAL vector
of uniquenesses.  When __USELOGSGLS is True, theta is interpreted as
log(psi).  glsresid saves a copy of psi in variable _PSIGLS.
```

```
rhat = dmat(psi) + V, where V is the best rank m approximation to r -
dmat(psi) in a generalized least squares sense.

glsresids() is used by macro glsfactor() which does generalized least
squares (GLS) factor extraction.
```

7.19 goodfit()

Usage:

```
   goodfit(s,lhat,psihat), REAL symmetric matrix lhat REAL vector or
     matrix, psihat REAL vector, all with no MISSING values
```

Keywords: factor analysis

```
                                  Usage
goodfit(r, L, psi), where r is a p by p symmetric correlation or
variance-covariance matrix, L is a REAL p by m matrix of purported
factor loadings, and psi is a length p REAL vector of factor analysis
uniquenesses computes three measures of lack of fit of r to the factor
analytic approximation rhat = L %*% L' + dmat(psi).

If dev = vector(r - rhat) is the length p^2 vector of deviations of r
from rhat, then goodfit() returns
       vector(max(abs(dev)), sum(dev^2), sum(abs(dev)))

When L and psi have been computed by some factor analysis estimation
method, and one or more of the elements of the result is large it may
indicate lack of fit of r to the factor analytic model.

These are intended to describe how bad the fit is, not to test it.
```

7.20 groupcovar()

Usage:

```
   groupcovar(groups, y), factor or vector of positive integers groups,
     REAL matrix y with no MISSING values
```

Keywords: covariance, descriptive statistics

```
                                  Usage
groupcovar(G, y), where G is a length N factor or vector of positive
integers and y is a REAL N by p data matrix with no MISSING values,
computes group means and the pooled sample covariance matrix for the
groups defined by the elements of G.  It is an error if no group has at
least 2 members.

                              Value returned
The result is structure(n:n, means:ybar, covariance:V).
  n = vector(n1, ...,ng) is the vector of group sample sizes, where g =
```

```
      max(G).
  ybar is a g by p REAL matrix with ybar[i,] = sample mean for
      group i.  When group i is empty, ybar[i,] is all MISSING.
  V is the pooled variance-covariance matrix.  When all the groups are
      non-empty, V = ((n1-1)*S1 + (n2-1)*S2 + ... + (ng-1)*Sg)/(N - g)
      where S1, ..., Sg are the sample variance-covariance matrices for
      the g groups.

                              Cross reference
See also tabs().
```

7.21 hotellval()

Usage:

```
   hotellval(x [, pval:T]), REAL matrix x with no MISSING elements
```

Keywords: hotelling tsq, hypothesis tests

```
                                    Usage
hotellval(x), where x is a n by p REAL matrix with no MISSING elements,
returns a REAL scalar containing Hotelling's T^2 statistic for testing
the null hypothesis H0:  mu = 0.  mu' is the expectation of each row of
x when the rows are assumed to be random sample from a multi-variate
population.

It is an error if n <= p.

hotellval(x, pval:T) does the same, except a P-value is also computed
under the assumption that the rows of x are independent multivariate
normal.  The result is structure(hotelling:tsq, pvalue:pval), where tsq
and pval are REAL scalars.

Examples:
  Cmd> hotellval(x - mu0', pval:T)

where mu0 is a REAL vector of length p, computes a test statistic and
associated P-value for testing H0: mu = mu0.

  Cmd> hotellval(x1 - x2, pval:T)

where x1 and x2 are both n by p, computes Hotelling's paired P^2
statistic for testing H0: E(x1 - x2) = 0, together with a P-value.

                              Cross references
See also hotell2val(), tval() and t2val().
```

7.22 hotell2val()

Usage:

```
hotell2val(x1, x2 [, pval:T]), REAL matrices x1 and x2 with no MISSING
  elements and ncols(x1) = ncols(x2)
```

Keywords: hotelling tsq, hypothesis tests

```
                                Usage
hotell2val(x1, x2), where x1 and x2 are REAL matrices with n1 and n2
rows and p columns, returns Hotelling's two-sample T^2 statistic for
testing the null hypothesis that the rows of x1 have the same mean as
the rows of x2 when they are considered as independent random samples
from two multivariate populations.  The statistic assumes that the
variance-covariance matrix is the same in the two populations.

It is an error if n1 + n2 <= p.

hotell2val(x1, x2, pval:T) does the same, except it also computes a
P-value under the assumption that the rows of x1 and of x2 constitute
independent random samples from multivariate normal distributions with
the same variance-covariance matrix.  The result is
structure(hotelling:tsq, pvalue:pval).

                            Cross references
See also hotellval(), tval() and t2val().
```

7.23 mlcrit()

Usage:

```
result <- mlcrit(logpsi, k, structure(r, nfact [, del])), REAL vector
  logpsi, integer scalar k, integer nfact > 0, small REAL scalar del
```

Keywords: factor analysis

```
                                Usage
mlcrit() is used by facanal() with method:"mle" to compute the maximum
likelihood ML criterion F_ml(x), x = log(psi) being minimized, its
gradient dF_ml(x)/dx and the current loadings and eigenvalues.

result <- mlcrit(logpsi, k, structure(r, nfact [,del]) computes a
scalar, vector or structure, depending on the value of integer k.

r is a REAL symmetric covariance or correlation matrix, nfact > 0 is the
number of factors, and del >= 0 is a small REAL scalar (default is 1e-5)
controlling how the gradient is computed.

When del = 0 (value used by facanal()), the gradient is computed
analytically.

When del > 0, the gradient vector has elements gradient[i] = (F_ml(x) +
del*(run(p)==i)) - F_ml(x))/del.
```

```
                                Result
The result computed is as follows:
   k           result
   0           scalar F_ml(x), x = logpsi
   1           vector gradient dF_ml(x)/dx
  -2           structure(loadings, eigenvals)

When r is not positive definite, MISSING (k = 0) or rep(MISSING,
nrows(s)) (k = 1) is returned.

No argument checking is done.

                            Cross references
See also facanal(), glscrit(), ulscrit()
```

7.24 jackknife()

Usage:

```
probs <- jackknife(groups,y [,prior:P]), factor or vector of positive
  integers groups, REAL matrix y and positive vector P with no MISSING
  elements
```

Keywords: discrimination, classification

```
                                Usage
jackknife(G, y), where G is a factor or vector of positive integers of
length n and y is a REAL n by p matrix with no MISSING elements, carries
out a jackknife validation of linear discriminant functions designed to
discriminate among the g groups defined by the levels of G.

                          What jackknife() does
When you try to estimate the error rate of a classification method by
counting the errors it makes in classifying the cases in the "training
sample", the data set you are using to estimate the method, your
estimate is biased in an optimistic direction.  That is, the proportion
of cases misclassified in the training sample tends to be smaller than
the proportion of cases misclassified in new sample (validation sample).
jackknife() attempts to avoid this bias by classifying each case in the
training sample with linear discriminant functions computed from all the
other cases in the training sample.  This is the "leave-one-out" method,
sometimes called the Lachenbruch-Mickey method.

                            Value returned
Macro jackknife() returns a n by g+1 matrix probs.

probs[i,j], for j = 1,...,g is an estimate of the posterior probability
that the data in y[i,] were derived from population j.

probs[i,g+1] is an integer from 1 to g indicating the population in
which the case should be classified, that is the population for which
```

```
the posterior probability is largest.

                          Posterior probabilities
For each i, 1 <= i <= n, the posterior probabilities probs[i,j], j =
1,..., g are computed as follows.

The linear discriminant function based on y[-i,], that is using all the
data except row i, and the discriminant functions scores for the data
in y[i,] are computed.  From these the posterior probabilities are
computed assuming equal prior probability 1/g for each of the groups.
Each group is assumed to be multivariate normal with the same variance-
covariance matrix in each group.

Because the discriminant functions used to classify y[i,] are computed
without using y[i,], the method is close to unbiased.

                              Keyword prior
jackknife(G,y,prior:P), where P is a REAL vector of length n with no
MISSING elements, does the same except the posterior probabilities are
computed using P.

                                 Example
Here is how you might use jackknife() to estimate the expected
probability of misclassification, assuming the prior probability that a
randomly selected case comes from population j is P[j].

  Cmd> probs <- jackknife(G, y, prior:P)

  Cmd> n <- tabs(,G,count:T) # vector of sample sizes

  Cmd> misclassprob <- tabs(,G,probs[,g+1],count:T)/n

  Cmd> misclassprob[hconcat(run(g),run(g))] <- 0# set diags to 0

  Cmd> sum(misclassprob' %*% P)

The last line computes the estimated probability case randomly selected
from a group with prior probabilitys P will be misclassified by
linear discriminant functions estimated from y.  misclassprob[i,j] with
i != j is an estimate that a case from population i is misclassifed as
population j.

This version of jackknife() is relatively fast since it computes the
successive leave-one-out discriminant functions by modification of the
discriminant functions using all the data, rather than starting fresh.
```

7.25 mulvarhelp()

Usage:

```
mulvarhelp(topic1 [, topic2 ...] [,usage:T])
mulvarhelp(index:T)
```

Keywords:

```
                                  Usage
mulvarhelp(topicname) prints help on a topic related to file
mulvar.mac.  Usually topicname is the name of a macro in the file.

When quoted, topicname may contain "wildcard" characters "*" and "?".
You can also use help keyword 'key'.  See help() for details.

mulvarhelp(topicname1, topicname2, ...) prints help on more than one
topic.

mulvarhelp(topicname1 [, topicname2 ...], usage:T) prints just a brief
summary of usage for the each topic.
```

7.26 mvngen()

Usage:

```
   mvngen() has been renamed rmvnorm().  Type 'usage(rmvnorm)'.
```

Keywords:

```
                                  Usage
mvngen() has been renamed rmvnorm().  Type 'help(rmvnorm)'.
```

7.27 probsquad()

Usage:

```
   probsquad(x,structure(Q,L,addcon,grandmean) [, prior:P]), REAL matrix x
     with no MISSING elements, argument 2 a structure as returned by macro
     discrimquad(), REAL vector P of prior probabilities with P[i] > 0,
     default = rep(1/g,g)
```

Keywords: classification, discrimination

```
@usage
Suppose info = structure(Q,L,addcon,grandmean) as computed by
discrimquad(groups, y) summarizes quadratic discriminant functions
estimated from a n by p matrix of data from g groups.

probsquad(x, info), where x a REAL vector of length p, returns the
length g vector of posterior probabilities, assuming multivariate
normality of each groups and equal prior probability 1/g for each group.

probsquad(x, info, prior:P), where P is a length g vector with P[j] >
0, does the same using P[j]/sum(P) as prior probability of group j.
```

```
In both these usages x can be a m by p matrix, each row of which is an
observation and probsquad() returns a m by g matrix of posterior
probabilities.

                              Cross reference
See probsquad() for information about the form of info.
```

7.28 rmvnorm()

Usage:

```
   rmvnorm(n , p), integers n > 0, p > 1
   rmvnorm(n, sigma), REAL positive definite symmetric matrix sigma,
     nrows(sigma) > 1
   rmvnorm(n, p or sigma, mu), REAL row or column vector mu with length(mu)
     = p or ncols(sigma)
```

Keywords: random numbers

```
                                    Usage
rmvnorm(n, p), where n > 0 and p > 0 are integer scalars, returns an n
by p matrix whose rows are a random sample from the standard
multivariate normal distribution MVN(0, I_p), where I_p is the p by p
identity matrix.

rmvnorm(n, p, mu) where mu is a REAL row or column vector of length p
does the same, except the rows of the result are MVN(mu, I_p).

rmvnorm(n, sigma [,mu]), where sigma is a positive definite p by p
symmetric matrix with p > 1, does the same, except the rows of the
result are MVN(0, sigma) or MVN(mu, sigma).

When p = 1, use mu + sd*rnorm(n) or mu + sd*rmvnorm(n,1) to generate a
normal sample with mean mu and standard deviation sd.

                              Cross reference
See also rnorm().
```

7.29 standardize()

Usage:

```
   ynew <- standardize(y [,locs [,scales]]), n by p REAL matrix y, locs and
     scales REAL scalars or row or column vectors of length p, defaults
     mean and standard deviations
```

Keywords: transformations

```
                                    Usage
ynew <- standardize(y), where y is a n by p REAL matrix, creates a n by
```

```
p REAL matrix ynew with ynew[i,j] = (y[i,j] - ybar[j])/sd[j], where ybar
and sd are vectors of column means and standard deviations of y.  ynew
will have column means = 0 and column standard deviations = 1.

ynew <- standardize(y,locs), where locs is a REAL row or column vector
of length p, does the same except ynew[i,j] = (y[i,j] - locs[j])/sd[j].

ynew <- standardize(y,locs,scales), where scales is a REAL row or column
vector of length p, does the same except ynew[i,j] = (y[i,j] -
locs[j])/scales[j].

                          Scalar locs or scales
If locs is a scalar, it is expanded to rep(locs, p).

If scales is a scalar, it is expanded to rep(scales, p).

                             Missing values
Any means or standard deviations needed are computed by describe(y).
MISSING elements are ignored except for a warning message.

If any elements of locs are MISSING, they are replaced by the mean of
the corresponding columns of y.

If any elements of scales are MISSING, they are replaced by the standard
deviations of the corresponding columns of y.

                             Cross reference
See also describe().
```

7.30 stepgls()

Usage:

```
stepgls(s, psi, m [, print:T]), REAL positive definite symmetric matrix
  s, REAL vector psi or structure with psi[1] a REAL vector, integer m
  > 0
```

Keywords: factor analysis, iteration

```
                                  Usage
stepgls(r, psi, m) performs one step of an iteration which attempts to
extract m factors in factor analyis.  r is a p by p correlation or
variance-covariance matrix, psi is a REAL vector of trial uniquenesses
of length p with min(psi) > 0 and integer m > 0.

                             Value returned
The value returned is structure(psi:newpsi, loadings:L, crit:criterion),
where newpsi is an updated vector of uniquenesses, L is a p by m matrix
of factor loadings satisfying L' %*% dmat(1/psi) %*% L is diagonal, and
criterion is a value of the criterion being minimized; see below.

                          Side effect variables
```

```
In addition, stepgls() creates side effect variabls PSI, LOADINGS, and
CRITERION containing newphi, L and criterion.

Actually L is the loadings that minimize the criterion for argument
phi, not the output newphi, and criterion is the criterion assocated
with phi and L.

                          What stepgls() does
Each step reduces the generalized least squares (GLS) criterion =
trace((I - solve(r) %*% rhat) %*% (I - solve(r) %*% rhat)), where rhat
has the form rhat = dmat(psi) + V where V is the unique rank m matrix
that minimizes this criterion for given psi.

                            Multiple steps
Argument psi can also have the form structure(psi1, ...) where psi1 is
the vector of uniquenesses.  This means you can do many steps of the
iteration as follows.

  Cmd> psi <- psi0 # psi0 a vector of starting values

  Cmd> for (i,1,500){psi <- stepgls(r,psi,m);;} # 500 steps

stepgls(r, psi, m, print:T) does the same except the updated psi and the
criterion value are printed.  For example,

  Cmd> for (i,1,500){psi <- stepgls(r,psi,m,print:i %% 50 == 0);;}

prints out psi and and the criterion every 50 steps.

                                Caveats
The iteration performed by stepgls() is similar but not identical to
what is known Principal Factor Iteration.  It is not unusual for it to
converge very slowly.  In addition, it is possible for a step to produce
a psi with min(psi) < 0 or r - dmat(psi) not non-negative definite.
When this happens stepgls() aborts.  You can retrieve the most recent
uniquenesses and loadings from variables PSI and LOADINGS.

It is usually preferable to use a method that more directly minimizes
the criterion, such as macro glsfactor().  In addition, one of the
function minimization macros, dfp, bfs or broyden could be used to
minimize the criterion as a function of psi or log(psi).
```

7.31 stepml()

Usage:

```
stepml(s, psi, m [, print:T]), REAL positive definite symmetric matrix
  s, REAL vector psi or structure with psi[1] a REAL vector, integer m
  > 0
```

Keywords: factor analysis, iteration

Usage

stepml(r, psi, m) performs one step of an iteration which attempts to extract m factors in factor analyis. r is a p by p correlation or variance-covariance matrix, psi is a REAL vector of trial uniquenesses of length p with min(psi) > 0 and integer m > 0.

Value returned

The value returned is structure(psi:newpsi, loadings:L, crit:criterion), where newpsi is an updated vector of uniquenesses, L a p by m matrix of factor loadings satisfying L' %*% dmat(1/psi) %*% L is diagonal, and criterion is a value of the criterion being minimized; see below.

Side effect variables

In addition, stepml() also creates side effect variabls PSI, LOADINGS, and CRITERION containing newphi, L and criterion.

Actually L is the loadings that minimize the criterion for argument phi, not the output newphi, and criterion is the criterion assocated with phi and L.

What stepml() does

Each step reduces the likelihood (ML) criterion = log(det(rhat)) - trace(r %*% solve(rhat)) - log(det(r)) - p, where rhat has the form rhat = dmat(psi) + V where V is the unique rank m matrix that minimizes this criterionfor given psi.

Multiple steps

Argument psi can also have the form structure(psi1, ...) where psi1 is the vector of uniquenesses. This means you can do many steps of the iteration as follows.

```
Cmd> psi <- psi0 # psi0 a vector of starting values

Cmd> for (i,1,500){psi <- stepml(r,psi,m);;}
```

Keyword print

stepml(r, psi, m, print:T) does the same except the updated psi and the criterion value are printed. For example,

```
Cmd> for (i,1,500){psi <- stepml(r,psi,m,print:i %% 50 == 0);;}
```

prints out psi and and the criterion every 50 steps.

Caveats

The iteration performed by stepml() is similar but not identical to what is known Principal Factor Iteration. It is not unusual for it to converge very slowly. In addition, it is possible a step to produce a psi with min(psi) < 0 or r - dmat(psi) not non-negative definite. When this happens stepml() aborts. You can retrieve the most recent uniquenesses and loadings from variables PSI and LOADINGS.

It is usually preferable to use a method that more directly minimizes the criterion. At present, none such is distributed with MacAnova.

However, one of the function minimization macros, dfp, bfs or broyden could be used to minimize the criterion as a function of psi or log(psi).

7.32 stepuls()

Usage:

```
stepuls(r, psi, m [, print:T]), REAL positive definite symmetric matrix
  r, REAL vector psi or structure with psi[1] a REAL vector, integer m
  > 0
```

Keywords: factor analysis, iteration

Usage

stepuls(r, psi, m) performs one step of an iteration which attempts to extract m factors in factor analyis. r is a p by p correlation or variance-covariance matrix, psi is a REAL vector of trial uniquenesses of length p with min(psi) > 0 and integer m > 0.

Value returned

The value returned is structure(psi:newpsi, loadings:L, crit:criterion), where newpsi is an updated vector of uniquenesses, L a p by m matrix of factor loadings satisfying L' %*% L is diagonal, and criterion is a value of the criterion being minimized; see below.

Side effect variables

In addition, stepuls() also creates side effect variabls PSI, LOADINGS, and CRITERION containing newphi, L and criterion.

Actually L is the loadings that minimize the criterion for argument phi, not the output newphi, and criterion is the criterion assocated with phi and L.

Each step reduces the unweighted least squares (ULS) criterion = trace((r - rhat) %*% (r - rhat)), where rhat has the form rhat = dmat(psi) + V where V is the unique rank m matrix that minimizes this criterion for given psi.

Multiple steps

Argument psi can also have the form structure(psi1, ...) where psi1 is the vector of uniquenesses. This means you can do many steps of the iteration as follows.

```
Cmd> psi <- psi0 # psi0 a vector of starting values

Cmd> for (i,1,500){psi <- stepuls(r,psi,m);;}
```

Keyword print

stepuls(r, psi, m, print:T) does the same except the updated psi and the criterion value are printed. For example,

```
  Cmd> for (i,1,500){psi <- stepuls(r,psi,m,print:i %% 50 == 0);;}

prints out psi and and the criterion every 50 steps.

                                  Caveats
The iteration performed by stepuls() is also known as Principal Factor
Iteration.  It is not unusual for it to converge very slowly.  In
addition, it is possible a step to produce a psi with min(psi) < 0 or r
- dmat(psi) not non-negative definite.  When this happens stepuls()
aborts.  You can retrieve the most recent uniquenesses and loadings
from variables PSI and LOADINGS.

It is usually preferable to use a method that more directly minimizes
the criterion, such as macro ulsfactor().  In addition, one of the
function minimization macros, dfp, bfs or broyden could be used to
minimize the criterion as a function of psi or log(psi).
```

7.33 ulscrit()

Usage:

```
   result <- ulscrit(logpsi, k, structure(r, nfact [, del])), REAL vector
     logpsi, integer scalar k, integer nfact > 0, small REAL scalar del
```

Keywords: factor analysis

```
                                   Usage
ulscrit() is used by facanal() with method:"uls" to compute the
unweighted least squares (ULS) criterion F_uls(x), x = log(psi) being
minimized, its gradient dF_uls(x)/dx and the current loadings and
eigenvalues.

result <- ulscrit(logpsi, k, structure(r, nfact [,del]) computes a
scalar, vector or structure, depending on the value of integer k.

r is a REAL symmetric covariance or correlation matrix, nfact > 0 is the
number of factors, and del >= 0 is a small REAL scalar (default is 1e-5)
controlling how the gradient is computed.

When del = 0 (value used by facanal()), the gradient is computed
analytically.

When del > 0, the gradient vector has elements gradient[i] = (F_uls(x) +
del*(run(p)==i)) - F_uls(x))/del.

                              Value returned
The result computed is as follows:
   k           result
   0           scalar F_uls(x), x = logpsi
   1           vector gradient dF_uls(x)/dx
  -2           structure(loadings, eigenvals)
```

```
When r is not positive definite, MISSING (k = 0) or rep(MISSING,
nrows(s)) (k = 1) is returned.

No argument checking is done.

                            Cross references
See also facanal(), glscrit(), mlcrit().
```

7.34 ulsfactor()

Usage:

```
   ulsfactor(s, m [, quiet:T,silent:T,start:psi0,uselogs:T], maxit:nmax,
     minit:nmin, print:T,crit:crvec]), REAL symmetric matrix s with no
     MISSING values, integers nmax > 0 and nmin > 0, crvec =
     vector(numsig, nsigsq, delta)
```

Keywords: factor analysis

```
                                    Usage
You use ulsfactor() to find ULS (unweighted least squares) estimates of
the vector psi of factor analysis uniquenesses and the loading matrix L
by means of nonlinear least squares.  It uses macro ulsresids() to
compute residuals required by non-linear least squares macro levmar().

ulsfactor(r, m) estimates uniquenesses and loadings for a m-factor
analysis based on p by p symmetric correlation or variance-covariance
matrix r.  m > 0 must be an integer < (2*p + 1 - sqrt(8*p+1))/2.  The
default starting value for the uniquenesses is psi0 = 1/diag(solve(r)).

                               Printed output
ulsfactor() prints the estimated uniquenesses psihat and loading matrix,
the minimized criterion value, and vals, a vector of numbers computed
from certain eigenvalues; see below.

                               Value returned
The result is an "invisible" (assignable but not automatically printed)
variable of the form
   structure(psihat:psi,loadings:L,criterion:crit,eigenvals:vals,\
      status:iconv,iter:n)
The values of these components are as follows

   psi     length p vector of estimated uniquenesses
   L       p by m matrix of estimated loadings satisfying
           L' %*% L is diagonal
   crit    sum of squared "residuals" = vector(r - rhat)
   vals    v, where v is the vector of eigenvalues of r - dmat(psi)
   iconv   integer >= 0 indicating convergence status, with 0 and 4
           indicating non-convergence
   n       number of iterations

                          Keyword phrase arguments
```

```
There are several keyword phrases that can be used as arguments to
control the behavior of ulsfactor().
  start:psi0     psi0 a REAL vector of length p with min(psi0) > 0
                 to be used as starting values for the iteration
  uselogs:T      log(psi) is used in the iteration instead of psi;
                 this ensures psi does not become negative
  maxit:nmax     No more than nmax iterations are to be used; the
                 default is 30
  minit:nmin     At least nmin iterations will be performed; the
                 default is 1
  crit:crvec     crvec = vector(numsig, nsigsq, delta), 3 criteria
                 for convergence; a negative criterion is ignored;
                 see macro levmar() for details
  quiet:T        uniquenesses, loadings and vals are not printed;
                 only certain warning messages are printed
  silent:T       nothing is printed except error messages
  print:T        macro levmar() will print a status report on every
                 iteration

                      Convergence status indicator
Values of iconv indicate the following situations
 iconv  Meaning
   0    Not converged
   1    Converged with relative change in psi or log(psi) < 10^-numsig
   2    Converged with relative change in rss <= 10^-nsigsq
   3    Converged with ||gradient|| < delta
   4    Interation stopped; could not reduce rss.

                                  Caveat
With or without uselogs:T, there is no guarantee that the solution
found is admissible in the sense that r - dmat(psihat) is positive
semi-definite.  A warning is printed when is not.  With uselogs:T this
situation is sometimes indicated by an 'argument to solve() singular'
error message.

Without uselogs:T, there is no guarantee that min(psihat) > = 0.  A
warning is printed if it is not.

                       Invisible variable _PSIGLS
Every time macro ulsresids() is called by levmar(), it saves a copy of
the current value of psi in invisible variable _PSIGLS.  Type
print(_PSIGLS) to see this value.

ulsfactor() is very much a work in progress that may in fact be
abandoned in favor of other methods using optimization macros in macro
file Math.mac distributed with MacAnova.
```

7.35 ulsresids()

Usage:

```
ulsresids() is used by macro ulsfactor() to do ULS factor extraction by
a nonlinear least squares method.
```

Keywords: factor analysis

```
                                    Usage
ulsresid(theta,0,vector(r),m) returns the residual matrix r - rhat as a
vector of length p^2, where r is a REAL p by p correlation or variance-
covariance matrix.  m > 0 is an integer specifying the number of
factors to extract.

rhat = dmat(psi) + V, where V is the best rank m approximation to r -
dmat(psi) in the least squares sense.

ulsresids() is used by macro ulsfactor() to do unweighted least squares
(ULS) factor extraction.

                                 Use of logs
When __USELOGSULS, a LOGICAL scalar defined by ulsfactor(), has value
False, argument theta is interpreted as psi, the length(p) REAL vector
of uniquenesses.  When __USELOGSULS is True, theta is interpreted as
log(psi).  ulsresids() saves a copy of psi in variable _PSIULS.
```

Chapter 8

Regression Macros Help File

This Chapter contains help for the set of macros related to regression analysis and linear models distributed with MacAnova in the file Regress.mac.txt. The material here is a reformatting of the help in file Regress.mac.txt.

8.1 anovapred()

Usage:

```
anovapred(a,b,...), a, b, ... all the factors in STRMODEL
```

Keywords: glm, anova, standard error, confidence limits, prediction limits

```
                                  Usage
anovapred(a,b,...), where a, b, ... are all the factors in the most
recent GLM model, computes the fitted (predicted) value, the standard
error of estimation, and the standard error of prediction for each
cell. The result is a structure with components 'estimate', 'SEest' and
'SEpred', each of which is a vector, matrix, or array with dimensions
derived from the number of levels of a, b, ....  It uses side effect
variables DEPVNAME, RESIDUALS, and HII.

When the most recent GLM model was manova() with a p-dimensional
dependent variable, each component will have an extra dimension of size
p.

When the most recent GLM model included variates (non-factors), or when
you do not include all factors in the argument list, the results will
probably be wrong, although no warning message will be printed.

anovapred() is implemented as a pre-defined macro.

                            Cross references
See also predtable(), regpred(), 'glm'.
```

8.2 betalimits()

Usage:

```
betalimits(Term, level), Term a CHARACTER scalar, a positive integer or
  a variable in the most recent regression model, REAL positive scalar
  level < 1.
```

Keywords: regression, confidence limits

```
                                  Usage
You use betalimits() to compute confidence limits for a regression
coefficient.  Its use must be preceded by a GLM command, usually
regress() or anova().

betalimits(Term, Level), where Term specifies a variable or a term in
the ANOVA table, and Level is a REAL scalar between 0.5 and 1, returns
upper and lower confidence limits for a regression coefficeint or ANOVA
main effects or interactions.

Term can be a quoted or unquoted predictor variable or term name, or
CONSTANT or a positive integer specifying the number of the term.  If
the term is an iteraction it must be quoted.

Level is the desired confidence coefficient.  If level < .5, a warning
message is given and the confidence coefficient is assumed to be 1 -
level.

After regress(), the result is vector(lower, upper), where lower and
upper are the confidence limits.

After anova(), when the term is a main effect with k levels, the result
is k by 2 matrix hconcat(lower, upper).  When the term is an
interaction of factors with k1, k2, ... levels, the result is
array(vector(upper,lower),k1,k2,...,2).

Example:
After regress("y=x1 + x2 + x3"), the following are all equivalent
  Cmd> betalimits(x2, .95)
  Cmd> betalimits("x2",.95)
  Cmd> betalimits(3,.95)  #counting the constant, x2 is the third term

They all return a vector of length 2.

If a and b are factors with 3 and 4 levels, respectively, after
anova("y = a*b") (equivalent to anova("y = a + b + a.b"))
  Cmd> betalimits("a.b",.95) # quotes are required
and
  Cmd> betalimits(4, .95) # a.b is term 4
are equivalent and return a 3 by 4 by 2 array with a[i,j,] the upper
and lower limits for the i,j interaction effect.

In all these replacing .95 by .05 gives the same result except a
warning message is printed.
```

```
                            Cross references
See also coefs(), secoefs(), regress()
```

8.3 entervar()

Usage:

```
   entervar(var1 [,var2 ...] [,silent:T]), var1, var2, ... the names or
     numbers of independent variables in the complete stepwise model but
     not in the current stepwise model
```

Keywords: stepwise regression, regression

```
                                    Usage
entervar(NewVar) enters independent variable Var to the current stepwise
regression model.  NewVar can either be a quoted name ("z3"), an
unquoted name (z3) or the number of a variable in the complete stepwise
model but not in the current stepwise model. Thus if the full model is
"y=x1+x2+x3+x4+x5", entervar(x2), entervar("x2") and entervar(2) are
equivalent.

It is an error if NewVar is not an independent variable in the complete
stepwise model or if it has already been entered in.

Invisible variable _STEPSTATUS is updated to reflect the changed model.
See topic '_STEPSTATUS'.

                                Printed output
The F-to-remove statistics with P-values are printed for all the
variables in the model, including NewVar, and F-to-enter statistics
with P-values are printed for any variables in the full model that have
not yet been entered.

In addition, entervar() prints an overall F statistic and its P-value,
Mallow's Cp statistic, adjusted R^2 and R^2.  The F-statistic tests
the null hypothesis that the coefficients of the "in" variables are 0.
Because the "in" variables have been selected because they appear to
contributed importantly to the regression, the P-value should not be
interpreted literally.

                                Value returned
The value, which can be assigned (stuff <- entervar(x3)) but is not
printed, is a copy of the updated invisible variable _STEPSTATUS.

                            Entering several variables
entervar(Var1, Var2 ...) does the same except that more than one
variable is entered.  The model and other statistics are printed after
each variable is entered.  The value returned is _STEPSTATUS after all
have been entered.  An example when the full model is "y=x1+x2+x3+x4+x5"
might be entervar(x1,"x2",4).  This would enter x1, x2 and x4 in that
order.
```

```
                              Keyword silent
entervar(Var1 [, Var2 ...], silent:T) does the same, except that the
model and summary statistics are not printed.  You can then use
stepstatus() to print the summary statistics for the new current state
of the stepwise regression process.

                            Cross references
See also removevar(), stepsetup(), stepstatus()
```

8.4 estimlimits()

Usage:

```
estimlimits(x, confLevel), REAL vector or matrix x with no MISSING
  elements, 0 < confLevel < 1 a REAL scalar
estimlimits(NULL,factorValues, confLevel), REAL vector or matrix
  factorValues with no MISSING elements
estimalimits(x,factorValues, confLevel)
```

Keywords: regression, confidence limits

```
Macro estimlimits() computes confidence limits for the expected value
E(y|x) after a regression.  You can also use it to compute limits for
the expectation of y after anova() when the model includes one or more
factors.

Before using estimlimits(), you must have run regress("y=x") or
regress("y=x1 + x2 + ...  xk") or more generally anova(Model) where
Model may contain both one or more predictors (covariates) and/or one
or more factors

When the arguments to estimlimits() specify a single condition under
which limits are wanted for E(y), the result is vector(lower,upper),
where lower and upper are the limits.

When the arguments specify several conditions (several values of x
and/or several factor levels), the result is hconcat(lower,upper),
where lower and upper are vectors with length(lower) = length(upper) =
number of conditions.

                        Usage after regress()
estimlimits(x, confLevel), where REAL variable x is a scalar (simple
linear regression) or a vector (multiple regression) returns
vector(lower, upper), the confidence limits for E(y|x) for the
specified value of of the predictor variable(s).  x can contain no
MISSING elements and confLevel must be a real scalar between 0 and 1.

When there is more than one predictor variable, length(x) = number of
predictors.

When there is only one predictor (simple linear regression), x can be a
vector.  When there is more than one predictor, x can be a matrix with
```

```
ncols(x) = number of predictors.  In both cases, the result is
hconcat(lower,upper).

When confLevel < .5, a warning message is printed and the confidence
level is assumed to be 1 - confLevel.

            Usage after anova() with factors in the model
estimlimits(NULL,FactorValues,confLevel) is appropriate after anova()
with a model with no variates (covariates).

When there is just one factor ("y = a"), FactorValues should be a
scalar or vector of permissible factor levels.  When there are nFactors
> 1 factors, FactorValues should be a vector with length(FactorValues) =
nFactors, or a matrix with ncols(FactorValues) = nFactors.

estimlimits(x,FactorValues,confLevel) is appropriate after anova() with
a model containing both factors and (co)variates.  x and FactorValues
are as in the no factor case and no variate case, respectively.

                        Examples after regress()
After regress("y=x1 + x2 + x3"):
 estimlimits(vector(2,3,4),.95) returns where lower and upper are the
 limits when x1 = 2, x2 = 3 and x3 = 4.

 estimlimits(hconcat(x01, x02, x03),.95), returns hconcat(lower,upper),
 where lower[I] and upper[I] are limits whenx1 = x01[I], x2 = x02[I]
 and x3 = x03[I]. x01, x02, and x03 must all be vectors of the same
 length.

                         Examples after anova()
After anova("y = a + b")
  estimlimits(NULL,vector(1,2),.95) returns vector(lower,upper), where
  lower and upper are limits when a = 1 and b = 2

 After anova("y = x + a + b")
  estimlimits(vector(3,4),vconcat(vector(1,2)',vector(1,3)'),.95)
  returns hconcat(lower,upper), where lower[1] and upper[1] are limits
  when x = 3, a = 1 and b = 2 and lower[2] and upper[2] are limits
  when x = 4, a= 1 and b = 3.

                            Cross references
See also predlimits(), regpred(), glmpred().
```

8.5 nlreg()

Usage:

```
   nlreg(b,x,y,f,param [,deriv:deriv,crit:vec,active:active,maxit:itmax,\
     minit:itmin, print:T, keep:T, quiet:T])
   nlreg(b,x,y param ,resid:res [,deriv:deriv,crit:vec,active:active,\
     maxit:itmax,minit:itmin,print:T,keep:T, quiet:T])
    b      REAL vector of starting values for the iterative fitting
    x      REAL matrix; with argument f, nrows(x) = nrows(y); with
           'resid:res' nrows(x) = nrows(y) can be omitted; if x is not
           used by res(), it should be 0
    y      REAL vector
    f      macro: f(b,x,param) computes a vector of fitted values for y
           (not allowed with 'resid:res'; required without
           'resid:res')
    param  a vector or structure of additional parameters for f() or NULL
    res    macro: res(b,x,y [,param]) computes a vector of residuals from
           a function defined or directly referenced in res() (not allowed
           when f is an argument; required when f is not an argument).
    deriv  optional macro: deriv(b,x,y,param,j) computes derivative of
           f(x,b,param) or -res(b,x,y) with respect to b[j], returning a
           vector the same length as y
    active LOGICAL vector the same length as b specifies which parameters
           are included in iteration
    vec    vector(numsig, nsigsq, delta): 3 criteria for convergence
           numsig = number of digits of accuracy in coefficients
           nsiqsq = number of digits of accuracy in residual SS
           delta  = threshhold for norm of gradient
    itmin  the minimum number >= 0 of iterations performed
    itmax  the maximum number >= itmin of iterations allowed
    print  If T, partial results printed at each iteration
    keep   If T, nlreg() returns the structure returned by levmar()
           with components, coefs, hessian, jacobian, gradient,
           rss, residuals, nobs, iter, iconv plus component edf
    quiet  If F (default, unless keep:T), no summary results are printed.
           quiet:T is illegal without keep:T
```

Keywords: nonlinear fitting, regression

```
                              Introduction
nlreg() is a macro for carrying out nonlinear least squares regression.
It uses macro levmar() to do the actual minimization.  The function
fitted and/or residuals from the function are specified by macros in the
argument list.  Derivatives may be computed by a macro.  If no macro to
compute derivatives is provided, derivatives are computed numerically by
differencing.

                           Usage and arguments
 nlreg(b,x,y,f,param [,deriv:deriv,crit:vec,active:active,maxit:itmax,\
       minit:itmin,print:T, keep:T, quiet:T])
or
 nlreg(b,x,y, param ,resid:res [,deriv:deriv, crit:vec,\
       active:active,maxit:itmax,minit:itmin,print:T, keep:T, quiet:T])
```

```
  b      REAL vector of starting values for the iterative fitting
  x      REAL matrix; when f is an argument, nrows(x) = nrows(y) is
         required; when 'resid:res' is an argument, nrows(x) !=
         nrows(y) is allowed; if res() does not use x, x should be 0
  y      REAL vector
  f      macro; f(b,x,param) computes a vector with length nrows(y) of
         fitted values for y (not allowed with 'resid:res')
  param  a vector or structure of additional parameters for func or NULL
  res    macro; res(b,x,y [,param]) computes a vector of residuals from
         a function defined or directly referenced in res().  f must not
         be an argument when resid:res is an argument.  Conversely,
         when resid:res is not an argument, f is a required  argument.
         res() must return a vector of length nrows(y) and need not use
         x.
  deriv  optional macro: deriv(b,x,y,param,j) computes derivative of
         f(x,b,param) or -res(b,x,y,param) with respect to b[j],
         returning a vector the same length as y
  active LOGICAL vector the same length as b.  active[i] = F means b[i]
         is kept constant and does not participate in iteration
  vec    vector(numsig, nsigsq, delta), 3 criteria for convergence
         numsig = number of digits of accuracy in coefficients [8]
         nsiqsq = number of digits of accuracy in residual SS [5]
         delta  = norm of gradient threshhold [-1]
  itmin  the minimum number >= 0 of iterations performed
  itmax  the maximum number >= itmin of iterations allowed
  print  If T, partial results printed at each iteration
  keep   If T, nlreg() returns the structure returned by levmar() with
         components, coefs, hessian, jacobian, gradient, rss, residuals,
         nobs, iter, iconv plus component edf (error degrees of freedom)
  quiet  If F (default, unless keep:T), no summary results are
         printed.  quiet:T is illegal without keep:T

                              Value returned
With 'keep:T', nlreg() returns structure(coefs:b_hat,hessian:hes,
jacobian:jac,gradient:g,rss:Rssmin,residuals:resids,nobs:n,iter:niter,
iconv:convflag,edf:errordf).  The components are the same as returned by
levmar() with the addition of 'edf'.

The component values are as follows:

  b_hat     REAL vector which minimizes Rss
  hes       jac' %*% jac, an approximation to the Hessian matrix H,
            where H[i,j] = 2nd order partial derivative of Rss/2
            with respect to b_hat[i] and. b_hat[j]
  jac       the nrows(y) by nrows(b) Jacobian matrix; -jac[,j] = vector
            of partial derivatives of the elements of the residual
            vector with respect to b_hat[j].
  g         the nrows(b) REAL gradient vector with g[j] = partial
            derivative of Rss/2 with respect to b_hat[j].  g should be
            close to a vector of zeros at convergence
  Rssmin    the minimized value of Rss
  resids    vector of residuals of length nrows(y)
  n         positive integer = nrows(y) = nrows(resids) = nrows(jac)
```

```
  niter     positive integer = the number of iterations
  conflag   Convergence status flag; 0 = not converged, 1 = met relative
            change in b_hat criterion, 2 = met relative change in Rss
            criterion, 3 = met norm of gradient vector criterion, 4 =
            failed to reduce Rss on a step of the iteration.
  edf       Error degrees of freedom = nrows(y) - length(b_hat).

When any parameters are inactive as specified by keyword 'active' (see
below), jac is nrows(y) by p, hes is p by p, g has length p and edf =
nrows(y) - p, where p = number of active parameters.

                        Keyword phrase arguments
There are several of keywords that can be used to control the iteration
and hold parameters at fixed values.

  Keyword Phrase   Value and Explanation
  crit:crvec       vector(numsig, nsigsq, delta), 3 criteria for
                   convergence (default = vector(8,5,-1).  See below.
  active:act       LOGICAL vector the same length as b.  b[j]
                   "participates" in the iteration only if act[j] is
                   True (default = rep(T,length(b))).  When act[j] is
                   False, b_hat[j] remains at the starting value
  maxit:itmax      Non-negative integer specifying the maximum number of
                   iterations (default = 30).  When itmax = 0, no iterations
                   are done and the quantities returned are computed at
                   the starting value b
  minit:itmin      Non-negative integer < itmax specifies the mininum
                   number of iterations
  print:T          When T, partial results are printed on each iteration

Iteration stops when (i) itmax is exceeded, (ii) any of the three
convergence criteria are satisfied, or (iii) when there has been no
reduction in Rss on an iteration after 10 halvings of the initial step
size.

                           Use of argument param
One use for param might be to provide a choice between several functions
to be fit.  For example, the code for macro f might be the following:

  @b <- $1; @x <- $2; @p <- $3
  @val <- @b[1]
  for(@i,1,@p){
      @P <- @b[3*@i+1]
      @val <- @val + @b[3*@i-1]*cos(@x/@P,cycles:T) +\
        @b[3*@i]*sin(@x/@P,cycles:T)
  }
  @val # return

Here is the equivalent code for a macro res():

  @b <- $1; @x <- $2; @y <- $3; @p <- $4
  @res <- @y - @b[1]
  for(@i,1,@p){
```

```
    @P <- @b[3*@i+1]
    @res <- @res - @b[3*@i-1]*cos(@x/@P,cycles:T) -\
      @b[3*@i]*sin(@x/@P,cycles:T)
  }
  @res # return
```

Either might be used in fitting a curve with p cosine components with unknown periods, amplitudes or phases, with the number of terms specified by params.

Convergence criteria

There are 3 possible convergence criteria, at least one of which must be enabled. nlreg() terminates iteration when at least itmin iterations have been completed and any convergence criterion is satisfied or when itmax iterations have been completed.

The criteria are specified by optional argument crit:vec, where vec is vector(numsig [, nsigsq [, delta]]) with length <= 3. The default values for numsig, nsiqsq and delta are 8, 5 and -1, respectively.

A negative value for a criterion means it is not active.

numsig — Desired number of significant digits in every active coefficient. Specifically, the criterion is satisfied if, for every active coefficient b[j], the change d_j satisfies abs(d_j) < 10^-numsig*max(.5,abs(b_j)) where b_j is the updated value of b[j]. Component 'iconv' of the return value is 1 when satisfied.

nsigsq — Desired number of significant digits in Rss = the residual sum of squares. Specifically, the criterion is satisfied when abs(Rss_new - Rss_old) < 10^-nsigsq*max(.5,Rss_new). Component 'iconv' of the return value is 2 when satisfied.

delta — Desired maximum norm ||g|| of the gradient vector g. Specifically, the criterion is satisfied when sqrt(sum(g^2)) <= delta. Component 'iconv' of the return value is 3 when satisfied.

For numsig and nsigsq, the returned coefficients are the values updated on that iteration. For delta, the returned coefficients are the values found on the previous iteration.

Criteria are checked in the order delta, numsig and nsigsq.

Test macros and data

Included in this file are the following four macros used in debugging and testing nlreg():

```
 linear()    Linear function f <- x %*% b
 asymptot()  f = b[1] + b[2]*(b[3])^x & derivatives
 DandSFunc() f = b[1] + (.49 - b[1])*exp(-b[2]*(x - 8)) & derivatives
 testfun()   f = b[1]+b[2]*x+b[3]*x^2+b[4]*sin(b[5]*x)+b[6]*exp(-b[7]*x)
```

```
In addition, the following data sets are included in this file:
 DandS_T10.2  Data from Draper and Smith to be used with DandSFunc()
 SandCT19.8.1 Data from Snedecor and Cochran to be used with
              asymptot()
Example:
Fit the function b1 + b2*b3^x to data from Snedecor and Cochran with
starting values b1 = b2 = 40 and b3 = 1.

The data are also available in data set SandCT19.8.1 and the macro is
available as macro asymptot(), both in this file.

  Cmd> x <- vector(0, 1, 2, 3, 4, 5)

  Cmd> y <- vector(57.5, 45.7, 38.7, 35.3, 33.1, 32.2)

  Cmd> f <- macro("@b <- $1; @b[1]+@b[2]*@b[3]^($2)", dollars:T)

  Cmd> startVal <- vector(40,40,1) # starting values for iteration

  Cmd> nlreg(startVal,x,y,f)
             Coef      StdErr           t     P Value
  B 1      30.724     0.23099      133.01  9.3704e-07
  B 2      26.821      0.2577      104.08  1.9555e-06
  B 3     0.55184    0.008448      65.322  7.9056e-06
  ------------------------------------------------
  N: 6, MSE: 0.032416, DF: 3
  Converged with relative change in all coefs < 1e-05 in 7 iterations

  Cmd> resfunc <- macro("($3) - f($1,$2)") # computes residuals

  Cmd> nlreg(startVal,x,y,resid:resfunc) # identical
             Coef      StdErr           t     P Value
  B 1      30.724     0.23099      133.01  9.3704e-07
  B 2      26.821      0.2577      104.08  1.9555e-06
  B 3     0.55184    0.008448      65.322  7.9056e-06
  ------------------------------------------------
  N: 6, MSE: 0.032416, DF: 3
  Converged with relative change in all coefs < 1e-05 in 7 iterations

  Cmd> stuff <- nlreg(startVal,x,y,f,keep:T)

  Cmd> compnames(stuff)
   (1) "coefs"
   (2) "hessian"
   (3) "jacobian"
   (4) "gradient"
   (5) "rss"
   (6) "residuals"
   (7) "nobs"
   (8) "iter"
   (9) "iconv"
  (10) "edf"
```

```
  Cmd> stuff[vector(1,2,5,8,9,10)]
  component: coefs
  (1)       30.724       26.821      0.55184
  component: hessian
  (1,1)            6       2.1683       111.39
  (2,1)       2.1683       1.4367       30.242
  (3,1)       111.39       30.242       2675.8
  component: rss
  (1)     0.097248
  component: iter
  (1)            7
  component: iconv
  (1)            1
  component: edf
  (1)            3

  Cmd> # compute approximate standard errors

  Cmd> sqrt((stuff$rss/stuff$edf)*diag(solve(stuff$hessian)))
  (1)      0.23099       0.2577     0.008448

                             Cross reference
See also levmar().
```

8.6 predlimits()

Usage:

```
   predlimits(x, confLevel), x REAL scalar, vector or matrix with no
     MISSING elements, 0 < confLevel < 1 scalar
```

Keywords: regression, prediction limits

```
                                    Usage
You can use macro predlimits() to compute prediction limits for y =
E(y|x) + epsilon or y = E(y | x1, x2 ...) + epsilon after running
regress("y=x") or regress("y=x1+x2+..+xk").  These are limits on the a
future value of y for specified values of the predictor variable or
variables.

predlimits(x, confLevel), where x is a REAL vector with length(x) =
number of predictors (1 for simple linear regression), returns
vector(lower,upper), where lower and upper are prediction limit with
confidence level confLevel.  Argument confLevel must be a REAL scalar
between 0.5 and 1.

When confLevel < .5, a warning message is printed and 1 - confLevel is
used.

                         Example of single prediction
Example:
```

```
  After regress("y=x1 + x2 + x3"), predlimits(vector(2,3,4), .95)
  returns vector(lower,upper) where lower and upper are the limits when
  x1=2, x2=3 and x3=4

                    Limits for several predictions
You can use predlimits() to get limits for several values at once,
returning hconcat(lower,upper), where lower and upper are vectors of
limits for the various values.

For simple linear regression, x should be a vector containing the
values for which you want prediction limits.

When there are k predictors, x should be a matrix with k columns, and
lower and upper will have length(nrows(x)).

                    Example of several predictions
Example:
  After regress("y = x1 + x2"), predlimits(vconcat(vector(1,2)',
  vector(2,1.5)',vector(3,3.2)'), .95) returns hconcat(lower,upper)
  where, for example, lower[2] and upper [2] are the limits when x1 = 2
  and x2 = 1.5.

                           Cross references
See also estimlimits(), regpred(), glmpred().
```

8.7 regcoefs()

Usage:

```
   regcoefs(Model [,pvals:T] [,byvar:F]) or regcoefs([pvals:T] [,byvar:F]),
     where Model is a CHARACTER scalar
```

Keywords: glm, anova, regression, confidence limits, standard error

```
                                  Usage
regcoefs(Model) returns a matrix with appropriately labeled rows and
columns of the regression coefficients, their standard errors and
t-statistics from a least squares fit to the regression model specified
by Model.  There can be no factors in Model.  If Model is omitted, the
most recent GLM model is used.

regcoefs(Model,pvals:T) or regcoefs(pvals:T) also computes two-tail P
values corresponding to the t-statistics on the basis of Student's
t-distribution with degrees of freedom from the last element of side
effect variable DF.

Because of the row and column labels, after any GLM command with a model
withut factors, typing regcoefs([pvals:T]) produces a table similar to
that produced by regress().  After non-linear fits such as logistic() or
poisson(), the P-values will not necessarily be appropriate.
```

```
                          Multivariate response
If the response variable is multivariate, the result is a structure,
each of whose components is a labeled matrix of coefficients, standard
errors and t-statistics.  regcoefs(Model,byvar:F) or regcoefs(byvar:F)
returns a single labeled matrix, with separate columns for the
coefficients, standard errors, ... for each variable.

                            Cross references
See also topics 'glm', regress(), secoefs().
```

8.8 regresshelp()

Usage:

```
regresshelp(topic1 [, topic2 ...] [,usage:T])
regresshelp(index:T)
```

Keywords: general

```
regresshelp(topicname) prints help on a topic related to file
regress.mac.  Usually topicname is the name of a macro in the file.

When quoted, topicname may contain "wildcard" characters "*" and "?".
You can also use help keyword 'key'.  See help() for details.

regresshelp(topicname1, topicname2, ...) prints help on more than one
topic.

regresshelp(topicname1 [, topicname2 ...], usage:T) prints just a brief
summary of usage for the each topic.
```

8.9 regs()

Usage:

```
regs(x,y [,T] [GLM keywords]), x and y REAL matrices with the same
  number of rows; T means no intercept
```

Keywords: glm, regression

```
                                  Usage
regs(x,y) computes the regression of y on the columns of x.  For
example, when data is a n by 7 matrix, say, you can compute the
regression of the last column on the first 6 by regs(data[,-7],
data[,7]).

regs(x,y,T) does the same, except the model fit has no constant term
(intercept).

You can use GLM keyword phrases such as 'pvals:T', 'silent:T',
'wts:weights', and 'marginal:T' as additional arguments.
```

```
Macro regs() creates temporary variables @X1, @X2, ... and @Y and then
invokes regress() or, if y has more than 1 column, manova().

When y is univariate, immediately follow regs() by anova() to see the
ANOVA table.  When y is multivariate, follow regs() by regcoefs() to see
coefficients and standard errors.

Because the model for regress() or manova() uses temporary variables,
STRMODEL cannot be used as a model for a subsequent regress(), anova(),
or manova() command.  However, these variables can be retrieved, by
modelvars().   For example, when x has 3 columns,

  Cmd> makecols(modelvars(x:T),X1,X2,X2)

creates variables X1, X2 and X3.  Of course, if x still exists,
makecols(x,X1,X2,X3) does the same.

                            Cross references
See also regress(), anova(), modelvars(), regcoefs(), and 'glm_keys'.
```

8.10 removevar()

Usage:

```
removevar(var1 [,var2 ...] [,silent:T]), var1, var2, ... the names or
  numbers of independent variables in the current stepwise model
```

Keywords: stepwise regression, regression

```
                                  Usage
removevar(Var) removes independent variable from the current stepwise
regression model.  Var can be either a quoted name ("z3"), an unquoted
name (z3) or the number of a variable in the current stepwise model (an
'in' variable).  Thus if the full model is "y=x1+x2+x3+x4+x5",
removevar(x2), removevar("x2") and removevar(2) are equivalent.

It is an error if the variable is not an independent variable in the
full stepwise model or if it is not in the current model.

Invisible variable _STEPSTATUS is updated to reflect the changed model.
See topic '_STEPSTATUS'.

                              Printed output
The F-to-remove statistics with P-values are printed for all the
variables in the model, and F-to-enter statistics with P-values are
printed for any variables not in the model, including Var.

In addition, if there are any variables left in the model, removevar()
prints an overall F statistic and its P-value, Mallow's Cp statistic,
adjusted R^2 and R^2.  The F-statistic tests the null hypothesis that
the coefficients of the "in" variables are 0.
```

```
                              Value returned
The value returned is the updated invisible variable _STEPSTATUS.  It
can be assigned (stuff <- removevar(x3)), but is not printed.

                          Removing several variables
removevar(Var1, Var2 ...) does the same except that more than one
variable is removed.  All variable must be in the current stepwise
model.  The model and other statistics are printed after each variable
is removed.  The value returned is _STEPSTATUS after all have been
removed.  An example when the full model is "y=x1+x2+x3+x4+x5"
might be removevar(x1,"x2",4).  This would remove x1, x2 and x4 in that
order.

                              Keyword silent
removevar(Var1 [,Var2 ...], silent:T) does the same, except that the
model and summary statistics are not printed.

                             Cross references
See also entervar(), stepsetup(), stepstatus()
```

8.11 resid()

Usage:

```
resid() or resid(Model)
```

Keywords: glm, residuals, anova, regression

```
                                    Usage
resid(), with no argument, computes a REAL matrix of various quantities
useful in the analysis of residuals.  It uses side effect variables
RESIDUALS, HII, etc. produced by the most recent GLM (generalized linear
or linear model) command such as regress(), anova(), or poisson().

It is an error if any of the needed side effect variables do not exist.

resid(Model) first executes manova(Model, silent:T) to compute the
required side effect variables before computing the residual-related
quantities.  Model should be a CHARACTER variable or string specifying a
linear ANOVA or MANOVA model.  Any factors in the model will be treated
as factors.  If you want them treated as variates, use resid(Model,T).

                            Description of output
Each row of the result corresponds to a case.  When the dependent
variable Y is univariate, there are 5 columns, as follows:
   Col. 1   Y = observed response
   Col. 2   Studentized residuals = RESIDUALS/SE(RESIDUALS)
   Col. 3   HII = leverage
   Col. 4   Cook's distance
   Col. 5   t-statistics = externally studentized residuals =
```

```
            RESIDUALS/SE*(RESIDUALS), where SE* for each case is
            a standard error based on the model fit excluding that case.

When Y is multivariate of dimension p, there are 4*p + 1 columns -- the
p values for Y, the p standardized residuals, HII, the p Cook's
distances, and the p externally studentized residuals.

If a case has missing values, most entries for that case will be MISSING
and there are no useful numbers.

                      After nonlinear GLM commands
After non-linear GLM commands such as poisson() and logistic(), the
results are based on the last stage of the iteratively reweighted least
squares algorithm used to fit the model.  Residuals are standardized by
the error mean square in the linear scale.  They should still be valid
for diagnosing departures from the model.

The output of resid() is modeled on what is printed by the resid()
command in program Multreg.

resid() is implemented as a pre-defined macro.

                            Cross references
See also topics 'glm', yhat(), resvsindex(), resvsrankits(),
resvsyhat().
```

8.12 resvsindex()

Usage:

```
resvsindex([varNo,] [usehii:T or F] [,standres:F]\
  [,graphics keyword phrases]), 1 <= varNo <= ncols(RESIDUALS)
```

Keywords: plotting, glm, residuals, anova, regression

```
                                  Usage
You use resvsindex() to plot standardized (default) or non-standardized
residuals vs case numbers.

resvsindex([graphics keyword phrases]) plots standardized residuals
against case number.

resvsindex(usehii:T [,graphics keyword phrases]) does the same using
leverages HII in standardizing.  This is the default after a GLM
command.

resvsindex(usehii:F [,graphics keyword phrases]) does the same without
using leverages HII.  This is the default after arima().

resvsindex(standres:F [,graphics keyword phrases]) does the same without
any standardization.
```

The residuals are from variable RESIDUALS or WTDRESIDUALS produced by the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson(), or from an ARIMA fit computed by macro arima().

If the most recent command was manova(), only column 1 of the residual matrix is plotted, but see the next usage for plotting other columns.

After manova()

resvsindex(varNo [, usehii:T or F] [, standres:F] [,graphics keyword phrases]), where varNo is an integer between 1 and ncols(RESIDUALS), plots residuals associated with variable varNo against case numbers. varNo > 1 is legal only when RESIDUALS was computed by manova().

Plotting symbols

The default plotting symbol is the same as for plot(), a drawn asterisk or star ("\6"). You can change it by including 'symbols:c' as an argument, where c is a CHARACTER or integer scalar or vector. c = 0 is special: it is equivalent to c = "###" and results in points being labeled with case number. See chplot(), subtopic 'symbols_used'.

Graphics keywords

You can use all the usual graphics keywords to modify the default plot characteristics. These include 'title', 'xlab', 'ylab', 'symbols' 'impulse' and 'lines'. See topics 'graphs', 'graph_keys', 'graph_border' and 'graph_ticks'.

When you have set option 'dumbplot' to False (see 'options'), the plot will be a low resolution plot unless 'dumb:F' is an argument.

What is plotted

Without standres:T, the quantities plotted are r[i]/sd[i] where r[i] is RESIDUALS[i] or WTDRESIDUALS[i] and sd[i] is the estimated standard deviation. WTDRESIDUALS[i] is used after regress(), anova(), or manova() with 'weights:wts' or after nonlinear GLM commands such as logistic() and poisson().

When usehii is True (the default after GLM commands), sd[ii] = sqrt(mse*(1-HII[i])), where mse is the residual mean square after regress(), anova() or manova(), the mean error deviance after non-linear GLM commands or the estimated innovation variance after arima().

When usehii is False (the default after arima()), sd[i] = sqrt(mse).

With standres:F, the quantities plotted are r[i].

The values on the X-axis are 1, 2, ..., nrows(RESIDUALS).

resvsindex() is implemented as macro.

Cross references

See also topics resvsrankits(), resvsyhat(), resid().

8.13 resvsrankits()

Usage:

```
resvsrankits([varNo,] [usehii:T or F] [,standres:F]\
  [,graphics keyword phrases]), 1 <= varNo <= ncols(RESIDUALS)
```

Keywords: plotting, glm, residuals, anova, regression

```
                                Usage
resvsrankits([graphics keyword phrases]) plots standardized residuals
against normal scores as computed by function rankits().

resvsrankits(usehii:T [,graphics keyword phrases]) does the same using
leverages HII in standardizing.  This is the default after a GLM
command.

resvsrankits(usehii:F [,graphics keyword phrases]) does the same without
using leverages HII.  This is the default after arima().

resvsrankits(standres:F [,graphics keyword phrases]) does the same
without any standardization.

The residuals are from variable RESIDUALS or WTDRESIDUALS produced by
the most recent GLM (generalized linear or linear model) command such as
regress(), anova(), or poisson(), or from an ARIMA fit computed by macro
arima().

If the most recent command was manova(), only column 1 of the residual
matrix is plotted, but see the next usage for plotting other columns.

                            After manova()
resvsrankits(varNo [, usehii:T or F] [, standres:F] [,graphics keyword
phrases]), where varNo is an integer between 1 and ncols(RESIDUALS),
plots residuals associated with variable varNo against case numbers.
varNo > 1 is legal only when RESIDUALS was computed by manova().

                          Plotting symbols
The default plotting symbol is the same as for plot(), a drawn asterisk
or star ("\6").  You can change it by including 'symbols:c' as an
argument, where c is a CHARACTER or integer scalar or vector.  c = 0 is
special: it is equivalent to c = "###" and results in points being
labeled with case number.  See chplot(), subtopic 'symbols_used'.

                          Graphics keywords
You can use all the usual graphics keywords to modify the default plot
characteristics.  These include 'title', 'xlab', 'ylab', 'symbols'
'impulse' and 'lines'.  See topics 'graphs', 'graph_keys',
'graph_border' and 'graph_ticks'.

When you have set option 'dumbplot' to False (see 'options'), the plot
```

```
will be a low resolution plot unless 'dumb:F' is an argument.

                              What is plotted
Without standres:T, the quantities plotted are r[i]/sd[i] where r[i] is
RESIDUALS[i] or WTDRESIDUALS[i] and sd[i] is the estimated standard
deviation.  WTDRESIDUALS[i] is used after regress(), anova(), or
manova() with 'weights:wts' or after nonlinear GLM commands such as
logistic() and poisson().

When usehii is True (the default after GLM commands), sd[ii] =
sqrt(mse*(1-HII[i])), where mse is the residual mean square after
regress(), anova() or manova(), the mean error deviance after non-linear
GLM commands or the estimated innovation variance after arima().

When usehii is False (the default after arima()), sd[i] = sqrt(mse).

With standres:F, the quantities plotted are r[i].

The values on the X-axis are normal scores computed by rankits(r).  See
rankits() for more information.


resvsrankits() is implemented as macro.

                             Cross references
See also topics resvsindex(), resvsyhat(), resid().
```

8.14 resvsyhat()

Usage:

```
   resvsyhat([varNo,] [usehii:T or F] [,standres:F]\
     [,graphics keyword phrases]), 1 <= varNo <= ncols(RESIDUALS)
```

Keywords: plotting, glm, residuals, anova, regression

```
                                    Usage
resvsyhat([graphics keyword phrases]) plots standardized residuals
against fitted or predicted values.

resvsyhat(usehii:F [,graphics keyword phrases]) does the same without
using leverages HII in standardizing.  The default is to use HII.

resvsyhat(standres:F [,graphics keyword phrases]) does the same without
any standardization, that is, the residuals y - y_hat are plotted.

The residuals are from variable RESIDUALS or WTDRESIDUALS produced by
the most recent GLM (generalized linear or linear model) command such as
regress(), anova(), or poisson().

If the most recent command was manova(), only column 1 of the residual
matrix is plotted, but see below for plotting other columns.
```

Unlike resvsrankits() and resvsindex(), resvsyhat() cannot be used to make a residual plot after arima() was used to estimate an ARIMA time series model.

After manova()

resvsyhat(varNo [, usehii:T or F] [, standres:F] [,graphics keyword phrases]), where varNo is an integer between 1 and ncols(RESIDUALS), plots residuals associated with variable varNo against case numbers. varNo > 1 is legal only when RESIDUALS was computed by manova().

Plotting symbols

The default plotting symbol is the same as for plot(), a drawn asterisk or star ("\6"). You can change it by including 'symbols:c' as an argument, where c is a CHARACTER or integer scalar or vector. c = 0 is special: it is equivalent to c = "###" and results in points being labeled with case number. See chplot(), subtopic 'symbols_used'.

Graphics keywords

You can use all the usual graphics keywords to modify the default plot characteristics. These include 'title', 'xlab', 'ylab', 'symbols' 'impulse' and 'lines'. See topics 'graphs', 'graph_keys', 'graph_border' and 'graph_ticks'.

When you have set option 'dumbplot' to False (see 'options'), the plot will be a low resolution plot unless 'dumb:F' is an argument.

What is plotted

Without standres:T, the quantities plotted are r[i]/sd[i] where r[i] is RESIDUALS[i] or WTDRESIDUALS[i] and sd[i] is the estimated standard deviation. WTDRESIDUALS[i] is used after regress(), anova(), or manova() with 'weights:wts' or after nonlinear GLM commands such as logistic() and poisson().

When usehii is True (the default after GLM commands), sd[ii] = sqrt(mse*(1-HII[i])), where mse is the residual mean square after regress(), anova() or manova() or the mean error deviance after non-linear GLM commands.

When usehii is False, sd[i] = sqrt(mse).

With standres:F, the quantities plotted are r[i].

The values on the X-axis are the estimated means of the response variable. After a nonlinear GLM command, they are in the original scale, not the transformed scale.

resvsyhat() is implemented as macro.

Cross references

See also topics resvsindex(), resvsrankits(), resid(), yhat().

8.15 steplook()

Usage:

```
steplook(item1, item2, ...), item1, item2, ... unquoted words selected
  from 'model', 'sscp', 'in', 'F', 'dfe', 'fullmse' and 'history'
```

Keywords: stepwise regression, regression

```
                                Usage
steplook(item1, item2, ... ) returns the values of components of
hidden variable _STEPSTATUS (see topic '_STEPSTATUS').  The arguments
must match component names of _STEPSTATUS, that is they must be one
or more of 'model', 'sscp', 'in', 'F', 'dfe', 'fullmse' and 'history'.

If only one item is requested, the value is a scalar, vector or
matrix, depending on the item.  Otherwise, the value is a structure.

Examples:
  Cmd> steplook(model, history, in)
  returns structure(model:_STEPSTATUS$model,history:_STEPSTATUS$history,
    in:_STEPSTATUS$in)

  Cmd> steplook(in)
  returns _STEPSTATUS$in

  Cmd> steplook(F)[1,steplook(in)]
  returns F-to-remove for variables in current stepwise model

  Cmd> steplook(F)[2,!steplook(in)]
  returns P-values for F-to-enter for variables not in current stepwise
  model

  Cmd> regress(steplook(model),pvals:T)

  estimates the coefficients for the current stepwise model.

                           Cross references
See also topics stepsetup(), entervar(), removevar(), stepstatus()
```

8.16 stepsetup()

Usage:

```
stepsetup([Model] [,silent:T] [,allin:T or in:logVec]), Model of form
  "y = x1+x2+...+xk" or "y = x1+x2+...+xk - 1", logVec a LOGICAL
  vector of length k
```

Keywords: stepwise regression, regression

```
                                Usage
stepsetup(Model), where Model is a CHARACTER scalar specifying a
regression model, initializes a stepwise regression process.
```

```
It creates invisible variable _STEPSTATUS and prints the F-to-enter
statistics and P-values for all the variables in the model.

It returns _STEPSTATUS as value.  This can be assigned (stuff <-
stepsetup("y=x1+x2+x3+x4")) but is not printed.  See topic
'_STEPSTATUS'.

stepsetup(), with no model specified, does the same, except it uses
STRMODEL, the model for the most recent GLM command, as model.

                      Keywords silent, in and allin
stepsetup([Model,] silent:T) does the same except nothing is printed.

stepsetup([Model,] allin:T [,silent:T]) does the same, except all the
variables are entered in the model immediately so that backward
stepwise regression can be done.

stepsetup([Model,] in:In [,silent:T]) does the same, except the
variables specified by In are entered in the model immediately.  In
must be a LOGICAL vector the same length as the number of independent
variables in Model.  Variable j is considered in the model if In[j] is
True.

                            Printed output
When variables are initially entered in the model, stepsetup() prints an
overall F statistic and its P-value, Mallow's Cp statistic, adjusted
R^2 and R^2.  The F-statistic tests the null hypothesis that the
coefficients of the "in" variables are 0.

Examples:
  Cmd> stepsetup("y=x1+x2+x3+x4+x5")
starts stepwise regression process with no variables in the model.

  Cmd> stepsetup("y=x1+x2+x3+x4+x5+x6-1",in:vector(F,F,F,T,T,T))
starts stepwise regression process with variables x4, x5 and x6 in the
model.  Because of "-1" in the model, no intercept is included.

                            Cross references
See also stepstatus(), entervar(), removevar(), steplook().
```

8.17 stepstatus()

Usage:

```
stepstatus([silent:T])
```

Keywords: stepwise regression, regression

```
                                  Usage
stepstatus() prints the current stepwise regression model, F-to-remove
```

```
statistics with P-values for all the variables currently in the model,
and F-to-enter statistics with P-values for any variables not in the
model.

In addition, if there are any variables in the model, stepstatus()
prints an overall F statistic and its P-value, Mallow's Cp statistic,
adjusted R^2 and R^2.  The F-statistic tests the null hypothesis that
the coefficients of the "in" variables are 0.

It returns _STEPSTATUS as value.  This can be assigned (stuff <-
stepstatus()) but is not printed.  See topic '_STEPSTATUS'.

                              Keyword silent
stepstatus(silent:T) prints nothing but returns _STEPSTATUS as value.
```

8.18 testbeta()

Usage:

```
testbeta(Term, hypValue [,df:T] [,pval:T]), Term a quoted or unquoted
  variable name or an integer term number, hypValue a non-MISSING real
  scalar
```

Keywords: regression, hypothesis test

```
                                    Usage
You can use testbeta() to compute a t-statistic to test a null
hypothesis of the form H0: betaj = hypValue, where betaj is a regression
coefficient.  There must be an active regression model, that is you need
to have executed a command of the form regress("y = x" or regress("y=x1
+ x2 + ... xk").

testbeta(Term, betaj0), where Term is the quoted or unquoted name of a
predictor in the regression (for example, "x3" or x3) and betaj0 is a
REAL scalar whose value is the hypothesized value of the regression
coefficient of the predictor.  Term can also be the number of the term
associated with the variable in an ANOVA.

The value is the test statistic Tstat = (betajHat-betaj0)/SE[betajHat].

                           Keywords pvalue and df
testbeta(Term, betaj0, pvalue:T) does the same except the value returned
is structure(tstat:Tstat, pvalue:Pvalue), where Pvalue is the two-tail
P-Value associated with Tstat.

testbeta(Term, betaj0, df:Df [, pvalue:T]), returns
structure(tstat:Tstat, df:Df [,pvalue:Pvalue]), where Df = error
degrees of freedom.

Example:
  Cmd> regress("y = x", silent:T) # output suppressed
```

```
  Cmd> testbeta(x, 1, pvalue:T, df:T) # tests H0: beta = 1

                            Cross references
See also secoefs(), twotailt().
```

8.19 testestim()

Usage:

```
   testestim(x, hypValue), x REAL scalar or vector with no MISSING
     elements, hypValue a non MISSING REAL scalar
```

Keywords: regression, hypothesis test

```
                                  Usage
You can use macro testestim() to test a null hypothesis of the form
H0: E(y|x) = hypValue or E(y|x1,x2,...,xk) = hypValue, where hypValue
is a hypothesized value for the expectation of y for given x values.
You must previously have run regress("y=x") or regress("y=x1+...+xk").

testestim(x, hypValue), where hypValue is the hypothesized value, x is
a scalar (simple linear regression) or a vector of length k (multiple
regression) returns a t-statistic of for testing H0.  It is an error if
there is not an active regression model or if length(x) != k.

                          Keywords df and pvalue
testestim(x, hypValue, df:T) does the same except the result is
structure(tstat:tvalue, df:errorDF), where errorDF is the error degrees
of freedom needed to compute a P-value or find a critical value.

                            Keyword pvalue
testestim(x, hypValue [,df:T] , pvalue:T) does the same except the
result is structure(tstat:tvalue [,df:errorDF], pvalue:Pvalue), where
Pvalue is the two-tail P-value associated with the test statistic.

Example:
After regress("y=x1 + x2 + x3"),
  testestim(vector(2,3,4), 17) returns the t-statistic for testing H0:
     E(y|x) = 17
  testestim(vector(2,3,4), 17, pval:T, df:T) returns
     structure(tstat:t_statistic, df:ErrorDF, pval:p_value)

                            Cross references
See also estimlimits(), predlimits(), regpred().
```

8.20 yhat()

Usage:

```
   yhat() or yhat(Model [,T])
```

Keywords: glm, regression, anova

Usage

yhat(), with no argument, computes a REAL matrix of various quantities useful in making predictions from a regression or analysis of variance model.

yhat() uses side effect variables RESIDUALS, HII, etc. produced by the most recent GLM (generalized linear or linear model) command such as regress() or anova(). If weights were supplied, it also uses the result of modelinfo(weights:T).

NOTE: If the most recent GLM was not linear, that is, not regress(), anova(), manova() or their weighted variants, only the first two columns computed by yhat() are meaningful.

It is an error if any of the needed side effect variables do not exist.

Specified model

yhat(Model) first executes manova(Model, silent:T) to compute the side effect variables and then computes its usual output. Model should be a CHARACTER variable or string specifying a linear model. Any factors in the model will be treated as factors.

yhat(Model,T) does the same, except any factors in the model are treated as variates.

Value returned

Each row of the result corresponds to a case. When the dependent variable Y is univariate (has one column), the result has the following 5 columns:

Col. 1	Y = observed response
Col. 2	Yhat = predicted or fitted value computed using all data
Col. 3	Predictive residuals = Y - (Yhat computed excluding the case)
Col. 4	SE[Yhat] = estimated standard error of Yhat as estimate of E[Y | x]
Col. 5	SE[pred] = estimated s.e. of prediction error

When the all the independent variables in the model "y=x1+x2+x3+...+xk" are variates and not factors, columns 2, 4, and 5 of the output correspond to components 'estimate', 'SEest', and 'SEpred" in the output of regpred(hconcat(x1,x2,...)) following regress("y=x1+x2+...+xk"). See regpred().

Multivariate model

When Y is multivariate, with p columns, there are 5*p columns in groups of p -- the p columns of Y, the p columns of YHat, and so on.

If a case has missing values, most entries will be MISSING and there are no useful numbers.

```
Example:
  Cmd> yhat("y=x1+x2+x3+x4")

The output of yhat() is modelled on output printed by the yhat() command
in program Multreg.

yhat() is implemented as a pre-defined macro.

                             Cross references
See also topics regpred(), 'glm', resid().
```

Chapter 9

Time Series Macros Help File

This Chapter contains help for the set of time series macros that are distributed with MacAnova in the file Tser.mac.txt. The material here is a reformatting of file Tser.hlp.txt.

9.1 arspectrum()

Usage:

```
arspectrum(Y,P [,nfreq:nfreq] [,nospec:T]), Y a REAL vector, integer P >
  0, integer nfreq > 0.
```

Keywords: spectrum analysis, frequency domain, autoregression, arima models

```
                                  Usage
arspectrum(Y,P) estimates the spectrum of Y considered as a discrete
parameter AR(P) (order P autoregressive) time series.  Y must be a REAL
vector with no MISSING elements and P > 0 an integer.

The value returned is a structure(phi:Phi,var:Var, spectrum:Sy)
   Phi    REAL vector of length P containing estimated AR coefficients
   V      REAL scalar containing the estimated variance of the residuals
          (innovations)
   Sy     REAL vector of length Nfreq (see below) containing the
          estimated spectrum computed at frequencies 0, 1/Nfreq,
          2/Nfreq, ..., (Nfreq-1)/Nfreq cycles per Delta_t, the
          interval between observations.

The default value for Nfreq is determined as follows:
  1.  If variable S is defined and is an integer > 2, Nfreq = S.  It is
  an error if S has a prime factor > 29.

  2.  Otherwise, Nfreq is the smallest integer >= 2*nrows(y) which has
  no prime factor > 29, that is Nfreq = goodfactors(2*nrows(y))

arspectrum(Y,P,nfreq:Nfreq) or arspectrum(Y,P,Nfreq), where Nfreq > 0 is
an integer, does the same, computing the spectrum at Nfreq frequencies.
It is an error if Nfreq has a prime factor > 29.
```

```
                              Keyword nospec
arspectrum(Y,P,nospec:T) computes phi and var but not spectrum,
returning structure(phi,var).

                                  Method
Estimated AR coefficients Phi are computed from the first P sample
autocorrelations by solving the Yule-Walker equations.  See
yulewalker().  The variance V = gammahat(0)*prod(1 - pacfˆ2),
where gammahat(0) = sum((y - ybar)ˆ2)/n and pacf is the vector of
P partial autocorrelations associated with Phi.

                              Cross reference
See also burg(), yulewalker().
```

9.2 autocor()

Usage:

```
autocor(y [, nlags [, nfreq]] [,full:T] [center:T] [,degree:d]), y a REAL
  vector or matrix, nlags > 0 and nfreq > 0 integers, d an integer
  scalar or vector
```

Keywords: time domain, autocorrelation

```
                                    Usage
r_y <- autocor(y, nlags) computes sample autocorrelations acf(h,y) (y a
REAL vector) or acf(h,y[,j]) (y a REAL matrix) for lags h = 1, 2, ...,
nlags.

The columns of y, which may have no MISSING elements, are interpreted as
time series defined at equally spaced time points.

For a vector y of length N, the sample autocorrelations are defined as
   acf(h,y) = acvf(h,y)/acvf(0,y),
where
   acvf(h,y) = sum((y[i] - ybar)*(y[i+h] - ybar),i=1,...,N-h)/N
is the sample autocovariance.  ybar = sum(y)/N is the sample mean of y.

When ny = ncols(y) = 1, r_y is a vector of length L = nlags with r_y[k]
= acf(k,y).  When ny > 1, r_y is a L by ny matrix with r_y[k,j] =
acf(k,y[,j]).

autocor(y) is the same as autocor(y, nrows(y) - 1), computing all
non-degenerate sample autocorrelations.

                              Keyword degree
You can specify that columns of y be detrended by subtracting a
polynomial in time using keyword phrase 'degree:d', where d is an
integer scalar or vector of length ny = ncols(y).  A scalar d is
interpreted as rep(d,ny).

r_y <- autocor(y [,nlags] degree:d) computes autocorrelations of the
```

```
residuals from polynomial trends fit by least squares to each column of
y.  The values of the fitted polynomial replace ybar in the definition
of acvf().

d[j] is the degree of the polynomial fit to column j.  When d[j] = 0,
only the sample mean is subtracted (default).  When d[j] < 0, nothing is
subtracted.

                                Keywords
You can modify the behavior of autocor() using keyword phrases 'full:T'
(compute acf for negative lags, with acf(-h,y) = acf(h,y) in row
2*nlags+2-h), and 'center:T' (same, but with acf(0,y) in row nlags+1).
Note that the output includes acf(0,y) which is not included in the
default output.  See crosscor() for details.

                               Computation
Autocovariances acvf(h,y) are computed using crosscov() which makes use
of discrete Fourier transforms (DFTs) of length goodfactors(N+nlags),
the smallest integer >= N + nlags which has no prime factors > 29.  See
topic 'fourier'.

autocor(y, nlags, M) where M is an integer, does the same, using DFTs of
length M.  It is an error if M < N + nlags or if M has a prime factor >
29.  See topic 'fourier'.

                         Plotting autocorrelations
You can use macro tsplot() to plot the computed autocorrelations.  Any
of the following might be appropriate to plot the first 50
autocorrelations of the columns of y.

  Cmd> tsplot(autocor(y, 50), 1) # line plot
  Cmd> tsplot(autocor(y, 50), 1, impulse:T) # impulse plot
  Cmd> tsplot(autocor(y, 50), 1, impulse:T, lines:T) # line and impulse
  Cmd> tsplot(autocor(y, 50), 1, char:charVec)#plot lines & symbols
  Cmd> tsplot(autocor(y,50,center:T),-50, impulse:T)

On each of these it would usually be appropriate to include keyword
phrases ymin:-1 and ymax:1 since autocorrelations are between -1 and 1.

                              Cross reference
See also autocov(), crosscov(), crosscor(), tsplot().
```

9.3 autocov()

Usage:

```
autocov(y [, nlags [, nfreq]] [,full:T] [center:T] [,degree:d]), y a REAL
  vector or matrix, nlags > 0 and nfreq > 0 integers, d an integer
  scalar or vector.
```

Keywords: time domain, autocovariance

```
                                Usage
```

```
c_y <- autocov(y, nlags) computes sample autocovariances acvf(h,y) (y a
REAL vector) or acvf(h,y[,j]) (y a REAL matrix) for lags h = 0, 1, ...,
nlags.

The columns of y, which may have no MISSING elements, are interpreted as
time series defined at equally spaced time points.

For a vector y of length N, the sample autocovariances are defined as
    acvf(h, y) = sum((y[i] - ybar)*(y[i+h] - ybar),i=1,...,N-h)/N
where ybar = sum(y)/N is the sample mean of y.

When ny = ncols(y) = 1, c_y is a vector of length L = nlags + 1 with
c_y[k] = acvf(k-1,y).  When ny > 1, c_y is a L by ny matrix with
c_y[k,j] = acvf(k-1,y[,j]).

autocov(y) is the same as autocov(y, nrows(y) - 1), computing all
non-degenerate sample autocovariances.

                              Keyword degree
You can specify that columns of y be detrended by subtracting a
polynomial in time using keyword phrase 'degree:d', where d is an
integer scalar or vector of length ny = ncols(y).  A scalar d is
interpreted as rep(d,ny).

c_y <- autocov(y [,nlags] degree:d) computes autocovariances of the
residuals from polynomial trends fit by least squares to each column of
y.  The values of the fitted polynomial replace ybar in the definition
of acvf().

d[j] is the degree of the polynomial fit to column j.  When d[j] = 0,
only the sample mean is subtracted (default).  When d[j] < 0, nothing is
subtracted.

                                 Keywords
You can modify the behavior of autocov() using keyword phrases 'full:T'
(compute acvf for negative lags, with acvf(-h,y) in row 2*nlags+2-h), and
'center:T' (same, but with acvf(0,y) in row nlags+1).  See crosscov() for
details.

                               Computation
Autocovariances are computed using crosscov() which makes use of
discrete Fourier transforms (DFTs) of length goodfactors(N+nlags), the
smallest integer >= N + nlags which has no prime factors > 29.  See
topic 'fourier'.

autocov(y, nlags, M) where M is an integer, does the same, using DFTs of
length M.  It is an error if M < N + nlags or if M has a prime factor >
29.  See topic 'fourier'.

                          Plotting autocovariances
You can use macro tsplot() to plot the computed autocovariances.  Any of
the following might be appropriate to plot the first 50 autocorrelations
of the columns of y.
```

```
  Cmd> tsplot(autocov(y, 50), 0) # line plot
  Cmd> tsplot(autocov(y, 50), 0, impulse:T) # impulse plot
  Cmd> tsplot(autocov(y, 50), 0, impulse:T, lines:T) # impulse and lines
  Cmd> tsplot(autocov(y, 50), 0, char:charVec) # plot lines & symbols

                              Cross reference
See also autocor(), crosscov(), crosscor(), tsplot().
```

9.4 bandwidth

Keywords: spectrum analysis, frequency domain

```
                                 Introduction
Suppose a spectrum estimate is computed by circular convolution (see
MacAnova topic convolve()) of weights w[1], w[2], ..., w[nfreq] with a
periodogram computed from an untapered time series at nfreq equally
spaced frequences 0,1/nfreq, 2/nfreq, ..., (nfreq-1)/nfreq cycles per
delta_t where delta_t is the time interval between observations.

An important characteristic of the estimate is its bandwidth, the
effective frequency range over which appreciable smoothing occurs.  The
greater the bandwidth, the more stable (smaller variance) is the
resulting estimate, but the greater the potential for bias because of
the smoothing.  Conversely, the smaller the bandwidth, the smaller is
the bias, at the cost of increased variance of the estimate.

A common definition of the bandwidth associated with a periodogram
computed at nfreq frequencies and smoothed by w[i] is
          B = {sum(w[i])^2/sum(w[i]^2)}/(nfreq*delta_t).

This is in units of cycles per unit time.

When {w[i]} is a "box car" of length m, that is w[i] = 0 except for m
consecutive values of i for which w[i] is constant, then B =
m/(nfreq*delta_t).  In the common situation when nfreq ~= 2*N, where N
= nrows(y), B = .5*m/(N*delta_t).

                            Convolution weights
A common type of smoothing weights are obtained by convolving
(non-circularly) a box car with itself P times.  This is sometimes
called the P-th convolution power of the box car.

For P = 2, the a graph of the weights is a triangle; for higher P the
graph becomes bell shaped.  For large P the graph approximates a normal
curve.  MacAnova macros spectrum(), crsspectrum() and compfa() all use
the 4-th convolution power of a box car as the smoothing weights.

When the weights are computed as the P-th convolution power of a box
car of length m, sum(w^2)/sum(w)^2 and sum(w)^2/sum(w^2) are as follows.
    P             sum(w^2)/sum(w)^2                  sum(w)^2/sum(w^2)
```

```
    1     1/m                                                  m
    2     (2*mˆ3+m)/(3*mˆ4)                          1.5*m-.75/m+O(mˆ-3)
    3     (11*mˆ5+5*mˆ3+4*m)/(20*mˆ6)               1.818*m-.826/m+O(mˆ-3)
    4     (151*mˆ7+70*mˆ5+49*mˆ3+45*m)/(315*mˆ8) 2.086*m-.967/m+O(mˆ-3)
```

Thus, when nfreq ˜= 2*N, the default smoothing using the 4th convolution power of a boxcar smoother of length m, has bandwidth approximately

```
     B = (2.09*m - .97/m)/(nfreq*delta_t) ˜= 1.043*m/(N*delta_t)
```

Equivalent degrees of freedom (EDF)

One common way to summarize the stability of a spectrum estimator is by its equivalent degrees of freedom or EDF. Large EDF means smaller relative standard deviation and thus greater stability.

For a positive random variable W, EDF[W] = 2*E[W]ˆ2/Var[W] = 2/CV[W]ˆ2, where CV[W] = SD[W]/E[W] is the coefficient of variation.

When W = k*Xˆ2, where Xˆ2 is distributed as chisq(f) (chi-squared on f degrees of freedom), EDF[W] = f.

Even when W is not a multiple of chi-squared, its distribution may often be well approximated by that of (E[W]/EDF[W])*chisq(EDF).

When the spectrum is sufficiently smooth, the approximate EDF of a smoothed periodogram with bandwidth B at a frequency f between B/2 and .5/deltat_t - B/2 is approximately 2*B*N*delta_t. At 0 and .5/deltat_t the EDF is B*N*deltat_t. For the default smoother, EDF ˜= (4.17*m-1.93/m)*N. Expressing m in terms of EDF you have the approximate relationship m ˜= EDF/4.17 + 1.93/EDF

Using a Taper or Data Window

In most situations it is desirable to "taper" the time series before computing the periodogram. If yr[i] = y[i] - muhat[i], where muhat[i] is an estimate of E[y[i]], the tapered time series is h[i]*yr[i], where (usually) h[i] tapers smoothly to 0 near i = 1 and i = N. The sequence h[1],...,h[N] is called a taper or a data window.

Tapering reduces "leakage" -- bias at a given frequency arising from variation a distant frequencies. However, it also reduces the EDF of a smoothed periodogram by a factor of approximately

```
             R = sum(hˆ2)ˆ2/{N*sum(hˆ4)} <= 1
```

thus increasing the variance. The greater the amount of tapering, the smaller is R.

EDF for cosine taper

Macro compfa(), which computes estimated spectra and, optionally, cross spectra, uses a "cosine" taper which tapers approximately A*N observations on each end, where 0 <= A <= .5. For this taper, the factor by which the EDF is reduced is

```
 R = (1 - 5*A/8)ˆ2/(1 - 93*A/128) = 1 - 0.5234*A + 0.0103*Aˆ2 + O(Aˆ3).
```

Therefore, for the default smoothing, with 100*A percent tapering on

```
each end and nfreq ~= 2*N, EDF ~= (1 - 0.5234*A)*1.043*m*N.

                              Cross reference
See costaper() for details on the form of a cosine taper.
```

9.5 burg()

Usage:

```
burg(Y, P [,degree:d, nfreq:Nfreq]), Y a REAL vector, P an integer > 0,
  Nfreq an integer > 0 (length of DFT used).
```

Keywords: spectrum analysis, frequency domain, time domain, autoregression

```
                                    Usage
burg(Y,P,degree:D,nfreq:Nfreq) estimates the spectrum of Y considered
as a discrete parameter AR(P) (order P autoregression) time series.

The value returned is a structure with the following components:
   phi       REAL vector of length P containing estimated AR
             coefficients
   var       REAL scalar containing the estimated variance of the res-
             iduals (innovations)
   spectrum  REAL vector of length Nfreq (see below) containing the
             estimated spectrum computed at frequencies 0, 1/Nfreq,
             2/Nfreq, ..., (Nfreq-1)/Nfreq cycles per Delta-t, the
             interval between observations.

Y must be a REAL vector and P an integer > 0.  The keyword phrases are
optional.  See below for the default values of D and Nfreq.

Before estimating the spectrum, Y is detrended by subtracting a degree D
polynomial in time fit by least squares.  D = 0 corresponds to
subtracting the sample mean and D < 0 directs that no detrending is
to be done, not even subtracting a mean.

Nfreq must be an integer >= nrows(Y) + P and must have no prime factors
> 29.

If degree:D is omitted, the default value for D is 0 (subtract the
mean).

                          Default number of frequencies
If nfreq:Nfreq is omitted, the default value for Nfreq = S if S is a
positive integer variable; it is an error if S has a prime factor >
29.

When such an S does not exist, Nfreq = smallest integer >= N + P that
has no prime factors > 29 otherwise, that is Nfreq = goodfactors(N+P),
where N = nrows(Y).
```

```
                                Algorithm
The estimated AR coefficients are computed using an algorithm due to
Burg which does not involve computing the sample autocorrelations.  The
method is sometimes called the maximum entropy method, although that can
equally well describe any method of estimating a spectrum by fitting an
autoregressive model.

                              Cross reference
See also arspectrum(), detrend(), getmacros().
```

9.6 compfa()

Usage:

```
   compfa(y, edf [, degree:D, alpha:A, nfreq:Nfreq, cross:T]), y a REAL
     vector or matrix, nfreq > 0, D integers, edf >=0 and 0 <= A <= .5
```

Keywords: frequency domain, spectrum analysis, cross spectrum

```
                                    Usage
Sf <- compfa(y, edf, degree:D, alpha:A, nfreq:Nfreq) computes spectrum
estimates which are smoothed periodograms of the detrended and tapered
data in the columns of REAL matrix y.

Sf is a Nfreq by ncols(y) matrix whose columns are the estimated
spectra of the detrended and tapered columns of y in Real form.

Nfreq must be a positive integer with no prime factors > 29.

More generally, y can be an array with dimensions n1, n2, n3, ... in
which case the result is also an array with dimensions Nfreq, n2, n3,
... with result[,i2,i3,...] containing the estimated spectrum of
y[,i2,i3,...].

See below for defaults for D, A and Nfreq.

                            Cross spectrum usage
Sf <- compfa(y, edf, cross:T, degree:D, alpha:A, nfreq:Nfreq), with y a
matrix with p > 1 columns and N rows, returns a matrix containing
estimated spectra (smoothed periodograms) and cross spectra (smoothed
cross periodograms) for the tapered detrended columns of y.

It is an error when ncols(y) == 1 (no cross spectrum is defined).

Sy is a Nfreq by Q matrix, where Q = p + p*(p-1)/2 = p*(p+1)/2.

Sy[,j], j = 1,...,p are the estimated spectra of y[,j], in Real form.

Sy[,p+1], Sy[,p+2],...,Sy[,Q] contain the estimated cross spectra of
y[,i] and y[,j], i < j, in Hermitian form.  The order is (i,j) = (1,2),
(1,3), ..., (1,p), (2,3),..., (2,p), ... (p-1,p) so that
Sy[,i*(p-(i+1)/2)+j] is the estimated cross spectrum of y[,i] and y[,j].
```

For example, when x and y are vectors, compfa(hconcat(x,y), edf) returns hconcat(Sxx, Syy, Sxy), where Sxx and Syy are estimated spectra and Sxy is the estimated cross spectrum. Sxx and Syy are in Real form and Sxy is in Hermitian form.

Amount of smoothing

When edf > 2 the amount of smoothing will be chosen so as to yield spectrum estimates with approximatly edf equivalent degrees of freedom. See 'bandwidth'.

When 0 < edf <= .5, edf is interpreted as a bandwidth B in cycles per delta_t and is translated to a working edf = 2*B*N, where N = dim(y)[1]. delta_t is the time interval between successive rows of y. See topic bandwidth.

When edf = 0 or edf = 2, no smoothing of periodograms will be done.

Default keyword values

The values of the optional keywords other than 'cross', are REAL scalars with the following restrictions and default values.

Keyword	Value	Default Value	Restrictions
degree	D	0	Integer
alpha	A	0	0 <= A <= .5.
nfreq	Nfreq	See below	Positive integer with no prime factors > 29

Default for Nfreq when nfreq:Nfreq is not an argument:
When variable S is defined and is a positive integer, Nfreq = S. It is an error if S has a prime factor > 29.

Otherwise, Nfreq is the smallest integer >= 2*nrows(y) that has no prime factor > 29, that is Nfreq = goodfactors(2*nrows(y)).

Detrending

When D > 0, the columns of y are detrended before any tapering with a polynomial of degree D fit by least squares, that is the residuals from the polynomial are tapered and analyzed. When D = 0, the sample mean is subtracted. When D < 0, no detrending is done, not even subtraction of the mean.

Amount of tapering

When A = 0, no tapering is done.

When A > 0, after detrending, each column of y will be multiplied by a cosine taper. The taper is chosen so as to modify approximately A*nrows(y) values on each end of the series, approximately 2*A*nrows(y) values in all.

For example alpha:.5, say, directs that all the observations, except the middle one, when N is odd, are tapered. Similarly, alpha:.1 directs that approximately 10% of the observations at the start and 10% at the end, or 20% in all, will be tapered.

```
NOTE: The interpretation of tapering proportion A differs from that
used by some practitioners, for whom the tapering proportion is the
proportion of the entire series modified that is tapered.  To modify
100*P percent of the entire series, use alpha:P/2.

See topic costaper() for an exact definition of the taper used.

              Comparison with spectrum() and crsspectrum()
Generally compfa() is to be preferred to macros spectrum() and
crsspectrum() because it allows polynomial detrending and cosine
tapering which spectrum() and crsspectrum() lack.  In addition, you
specify the amount of smoothing in terms of bandwidth or EDF instead of
the length of smoothing weights.

                            Other macros used
compfa() uses macros compza(), costaper(), detrend(), read from file
tser.mac if they haven't yet been defined.

                             Cross reference
See also 'complex_data'.
```

9.7 complex_data

Keywords: complex numbers, frequency domain

```
                 MacAnova representation of complex data
An unrestricted complex series (vector) of length N is
represented as a REAL N by 2 matrix with the real and imaginary parts in
columns 1 and 2.

An unrestricted N by p complex matrix (p unrestricted complex series)
is represented by a REAL N by 2*p matrix, with the real and imaginary
parts of the j-th series in columns 2*j-1 and 2*j.  A N by 2*p-1 matrix
can be considered to represent p-1 complex series and 1 real series in
the last column.

When the series represents a frequency function evaluated at Nfreq
frequencies, row k usually contains the values for frequency (k-1)/Nfreq
cycles.

                             Hermitian form
A complex vector with Hermitian symmetry (a Hermitian series) {x(j)} of
length N is represented as a REAL vector, say, hx, of length N with
  hx[1] = x(0)
  hx[j+1] = Re(x(j)), 1 <= j < N/2
  hx[N+1-j] = Im(x(j), 1 <= j < N/2.
  hx[N/2+1] = x(N/2) (only when N is even).
p Hermitian series are represented by the columns of a N by p matrix.

When the series represents a frequency function evaluated at Nfreq
```

```
frequencies, row 1 contains the value for frequency 0 and rows j and
Nfreq+1-j contain the real and imaginary parts of the function at
frequency (j-1)/Nfreq cycles.

                              Cross reference
See also topic 'hermitian'.
```

9.8 complex_fun

Keywords: complex numbers, frequency domain, fourier transforms

```
Here is a brief summary of MacAnova functions and macros for working
with complex data.

                          Notational conventions
  rx is a REAL vector or matrix whose columns represent real series
  hx is a REAL vector or matrix whose columns represent complex series
    with Hermitian symmetry (see topics 'hermitian' and 'complex_data')
  cx is a REAL matrix with successive pairs of columns representing
    unrestricted complex series.  If ncols(cx) is odd, the last column
    represents a real series.

                             Fourier transforms
rft(rx) returns in Hermitian form the DFT of real series in the columns
of rx.

hft(hx) returns as real series the DFT of Hermitian complex series in
hx.

cft(cx) returns as unrestricted complex series the DFT of unrestricted
complex series in cx.

                             Complex conjugation
hconj(hx) and cconj(cx) compute the complex conjugate of Hermitian
series or unrestricted complex series in hx or cx.

                          Real and imaginary parts
hreal(hx) and creal(cx) return the complete real parts of Hermitian
series or unrestricted complex series in hx or cx.

himag(hx) and cimag(cx) return the complete imaginary parts of Hermitian
series or unrestricted complex series in hx or cx.

The results of hreal(), creal(), himag() and cimag() are all
unretricted Real series.

cmplx(rx1,rx2) returns the unrestricted complex form of the series
rx1+i*rx2, where i = sqrt(-1).

                          Conversion to polar form
```

hpolar(hx) and cpolar(cx) return the polar forms of Hermitian series or unrestricted complex series in hx or cx. The amplitudes (moduli) are returned in the real parts and the phases (arguments) are returned in the imaginary parts.

The output of hpolar() is in Hermitian form. The phases are in radians, cycles, or degrees depending on the value of option 'angles'. See regular topic setoptions() for details. By default, the phases are "unwound" to minimize discontinuities if they wrap around the circle.

Conversion from polar form

hrect(hx) and crect(cx) return the usual representation in terms of real and imaginary parts of the polar forms of Hermitian series or unrestricted complex series in hx or cx. The phases of the polar form are assumed to be in radians, cycles or degrees depending on the value of option 'angles'.

Converting to and from hermitian form

htoc(hx) returns the unrestricted complex form of the Hermitian series in hx.

ctoh(cx) returns the Hermitian symmetrization of the unrestricted complex series in cx. If cx has Hermitian symmetry, then htoc(ctoh(cx)) returns cx.

Complex multiplication

hprdh(hx1, hx2) returns the Hermitian form of the elementwise complex product of the Hermitian series in hx1 and hx2. hprdh(hx) returns hprdh(hx,hx).

hprdhj(hx1,hx2) and hprdhj(hx) are equivalent to hprdh(hx1,hconj(hx2)) and hprdh(hx,hconj(hx)), respectively.

cprdc(cx1, cx2) returns the unrestricted complex form of the elementwise complex product of the complex series in cx1 and cx2. cprdc(cx) returns cprdc(cx,cx).

cprdcj(cx1,cx2) and cprdcj(cx) are equivalent to cprdc(cx1,cconj(cx2)) and cprdc(cx,cconj(cx)), respectively.

Complex division

hdivh(hx1, hx2) returns the Hermitian form of the elementwise complex ratio of the Hermitian series in hx1 and hx2. hdivh(hx) returns hdivh(hx,hx). When hx is a vector and no represented complex elements are 0, cdivh(hx,hx) is the Hermitian form of rep(1,nrows(hx)).

hdivhj(hx1,hx2) and hdivhj(hx) are equivalent to hdivh(hx1,hconj(hx2)) and hdivh(hx,hconj(hx)), respectively.

cdivc(cx1, cx2) returns the unrestricted complex form of the elementwise complex product of the complex series in cx1 and cx2. cdivc(cx) returns cdivc(cx,cx). When cx has 1 or 2 columns, with no zero rows, cdivc(cx,cx) is the fully complex form of rep(1,nrows(cx)).

cdivcj(cx1,cx2) and cdivcj(cx) are equivalent to cdivc(cx1,cconj(cx2)) and cdivc(cx,cconj(cx)), respectively.

Macros for complex matrices

In the following a and b are REAL vectors or matrices representing complex matrices A and B in unrestricted complex form. All the macros return their result in unrestricted complex form.

cmatmultc(a,b [,op:"%*%"), cmatmultc(a,b,op:"%c%") and cmatmultc(a,b, op:"%C") return matrix products of A and B.

ctrace(a) returns the trace of A.

cdiag(a) returns the complex diagonal of square A.

ctranspose(a) returns the transpose of A.

cjtranspose(a) is equivalent to ctranspose(cconj(a)).

csubscr(A), csubscr(A,i), csubscr(A,i,j) simulate A[i], A{i,], and A[i,j]. i and or j can be empty.

csolve(a) returns the inverse of non-singular square A.

ceigen(a) returns a structure containing the real eigenvalues and complex eigenvectors of a Hermitian matrix A (ctranspose(a) = cconj(a)).

Additional functions useful in time series analysis

autoreg()

autoreg(phi,x) applies an autoregressive operator specified by REAL vector phi of length p to the columns of REAL matrix x. Keywords 'reverse', 'limits' and 'start' allow for reversing the direction, operating on a subset of rows and providing starting values.

Uses for autoreg() include generation of autoregressive (AR(p)) time series (x random), computing residuals from a moving average (MA(p) time series with coefficients phi, computing cumulative sums (phi = 1) and solving difference equations.

The sign convention used corresponds to ARSIGN = -1. See arimahelp topic 'MASIGN'.

convolve()

convolve(wts,x) returns the circular convolutions of REAL vector wts and the columns of REAL matrix x.

convolve(wts,x,reverse:T) returns the sums of circularly lagged products of wts with the columns of x. With keyword phrase 'decimate:n' as an argument it returns only rows 1, n+1, 2*n+1, ... of the convolutions.

```
                                movavg()
movavg(theta,x) applies an moving average operator specified by REAL
vector theta of length q to the columns of REAL matrix x.  Keywords
'reverse', 'limits' and 'start' allow for reversing the direction,
operating on a subset of rows and providing starting values.

Uses for movavg() include generation of moving average (MA(q)) time
series (x random), computing residuals from a autoregressive (AR(q))
time series with coefficients theta, computing first differences (theta
= 1), and computing non-circular convolution.

The sign convention used corresponds to MASIGN = -1.  See arimahelp
topic 'MASIGN'.

                                padto()
padto(x,n) returns matrix x padded with n additional rows of zeros.  If
n < nrows(x), the result consists of the first n rows of x.

padto() is useful in computing Fourier transforms at Nfreq frequencies
where Nfreq > nrows(x), for example, rft(padto(x,2*nrows)) computes the
Fourier transforms of the columns of x at Nfreq = 2*nrows equally spaced
frequencies.

                                partacf()
phi_kk <- partacf(rho) computes the partial autocorrelations
corresponding to autocorrelations in the column or columns of REAL
vector or matrix rho.

rho <- partacf(phi_kk,reverse:T) computes the autocorrelations
corresponding to partial autocorrelations in the column or columns of
REAL or matrix vector phi_kk.

                               polyroot()
polyroot(coefs) returns a REAL matrix with 2*ncols(coefs) columns
containing the complex zeros of real polynomials whose coefficients are
specified by the columns of REAL vector or matrix coefs.

polyroot() is useful in determining whether a MA operator is invertible
or a AR operator is stationary.

REAL vector theta defines an invertible MA operator if and only if
   max(creal(cpolar(polyroot(theta)))) < 1.

REAL vector phi defines a stationary AR operator if and only if
   max(creal(cpolar(polyroot(phi)))) < 1.

The sign convention used corrsponds to MASIGN = -1 and ARSIGN = -1 and
is not affected by options 'masign' and 'arsign.  See help topic
'MASIGN'.

                                reverse()
reverse(x) returns a vector or matrix whose rows are the same as those
of REAL or LOGICAL vector or matrix x but in reverse order.
```

unwind()

unwind(theta) returns a vector or matrix derived from the columns of REAL vector or matrix theta interpreted as angles in units of radians, cycles or degrees depending on option 'angles'. Multiples of 2*PI radians, 1 cycle or 360 degrees are added or subtracted to remove large row to row jumps. You can specify the size of jumps using keyword 'crit'.

yulewalker()

phi <- yulewalker(rho) computes a vector or matrix whose columns are the autoregression coefficients that solve the Yule-Walker equations based on autocorrelations in the corresponding column of REAL vector or matrix rho.

rho <- yulewalker(phi,reverse:T) computes a vector or matrix autocorrelations that satisfies phi = yulewalker(rho).

Informational Topics

Cross reference

See topic 'hermitian' for the definition of Hermitian symmetry.

See topic 'fourier' for information on Fourier transforms.

See topic 'complex_data; for information on representing complex data.

See regular MacAnova help topics for full information on cft(), hft(), rft(), creal(), hreal(), cimag(), himag(), cmplx(), cpolar(), hpolar(), crect(), hrect(), ctoh(), htoc(), cprdc(), cprdcj(), cdivc(), cdivcj(), hprdh(), hprdhj(), hdivh(), hdivhj(), autoreg(), movavg(), convolve(), padto(), partacf(), polyroot(), reverse(), unwind() and yulewalker().

9.9 compza()

Usage:

compza(y [,degree:D, alpha:A, nfreq:Nfreq]), y a REAL vector or matrix, D and Nfreq != 0 integers, 0 <= A <= .5

Keywords: frequency domain, spectrum analysis

Usage

compza(y,nfreq:Nfreq,degree:D,alpha:A) computes scaled Fourier transforms of the columns of REAL matrix y after detrending and tapering them. They are scaled by dividing by sqrt(Ka) where Ka = sum(taper^2).

The number of frequencies at which the Fourier transform is computed is abs(Nfreq). It is an error if Nfreq > 0 and has a prime factor > 29.

Integer D = degree of polynomial used to detrend the columns of y. D < 0 means no detrending, not even subtracting a mean.

```
Scalar A >= 0 is the proportion of tapering on each end of the time
series.

See below for defaults for Nfreq, A and Nfreq.

The value is structure(za:Za, n:nrows(y), ka:Ka, alpha:A, degree:D)
  Za        The Nfreq by ncols(y) matrix of Fourier transforms of the
            detrended and then tapered columns of y, in Hermitian form
  Ka        sum(W^2), where vector W are the tapering factors

                         Default keyword values
The values of the optional keyword phrase arguments are REAL scalars
with the following restrictions and default values.
 Keyword  Value     Default Value      Restrictions
 degree     D             0           Integer
 alpha      A             0           0 <= A <= .5.
 nfreq      Nfreq      See below      Integer != 0 with abs(Nfreq)
                                      not having prime factors > 29

When nfreq:Nfreq is not an argument and positive integer variable S
does not exist, Nfreq is the smallest integer >= 2*nrows(y) with no
prime factor > 29, that is Nfreq = goodfactors(2*nrows(y)).

When S does exist and is a positive integer, nFreq = S.  It is an error
if S has a prime factor > 29.

                                 Detrending
When D > 0, the columns of y are detrended with a polynomial in time of
degree D fit by least squares, that is the residuals from the polynomial
are tapered and analyzed.  When D = 0, the sample mean is subtracted.
When D < 0, no detrending is done, not even subtraction of the mean.

                                  Tapering
When A = 0, no tapering is done.

When A > 0, after detrending, each column of y will be multiplied by a
cosine taper which modifies approximately A*nrows(y) values on each end
of the series, approximately 2*A*nrows(y) values in all.  Thus alpha:.5,
say, means that the all the observations, except the middle one, when N
is odd, are tapered.  Similarly, alpha:.1 directs that 10% of the
observations at the start and 10% at the end, or 20% in all, will be
tapered.

NOTE: The interpretation of tapering proportion A differs from that
used by some practitioners, for whom the tapering proportion is the
proportion of the entire series modified that is tapered.  To modify
100*P percent of the entire series, use alpha:P/2.

See topic costaper() for an exact definition of the taper used.

                                   Method
After detrending followed by tapering, enough zero rows are added to the
```

```
end to bring the total number of rows to Nfreq and then rft() is used to
compute the Hermitian form of the Fourier transforms of each column.
These are sometimes known as "modified Fourier transforms".  Finally
they are divided by sqrt(Ka), where Ka = sum(taper^2); see above.

                              Other macros used
compza() uses macros detrend() and costaper().  If they have not
previously been loaded, they are read from file tser.mac.

                               Cross reference
See also detrend(), costaper(), 'hermitian', 'complex_fun',
'complex_data', rft(), 'complex'.
```

9.10 costaper()

Usage:

```
costaper(N, alpha), N > 0 an integer, 0 <= alpha <= .5 a REAL scalar.
```

Keywords: tapering, frequency domain, spectrum analysis, time domain

```
                                    Usage
costaper(N, alpha) returns a vector h containing a 100*alpha percent
cosine taper (data window) of length N, tapering approximately N*alpha
elements on each end.  N must be a positive integer and alpha a REAL
scalar with 0 <= alpha <= .5.

When alpha = 0, h = rep(1,N), a "taper" that does no tapering.

For 0 < alpha <= .5 and L = ceiling(alpha*N),
   h[j] = .5*(1 - cos(PI*(j-.5)/L) = sin(.5*PI*(j-.5)/L)^2 , 1 <= j <= L
   h[j] = 1, L + 1 <= j <= N - L
   h[j] = h[N-j+1], N - L + 1 <= j <= N

A taper of length N is used to multiply a (usually) detrended time
series of length N before computing its discrete Fourier transform.

                  Interpretation of tapering proportion
NOTE: The interpretation of tapering proportion alpha differs from that
used by some practitioners, for whom the tapering proportion is the
proportion of the entire series modified that is tapered.  To compute a
cosine taper to modify 100*P percent of a series of length N, use
costaper(N, P/2).
```

9.11 crosscor()

Usage:

```
ccf <- crosscor(x [,nlags] [,degree:d] [,auto:T] [,full:T or center:T]\
  [,nfreq:M]), REAL matrix x, integer nlags > 0, integer scalar or vector
  d, integer M > 0
ccf <- crosscor(x, i, j [,nlags] [,degree:d] [,full:T  or  center:T]\
  [,nfreq:M]), integers i > 0, j > 0
```

Keywords: autocorrelation, crosscorrelation, autocovariance, crosscovariance, time domain

```
                                Usage
r_yy <- crosscov(y, nlags) computes sample auto- and crosscorrelation
functions ccf(h,y[,i],y[,j]) for lags h = 0, 1, ..., nlags.

The columns of y, a N by ny REAL matrix with no MISSING elements, are
interpreted as time series defined at equally spaced time points.

r_yy will be a nlags+1 by ny by ny REAL array, with r_yy[h+1,i,j] =
ccf(h,y[,i],y[,j]), h = 0, ..., nlags, where
  ccf(h,y[,i],y[,j]) =
    ccvf(h,y[,i],y[,j])/sqrt(ccvf(h,y[,i],y[,i])*ccvf(h,y[,j],y[,j])),
with
   ccvf(h,y[,i],y[,j]) =
       sum((y[l+h,i]-ybar[i])*(y[l,j]-ybar[j]),l=1,...,N-h)/N.
Note the divisor is N, not N-h or N-h-1.  ybar[j] = sum(y[,j])/N is the
sample mean of y[,j].

Note that y[,j] lags h behind y[,i].  Correlations where y[,j] leads h
ahead of y[,i] are in r_yy[h+1,j,i].

r_yy <- crosscor(y) does the same, except nlags = N-1.

In contrast to the output of autocor(), r_yy contains lag 0 correlations.

r_yy <- crosscor(y, i, j [,nlags]) computes ccf(h,i,j) for h =
-nlags,...,0,...,nlags as a vector of length 2*nlags+1.  Without
center:T (see below), results for lags 0, ..., nlags are in rows 1
through nlags+1 and those for lags -1, ..., -nlags are in rows
2*nlags+1, 2*nlags, ..., nlags+2.  If you want ccf(h,i,j) only for h >=
0, use full:F as an argument.

                               Keywords
You can also use crosscov() keyword phrases 'degree:d' (to control
detrending), 'auto:T' (to restrict computations to autocorrelations),
'full:T' (to compute results for negative lags), 'center:T' (to center
lag 0 between negative and positive lags) and 'nfreq:M' (to specify the
length of Fourier transforms used).  See crosscov() for details.

                              Computation
crosscor() is implemented as a macro which uses crosscov() with keyword
phrase 'cor:T' to compute auto- and cross-correlations using discrete
Fourier transforms.  See subtopic 'crosscov:"computation" for details
```

```
and for description of the use of keyword 'nfreq'.

                              Cross reference
See also crosscov(), autocor(), autocov().
```

9.12 crosscov()

Usage:

```
   ccvf <- crosscov(y [,nlags] [,degree:d] [,auto:T] [,cor:T]\
     [,full:T or center:T] [,nfreq:M]), REAL matrix y, integer nlags > 0,
     integer scalar or vector d, integer M > 0
   ccvf <- crosscov(y, i, j [,nlags] [,degree:d] [,cor:T] \
     [,full:T  or  center:T] [,nfreq:M]), integers i > 0, j > 0
```

Keywords: crosscovariance, crosscorrelation, autocovariance, autocorrelation, time domain

```
                                    Usage
c_yy <- crosscov(y, nlags) computes sample auto- and crosscovariance
functions ccvf(h,y[,i],y[,j]) for lags h = 0, 1, ..., nlags.

The columns of y, a N by ny REAL matrix with no MISSING elements, are
interpreted as time series defined at equally spaced time points.

c_yy will be a nlags+1 by ny by ny REAL array, with c_yy[h+1,i,j] =
ccvf(h,y[,i],y[,j]), h = 0, ..., nlags, where
   ccvf(h,y[,i],y[,j]) =
       sum((y[l+h,i]-ybar[i])*(y[l,j]-ybar[j]),l=1,...,N-h)/N.
Note the divisor is N, not N-h or N-h-1.  ybar[j] = sum(y[,j])/N is the
sample mean of y[,j].

Note that y[,j] lags h behind y[,i].  Covariances where y[,j] leads h
ahead of y[,i] are in c_yy[h+1,j,i].

crosscov(y) is the same as crosscov(y,nrows(y)-1), computing all
non-degenerate sample autocovariances.

c_yy <- crosscov(y, i, j [,nlags]) computes ccvf(k,i,j) for k =
-nlags,...,0,...,nlags as a vector of length 2*nlags+1.  Without
center:T (see below), results for lags 0, ..., nlags are in rows 1
through nlags+1 and those for lags -1, ..., -nlags are in rows
2*nlags+1, 2*nlags, ..., nlags+2.  If you want ccvf(k,i,j) only for k >=
0, use full:F as an argument.

                                 Keyword auto
c_yy <- crosscov(y [,nlags], auto:T) computes only autocovariances.  The
result is a matrix with ny columns with c_yy[h+1,j] = ccvf(h,
y[,j],y[,j]), h = 0,...,nlags, j = 1,...,ny.

crosscov(y, i, j [,nlags], auto:T) is legal only when i = j.
```

Keywords full and center

c_yy <- crosscov(y [,nlags] full:T) does the same, except auto- and cross-covariances are also computed for negative lags.

c_yy will be a 2*nlags+1 by ny by ny array, with results for lags 0 through nlags in c_yy[1,,], c_yy[2,,],..., c_yy[nlags+1,,], and results for lags -1 through -nlags in c_yy[2*nlags+1,,], c_yy[2*nlags,,],..., c_yy[nlags+2,,].

c_yy <- crosscov(y [,nlags] center:T) does the same as with full:T, except the auto- and cross-covariance functions are centered at row nlags+1. Specifically, results for lags -nlags, -nlags+1, ..., -1, 0, 1, ..., nlags are in c_yy[1,,], c_yy[2,,], ..., c_yy[2*nlags+1,,]

With auto:T, with either full:T or center:T, c_yy will be a 2*nlags+1 by ny matrix.

Keyword degree

You can specify that columns of y be detrended by subtracting a polynomial in time using keyword phrase 'degree:d', where d is an integer scalar or vector of length ny = ncols(y). A scalar d is interpreted as rep(d,ny).

c_yy <- crosscov(y [,nlags], degree:d) computes auto- and cross-covariances of the residuals from polynomial trends fit by least squares to each column of y. The values of the fitted polynomial replace ybar in the definition of ccvf.

d[j] is the degree of the polynomial fit to column j. When d[j] = 0, only the sample mean is subtracted (default). When d[j] < 0, nothing is subtracted.

You can use keyword phrases 'full:T', 'center:T' and 'auto:T' with 'degree:d'

Correlations

r_yy <- crosscov(y [,i,j] [,nlags], cor:T ...) computes auto- and cross-correlations instead of auto- and cross-covariances. You can use any of the other keywords with 'cor:T'.

Note that, unlike autocor(), the result includes lag 0 correlations.

Computational method

The necessary sums of lagged products are computed using discrete Fourier transforms (DFTs) of length >= N + nlags, the actual length chosen being M = goodfactors(N + nlags). See goodfactors().

When working with long time series, it may be possible to find M1 > M with more small factors resulting in faster DFT computation. For example, when N + nlags = 24388, M = goodfactors(24388) = 24389 = 29^3, but a DFT of length M1 = 24576 = 3*2^13 takes only about 40% as long to compute as a DFT of length M.

```
c_yy <- crosscov(y[,i,j] [,nlags], nfreq:M1 ...) uses DFTs of length M1.

                              Cross reference

See also crosscor(), autocov(), autocor().
```

9.13 crsspectrum()

Usage:

```
crsspectrum(y, len [,reps]), y a REAL matrix with ncols(y) >= 2, len > 0
  and reps > 0 integers.
```

Keywords: frequency domain, spectrum analysis, cross spectrum

```
                                    Usage
crsspectrum(y, len) computes smoothed periodogram and cross-periodogram
estimates of the spectrum and cross-spectrum of the multivariate time
series in the columns of REAL matrix y.

Smoothing is accomplished by 4 successive convolutions of a "boxcar"
smoother of length len.  That is, the smoothing weights are
rep(1/len,len).  The total number of frequencies involved in each
spectrum estimate is 4*len - 3.  When len = 1 no smoothing is done.

crsspectrum(y, len, reps) does the same, except that reps convolutions
with a boxcar are used, where reps > 0 is an integer scalar.  When reps
is odd, then len must also be odd.

See topic 'bandwidth' for information on the approximate bandwidth and
equivalent degrees of freedom associated with these estimates.

                              Form of result
Suppose p = ncols(y) and N = nrows(y).  Then, after Sy <-
crsspectrum(y, len [,reps]), Sy is a N by Q matrix, where Q = p +
p*(p-1)/2 = p*(p+1)/2.

Sy[,j], j = 1,...,p are the estimated spectra of y[,j], in Real form.

Sy[,p+1], Sy[,p+2],...,Sy[,Q] contain the estimated cross spectra of
y[,i] and y[,j], i < j, in Hermitian form.  The order is (i,j) = (1,2),
(1,3), ..., (1,p), (2,3),..., (2,p), ... (p-1,p)

Specifically, the estimated cross spectrum of y[,i] and y[,j] is in
column i*(p - (i + 1)/2) + j.

For example, when x and y are vectors, crsspectrum(hconcat(x,y), len)
returns hconcat(Sxx, Syy, Sxy), where Sxx and Syy are estimated spectra
and Sxy is the estimated cross spectrum.  Sxx and Syy are in Real form
and Sxy is in Hermitian form.

                          Detrending and tapering
```

```
The column means are subtracted before computing the periodograms and
cross-periodograms, but no tapering or other detrending is done.

                        Length of fourier transform
The spectra and cross spectra are computed at the Nfreq Fourier
frequencies 0, 1/Nfreq, 2/Nfreq, ..., (Nfreq-1)/Nfreq cycles per unit
time, were Nfreq is determined as follows:

  When variable S is defined and is a positive integer, Nfreq = S.  It
  is an error if S has a prime factor > 29.

  Otherwise, Nfreq = smallest integer >= 2*nrows(y) which has no prime
  factors > 29, that is Nfreq = goodfactors(2*nrows(y)).

                        Comparison with compfa()
Most of the time, a better choice for estimating cross spectra is macro
compfa() which includes tapering and detrending and for which the amount
of smoothing is specified as a bandwidth or an equivalent degrees of
freedom.  See compfa() for details.
```

9.14 detrend()

Usage:

```
detrend(x [,Degree]), REAL vector or matrix x, integer scalar or vector
  Degree
```

Keywords: time domain

```
                                    Usage
y1 <- detrend(y, k, time:tt) detrends each column of REAL vector or
matrix by subtracting a degree k polynomial P_k(tt).  Each column of y,
which can have no MISSING values, is considered as a time series
observed at time points tt[1], tt[2], ..., tt[N], where N = nrows(y).
tt is a REAL vector of length N and with no MISSING values.  y1 has the
same size and shape as y.

k is usually an integer scalar but can be a length ny = ncols(y) vector
of integers, in which case column j is detrended by a degree k[j]
polynomial.

When k = 1, a linear trend is removed.  When k = 0, only the sample mean
is subtracted.  When k < 0, nothing is subtracted and y1 = y.

Each polynomial is fit by least squares.

You can find the trend values themselves by y - detrend(y,i,time:tt).

detrend(y, time:tt) is equivalent to detrend(y, 1, time:tt), subtracting
least squares regression lines a + b*tt from columns of y.

detrend(y [,k]) is equivalent to detrend(y [,k], time:run(nrows(y))).
```

```
This is what you should use with discrete parameter time series observed
at equally spaced time points.

Caution: Some early versions of detrend() used k = 0 as the default,
so that detrend(y) removed only the column means.  It was changed
because it seemed inconsistent with the usual meaning of "detrend".

                                 Examples
Compute residuals from linear trend:

  Cmd> detrended <- detrend(y) # or detrend(y,1,time:run(nrows(y)))

Plot a discrete parameter monthly time series starting January 1995
together with cubic trend:

  Cmd> tsplot(hconcat(y,y-detrend(y,3)),1995,1/12,title:"Time series")

Detrend y[,1] linearly, y[,2] using a cubic polynomial and y[,3] not at
all:

  Cmd> y1 <- detrend(y,vector(1,3,-1))

                              Cross reference
See also regress(), orthopoly(), tsplot().
```

9.15 dpss()

Usage:

```
dpss(N, W, K [,First]), N >= 1, 1 <= K <= N, 1 <= First <= N - K + 1
  integers, 0 < W < .5
```

Keywords: tapering, spectrum analysis, frequency domain, tapering

```
                                    Usage
dpss(N, W, K) computes Discrete Prolate Spheroidal Sequences (DPSS) with
half bandwidth W for orders 0, 1, ..., K-1.  The order L DPSS is the
L+1-th orthonormal eigenvector of a certain tridiagonal symmetric matrix
of order N (see below).  The value returned is N by K REAL matrix, with
the first K eigenvectors in columns 1, 2, ... K in order of decreasing
eigenvalue.

DPSS sequences are used as tapers in multi-taper spectrum estimation.
See below.

dpss(N, W, K, J) does the same, except the DPSS have orders J-1, J, ...,
J+K-2, that is they are eigenvectors J, J+1, ..., J+K-1 of the
tridiagonal matrix.  It returns a N by K REAL matrix, with eigenvectors
J, J+1,..., J+K-1 in columns 1, 2, ... K.
```

```
N, K and J must be integers satisfying N >= 1, 1 <= K <= N, 1 <= J <= N
- K + 1.

W must be a REAL scalar, 0 < W < 0.5.

                              Use as tapers
Discrete Prolate Spheroidal Sequences (DPSS) are used as tapers (data
windows) in multi-taper spectrum estamation.  Their continuous Fourier
transforms are very highly concentrated in low frequencies with a very
sharp cutoff near frequencies W and -W cycles.  Because they are
eigenvectors of a symmetric matrix, they are orthogonal.

The diagonal and subdiagonal of the tridiagonal matrix are
    d = cos(2*PI*W)*(.5*run(-N+1,N-1,2))^2
and
    e = (run(N-1)*run(N-1,1))/2

The DPSS are computed by trideigen(d,e,J,J+K-1,values:F), followed by
certain sign changes.  See regular topic trideigen().
```

9.16 evalpoly()

Usage:

```
   evalpoly(coef,z), coef and z REAL matrices with ncols(z)=2*ncols(coef)
   evalpoly(coef,z,T), coef and z REAL matrices with ncols(z)=ncols(coef)
```

Keywords:

```
                                    Usage
evalpoly(coef,z) evaluates polynomials with REAL coefficients in coef
for complex arguments specified by z.

Specifically, when coef is a n by p REAL matrix and z is a N by 2*p
REAL matrix considered as representing a N by p complex matrix Z,
evalpoly(coef,z) evaluates
  Z^n - coefs[1,]*Z^(n-1) - ... - coef[n-1,]*Z - coef[n,],

The result is a N by 2*p REAL matrix representing a N by p complex
matrix.

By definition, evalpoly(coef,polyroot(coef)) should be zero within
rounding error.

It is an error when ncols(z) != 2*ncols(coef).

evalpoly(coef,x,T) does the same except x is considered to be a real
rather than complex matrix with ncols(x) = ncols(coef).  The result
is a matrix with the same dimensions as x and is the same as
creal(evalpoly(coef,cmplx(x)))

                              Cross reference
```

See also topics 'complex_data', cmplx(), creal() and polyroot().

9.17 ffplot()

Usage:

```
ffplot(G [, Range[, delta_t]] [,timeunit:Unit] [,plotting keywords]),
  G a REAL matrix, Range a real scalar or vector of length 2, REAL
  scalar delta_t > 0, CHARACTER scalar Unit
```

Keywords: frequency domain, plotting, complex numbers

Usage

ffplot(G, Range, delta_t) plots the columns of REAL matrix G considered as functions of frequency f for values of f specified by Range.

If N = nrows(G), the i-th row of G is associated with frequency (i-1)/N cycles per delta_t time units, or (i-1)/(N*delta_t) cycles per unit time so the full range of frequencies in G is assumed between 0 and ((N-1)/N)/delta_t.

When Range is vector(f1, f2), G is plotted for all Fourier frequencies between f1 and f2, inclusive.

Range = f, where f is a non-zero scalar, is equivalent to Range = vector(0,f).

Range = 0 is equivalent to Range = vector(0,.5/delta_t).

You can provide a default value for delta_t by setting variable DELTAT appropriately. You can also provide a time unit to be used in constucting the default x-axis label by setting variable TIMEUNIT. See below.

You can use the usual graphic keywords, including 'title', 'xlab', 'ylab', 'xmin', 'xmax', 'ymin', 'ymax', and 'linetype'.

Keyword timeunit

ffplot(G,Range,delta_t,timeunit:Unit), where Unit is a CHARACTER scalar such as "year", does the same, the default x-axis label will be, say, "Frequency (cycles/year)".

Unit should be specified consistantly with delta_t. For example, with monthly data and delta_t = 1/12, Unit should be "year", while with delta_t = 1, Unit should be "month".

Defaults for arguments

You can omit delta_t or both Range and delta_t, and provide a default time unit.

When you omit argument delta_t (ffplot(G, Range)), the default for delta_t is variable DELTAT if it is a positive scalar and is 1

```
otherwise.

When you omit both arguments Range and delta_t (ffplot(G), the defaults
for Range and delta_t are vector(0,.5/DELTAT) and DELTAT, when DELTAT
is a positive scalar, and vector(0, .5) and 1 otherwise

Without keyword 'timeunit', the default x-axis label will be
constructed from the value of CHARACTER scalar TIMEUNIT, if it exists
and differs from "".

Macro tsplot() also uses variables DELTAT and TIMEUNIT, as well as
variable START, to construct a default title and x-axis label.  You
can set them once and forget about them.  For example
  Cmd> START <- 1991; DELTAT <- 1/12; TIMEUNIT <- "year"

to ensure that the x-axis label for both frequency and time domain
plots will be informative.

                      Plotting outside normal range
If some or all of Range is outside of the interval (0,.5/delta_t), the
values plotted are the periodic extension (with period N) of each column
of G.  Thus ffplot(G,vector(-.5,.5), 1) is legal and plots the a full
cycle of each column from frequency -.5 to .5, with the values with
frequencies < 0 coming from rows i with i > N/2.

                            Hermitian argument
If the columns of G are complex in Hermitian form, ffplot(G, Range) will
produce the same plot as ffplot(hreal(G), Range) as long as the range
specified is contained in the interval (0, .5/DELTAT).  See
'complex_data'.

                              Cross reference
See also tsplot().
```

9.18 fourier

Keywords: frequency domain, fourier transforms, complex numbers

```
                      Discrete Fourier Transform (DFT)
Let {x(t)} = {x(0), x(1), ..., x(N-1)} = {x(t),0<=t<=N-1}} be a finite
complex or real series, that is, a series of N complex or real numbers.
Then the DFT (discrete Fourier transform) of {x(t)} is the finite
complex series of length N {X<N>(k),0<=k<=N-1}, where

   X<N>(k) = sum(x(t)*exp(-i*2*PI*t*k/N),0<=t<=N-1), k = 0,1,...,N-1

In the argument to exp(), i = sqrt(-1) is a pure imaginary number.

For any integer M > 0 we define
     X<M>(k) = sum(x(t)*exp(-i*2*PI*t*k/M),0<=t<=N-1), k = 0,1,...,M-1
```

When M > N, this can be viewed as the DFT of the series of length M obtained by "padding" {x(t),0<=t<=N-1} with M - N 0's.

There are two aspects of this notation:
The use of an upper case X for the Fourier transform of lower case x.
The use of suffix <M> to specify that f = k/M is used in the complex exponential exp(-i*2*PI*t*f).

The alternative notation DFT<M>{x}(k) = X<M>(k) is sometimes useful.

The definition of x<M>(k) works for any integer k < 0 and k >= M. With this extension, x<M>(k) is periodic with period M, that is for any k
x<M>(k+M) = x<M>(k-M) = x<M>(k).

Note that in the definition of the DFT, indices start with 0 rather than 1 as is assumed in MacAnova. In working with Fourier transforms in MacAnova, this correspondence must be kept in mind. The first element in a series is always considered to be x(0) or X(0), but to access it you will need to use x[1].

Inversion formula

If {X<N>(k)},k = 0, 1, ..., N-1 is the DFT of {x(t)},t = 0, 1, ..., N-1, then the following inversion formula holds:

```
x(t) = (1/N)*sum(X<N>(t)*exp(+i*2*PI*t*k/N),0<=k<=N-1)
     = (1/N)*conj(DFT<N>{conj(DFT<N>{x})}(t)), t  = 0, 1, ..., N-1,
```

where conj(x) is the complex conjugate of the complex number z.

More generally, when M > N

```
x(t) = (1/M)*sum(X<M>(t)*exp(+i*2*PI*t*k/M),0<=k<=M-1)
     = (1/M)*conj(DFT<M>{conj(DFT<M>{x})}(t)), t = 0, 1, ..., N-1,
```

For t = N,...,M-1, the sum is 0.

Periodic Extensions

When dealing with the DFT it is sometimes convenient to consider a finite complex or real series as being a segment of length N from a periodic infinite complex or real series {x<N>(t),-oo<t<oo} with period N:

```
x<N>(t) = x(t),      t  = 0, 1, ..., N-1
x<N>(t) = x<N>(t-N), t = N, N+1, ...
x<N>(t) = x<N>(t+N), t = -1, -2, ...
```

More generally, we can define x<M>(t) as the periodic extension with period M of x(t) padded with M-N 0's:

```
x<M>(t) = x(t),      t  = 0, 1, ..., N-1
x<M>(t) = 0,         t = N,...,M-1
x<M>(t) = x<M>(t-M), t = M, M+1, ...
x<M>(t) = x<M>(t+M), t = -1, -2, ...
```

With this extended definition, for any integer t0

```
  X<M>(k) = sum(x<M>(t)*exp(-i*2*PI*t*k/M),t0 <= t <= t0+M-1)

that is, as a summation over an arbitrary complete period of x<M>(t).

                    Continuous Fourier Transform (CFT)
The continuous Fourier transform of a finite real or complex series
{x(t),0<=t<=N-1} is the continuous function of the real variable f

        X^(f) = CFT{X}(f) = sum(x(t)*exp(-i*2*PI*t*f),0<=t<=N-1)

The argument f is the frequency at which X^(f) is evaluated and is in
units of cycles.

X^(f) is a periodic function of f with period 1, that is
  X^(f) = X^(f+k) = X^(f-k) for any integer k.

You can consider the DFT to be a "sampling" of the CFT, in the sense
that X<M>(k) = X^(k/M).

Because X<M>(k), k = 0, ..., M-1 is the DFT of x(t) "padded" with M - N
0's, by padding {x(t)} with enough additional zeros, you can use the DFT
to compute the CFT at an arbitrarily dense set of frequencies. For
purposes of computing spectra of a series of length N it is usually
desirable to compute Fourier transforms at approximately M = 2*N
frequencies.  Function padto() is useful for adding zero rows to a
vector or matrix.

You can define the continuous Fourier transform of an infinite real or
complex series {x(t), -oo < t < oo} in a similar manner as
  X^(f) = sum(x(t)*exp(-i*2*PI*t*f), -oo < t < oo)
when the infinite sum converges as it always will if only x(t) != 0 for
only a finite set of t's.

In another terminology, {x(t), -oo < t < oo} are the Fourier coeficients
of the periodic function X^(f).
```

9.19 gettsmacros()

Usage:

```
   gettsmacros(name1 [,name2 ... ]), name1, name2 ... unquoted of macro
     names on file TSMACROS or "tser.mac" if CHARACTER scalar TSMACROS does
     not exist
```

Keywords: general

```
                                  Usage
gettsmacros(Macro1,Macro2,...) retrieves macros Macro1, Macro2, ... from
a file.  The file name is the value of CHARACTER scalar TSMACROS if it
exists or "tser.mac" if it does not.  The macro names must not be
enclosed in quotes or be CHARACTER variables.
```

```
gettsmacros(Macro1,Macro2,...,quiet:T) retrieves the macros but
suppresses printing the descriptive comments associated with them.

If there is more than one copy of any of the named macros in the file,
getttmacros retrieves the first one found.

TSMACROS has no predefined value.

Example:  If TSMACROS does not exist or has value "tser.hlp",
  Cmd> gettsmacros(detrend, spectrum, costaper)
retrieves macros detrend(), spectrum() and costaper() from file
"tser.mac".

gettsmacros() should be faster than pre-defined macro getmacros()
because it searches only one file.  See MacAnova help topic getmacros().
```

9.20 hermitian

Keywords: frequency domain, fourier transforms, complex numbers

```
                                Definition
A complex series {y(j),0<=j<=N-1} of length N that satisfies

         y(0) real and y(n-j) = conj(y(j)), j = 1, ..., N-1

is said to have Hermitian symmetry, or simply to be a Hermitian series.
Here conj(z) denotes the complex conjugate of complex number z.

This should not be confused with the Hermitian symmetry of a square
complex matrix A for which A' = complex conjugate of A.

When N is even, y(N/2) is real for Hermitan y.

If you define the periodic extension as the infinite complex series
  y<N>(j) = y(j),      j = 0, ..., N-1
  y<N>(j) = y<N>(j-N), j = N, N+1, ...
  y<N>(j) = y<N>(j+N), j = -1, -2, ...

then Hermitian symmetry is equivalent to

  y<N>(-j) = conj(y<N>(j)), j = 0, +-1, +-2, ...

If {x(t),0<=t<=N-1} is a real series, then its DFT, {x<N>(t)} is a
Hermitian series.  Conversely if {x(t),0<=t<=N-1} is Hermitian, then
{x<N>(t)} is real.  See topic 'fourier' for information on the DFT.

If {x(t),0<=t<=N-1} is an unrestricted complex series, its Hermitian
symmetrized form is the Hermitian series {y(t),0<=t<=N-1} where
```

```
    y(0) = Re(x(0)), y(t) =(1/2)(x(t)+conj(x(N-t))), t = 1, ..., N-1

Using the periodic extensions {x<N>(t)} and {y<N>(t)}, this is
equivalent to

          y<N>(t) = (x<N>(t)+x<N>(-t))/2, j = 0, +-1, +-2, ...

                              Cross reference
See topic 'complex_data' for information on how complex series are
represented in MacAnova.
```

9.21 multitaper()

Usage:

```
multitaper(y, W, K [,degree:D,nfreq:Nfreq],deltat:delta_t,wts:wts]),
  REAL vector y, scalar W > 0, integer K >= 1, integer D, integer Nfreq
  >= length(y), REAL scalar delta_t > 0, REAL vector wts with wts[i] > 0
```

Keywords: spectrum analysis, tapering, frequency domain

```
                                    Usage
multitaper(y, W, K, degree:D, deltat:delta_t, nfreq:Nfreq, wts:Wts)
computes multitaper spectrum estimates using K discrete prolate
spheroidal sequences (DPSS) as tapers, with total bandwidth
approximately 2*W cycles per unit time.

The keyword phrases are optional.  See below for defaults.

y is a REAL matrix with no MISSING values whose columns are discrete
parameter time series observed at equally spaced times delta_t units
apart.

delta_t > 0 is a scalar; see below for default.

REAL scalar W specifies the half bandwidth W; 0 < W <.5/delta_t is
required.

Integer K > 0 is the number of DPSS tapers used.  Sensible values for K
should satisfy K < 2*nrows(y)*W*delta_t, but this is not checked.  At a
frequency where the spectrum is smooth, the approximate equivalent
degrees of freedom EDF (see topic 'bandwidth') of an estimate is
2*sum(Wts)^2/sum(Wts^2).  For equal weights, EDF = 2*K.

Wts is a REAL vector of length K with Wts[i] > 0.  Default is Wts =
rep(1/K, K), that is, equal weights.

The estimated spectrum is the weighted average, using weights in Wts,
of K periodograms of detrended columns of y, tapered using DPSS tapers.

                                  Reference
See Spectral Analysis for Physical Applications by D. B. Percival and
```

```
A. T. Walden (Cambridge Univ. Press, 1993) for details on multitaper
spectrum estimation.

                              Keyword deltat
Keyword phrase deltat:delta_t specifies the assumed interval between
observations.  It affects only the interpretation of W which is
interpreted as cycles per unit time.  Because of this, W must satisfy 0
< W < .5/delta_t.

For example, with hourly data and W in units of cycles per day, you
would use deltat:1/24 and W must satisfy 0 < W < 12.  See below for the
default value of delta_t.

                              Keyword degree
Keyword phrase degree:D specifies that the columns of y are detrended
using a polynomial of degree D fit by least squares.  When D < 0 no
detrending is done.  When D = 0 column means are subtracted.

                           Number of frequencies
When nfreq:Nfreq is an argument, estimated spectra will be computed at
Nfreq frequencies.  It is an error if Nfreq < nrows(y) or if Nfreq has
a prime factor > 29.

When nfreq:Nfreq is not an argument and S is a positive integer
variable, Nfreq = S.  It is an error if S has a prime factor > 29.
When such an S does not exist, Nfreq = smallest integer >= 2*nrows(y)
with no prime factor > 29, that is Nfreq = goodfactors(@*nrows(y))

                           Defaults for keywords
The default values of omitted keywords are as follows:
  Keyword    Default value
  degree     D = 0 (subtract only the mean).
  deltat     delta_t = variable DELTAT if it exists; otherwise 1
  nfreq      Nfreq = variable S if it exists or the smallest integer >=
             2*nrows(y) otherwise; it is an error if S has a prime
             factor > 29.
  wts        Wts = rep(1/K,K)

                            Other macros used
multitaper() uses macros dpss() and detrend().  If they are not defined,
an attempt is made to read them from tser.mac using getmacros.
```

9.22 spectrum()

Usage:

```
spectrum(y, len [,reps]), y a REAL matrix, and len > 0 and reps > 0
  integers
```

Keywords: spectrum analysis, frequency domain

```
                                  Usage
```

```
Sy <- spectrum(y, len) computes a smoothed periodogram estimate of the
spectrum of each column of REAL vector or matrix y.

Smoothing is accomplished by 4 successive convolutions of a "boxcar"
smoother of length len.  That is, the smoothing weights are
rep(1/len,len).  The total number of frequencies involved in each
spectrum estimate is 4*len - 3.  When len = 1 no smoothing is done.

spectrum(y, len, reps) does the same, except that reps convolutions
with a boxcar are used.  If reps is odd, then len must also be odd.

spectrum(y, 1) computes the unsmoothed periodogram.

The column means are subtracted before computing the periodograms, but
no other detrending or tapering is done.

See topic 'bandwidth' for information on the approximate bandwidth and
equivalent degrees of freedom associated with these estimates.

                          Number of frequencies
The spectrum is computed at the Nfreq Fourier frequencies 0, 1/Nfreq,
2/Nfreq, ..., (Nfreq-1)/Nfreq cycles per unit time, were Nfreq is
determined as follows:

  When variable S is defined and is a positive integer, Nfreq = S.  It
  is an error if S has a prime factor > 29.

  Otherwise, Nfreq = smallest integer >= 2*nrows(y) which has no prime
  factors > 29, that is Nfreq = goodfactors(2*nrows(y)).

                       Comparison with compfa()
Most of the time, macro compfa() is a better choice than spectrum() for
estimating spectra.  compfa() tapers the series and allows polynomial
detrending.  Moreover, you specify the amount of smoothing to use by
either a bandwidth or an EDF (equivalent degrees of freedom).  See
compfa() for details.

Other, quite different, choices for estimating spectra are macros
arspectrum() and burg().

                            Cross reference
See also arspectrum(), burg(), crsspectrum(), compfa(), compza().
```

9.23 testnfreq()

Usage:

```
testnfreq(nfreq), nfreq a vector of positive integers
```

Keywords: fourier transforms

```
                                  Usage
```

```
testnfreq(nfreq) returns a LOGICAL vector the same length as nfreq, a
vector of positive integers.  Element j of the result is True if and
only if nfreq[j] has no prime factors > 29.

testnfreq() is useful in macros which use one of the fast Fourier
transform functions, rft(), dft() and hft(), since these operate only on
vectors or matrices x for which nrows(x) has no prime factors > 29.  It
allows you to test whether this condition is true.

The use of testnfreq() is deprecated since goodfactors() can be used in
an equivalent test.  When N is a scalar, goodfactors(N) == N is True if
and only if testnfreq(N) is true.  Moreover, the value of goodfactors(N)
is the next integer >= N with no prime factors > 29.

                                  Example
  if (!testnfreq(nrows(y))){
    error("nrows(y) has prime factor > 29")
  }else{
    yft <- rft(y)
  }

                              Cross reference
See also topics goodfactors(), primefactors(), rft(), hft(), cft(),
'fourier'
```

9.24 tsplot()

Usage:

```
   tsplot(y, times [,symbols:Symb] [,lines:F] [,impulse:T]
     [,timeunit:Unit] [,graphics keywords]), y a REAL matrix, times a REAL
     vector with length(time) = nrows(y), Symb a REAL or CHARACTER scalar,
     vector or matrix, Unit a CHARACTER scalar
   tsplot(y [,start [,deltat]] [,symbols:Symb] [,lines:F] [,impulse:T]
     [,timeunit:Unit] [,graphics keywords]), Start and delta_t > 0 REAL
     scalars
```

Keywords: plotting, time domain

```
                                    Usage
tsplot(Y, Times) does a line plot of the columns of REAL matrix y vs
Times, a REAL vector whose non MISSING values must be in increasing
order.

tsplot(Y, Start, Delta_t) does the same using Times = Start +
Delta_t*run(0,nrows(Y)-1).  That is, the plotting times are equally
spaced starting with Start and incrementing by Delta_t.

tsplot(Y, Start) does the same using a default value for Delta_t.

tsplot(Y) does the same using default values for Start and Delta_t.
```

```
Defaults for Start and Delta_t are taken from variables START and
DELTAT, if they exist and are non-missing REAL scalars with DELTAT > 0.
When they do not exist the defaults for Start and Delta_t are 0 and 1,
respectively.

You can provide a time unit to be used in constucting a graph title and
x-axis label by setting variable TIMEUNIT or by using keyword
'timeunit'.  See below.

You can use the usual graphic keywords, including 'title', 'xlab',
'ylab', 'xmin', 'xmax', 'ymin', 'ymax', 'linetype' and 'impulse'.

With 'impulse:T', no connecting lines are drawn unless 'lines:T' is also
an argument.

Without 'symbols:Symb', no symbols are plotted unless 'lines:F' is an
argument and 'impulse:T' is not an argument.  See below.

                            Keyword timeunit
tsplot(Y, Times, timeunit:Unit) and tsplot(Y, Start, Delta_t,
timeunit:Unit)  do the same except that Unit is used in constructing
the default title and x-axis label.  Unit must be a CHARACTER scalar
such as "year" that is different from "".

Unit should be specified consistantly with Delta_t.  For example, with
monthly data and Delta_t = 1/12, Unit should be "year", while with
delta_t = 1, Unit should be "month".

Without keyword 'timeunit', the default title and x-axis label will be
constructed from the value of CHARACTER scalar TIMEUNIT, if it exists
and differs from "".

                            Keyword symbols
tsplot(Y, Times, symbols:Symb) and tsplot(Y [,Start ,[Delta_t]],
symbols:Symb) does the same, except that plotting symbols are taken from
REAL or CHARACTER scalar, vector or matrix Symb.  When Symb is a scalar,
it will be used for every point.  When Symb is a vector of length
ncols(Y), Symb[j] will be the plotting symbol for column j.  See
chplot() for further details.

'symbols:?'  is a special case.  It specifies that plotting symbols will
be 1, 2, ..., nrows(y) when Y is a vector and 1, 2, ..., ncols(y) for
each column of Y when ncols(Y) > 1.

                            Keyword impulse
tsplot(Y, Times, impulse:T, ...) and tsplot(Y [,Start ,[Delta_t]],
impulse:T, ...) do the same, except an "impulse" plot is drawn rather
than a line plot.  If you want both, also include 'lines:T' as an
argument.

                             Keyword lines
tsplot(Y, Times, lines:F, ...) and tsplot(Y [,Start ,[Delta_t]],
```

```
lines:F, ...) do the same, except that no lines or impulses are drawn.
If symbols are not supplied, the symbols are the default symbols drawn
by plot().

                            Keyword graphics
All the usual graphic keywords can be used, including 'impulse',
'lines', 'title', 'xlab', 'ylab', 'xaxis', 'yaxis', 'xmin', 'xmax',
'ymin', 'add', 'linetype', etc.

In particular, 'impulse:T' draws an impulse plot without lines unless
'lines:T'is also an argument.  See regular help topic 'graph_keys' for
details.

                                 Examples
Suppose the columns of x contain 10 years of monthly data starting in
January, 1948.  Then
  Cmd> tsplot(x, 1948, 1/12, symbols:"\1",xlab:"Year")

will make a plot of the columns of x against time in years, using a
small triangle as plotting symbol.  If DELTAT has value 1/12, argument 3
can be omitted.  If also variable START is 1948, argument 2 can be
omitted.

 Cmd> tsplot(x,1948+run(0,119)/12, symbols:"\1",xlab:"Year")

makes the same plot, ignoring DELTAT and START.

Suppose rhohat contains autocorrelation functions for the columns of x,
starting with lag 1 month, perhaps computed as rhohat <- autocor(x,60).
Then
  Cmd> tsplot(rhohat,1/12,1/12,impulse:T,ymin:-1,ymax:1,\
       xlab:"Lag (Years)", ylab:"Autocorrelation",\
       title:"Autocorrelation functions for x")

makes an impulse plot of the autocorrelation functions.

If you wanted the lags in months, use tsplot(rhohat,1,1,...,xlab:"Lag
(months)",...)  After DELTAT <- 1/12, simply tsplot(rhohat,DELTAT,
impulse:T, ...) would produce the same plot.

                             Cross reference
See also ffplot(), autocor().
```

Chapter 10

Graphical User Interface Help File

This Chapter contains help for the graphical user interface macros that are distributed with MacAnova in the file Gui.mac.txt. The material here is a reformatting of file Gui.hlp.txt.

10.1 alert()

Usage:

```
alert(mess) displays a message box containing mess
```

Keywords: dialogs

```
alter(mess) opens a dialog box displaying the message in the
character scalar mess.
```

10.2 doguihelp()

Usage:

```
doguihelp(filename) display html file filename in a simple html viewer
```

Keywords:

```
doguihelp(filename) displays the html file named in the CHARACTER
scalar filename in a simple browser.  filename should be a complete
path to the html file (see findfile()).  See also 'guihelp()' and
'help()'.
```

10.3 getdirname()

Usage:

```
getdirname() finds a directory
```

Keywords:

```
getdirname() brings up a dialog box to allow the user to select
a directory.  The full path to the directory is returned as a
CHARACTER scalar.
```

10.4 getmenubar()

Usage:

```
   getmenubar() returns a character scalar containing the menu resource
```

Keywords: xml

```
getmenubar() returns a character scalar containing the current
menu resource.
```

10.5 guiabout()

Usage:

```
   guiabout() displays the About Macanova dialog
```

Keywords:

```
guiabout() displays the About Macanova dialog.
```

10.6 guianova()

Usage:

```
   guianova() does basic anova via dialogs
```

Keywords:

```
guianova() does basic anova via dialogs
```

10.7 guiboxplot()

Usage:

```
   guiboxplot() does boxplots via dialogs
```

Keywords:

guiboxplot brings up a dialog box to collect information that will be used to construct a boxplot command. The dialog is a "tabbed" dialog, meaning that the information is collected on multiple panels that the user accesses via tabs.

The "Basic" tab collects the standard information for a basic boxplot without any bells and whistles. At the bottom of the panel, you choose the variable or variables to appear in the boxplot. If you choose multiple vector variables, each variable will determine one box. If you choose a matix variable, each column will determine one box. At the top of the panel, you can choose a "split-by" variable. If you have chosen a single vector variable at the bottom of the panel, you can choose a "split-by" variable to divide the single variable into groups. This split-by variable must have the same number of elements as the plotted variable, and a separate box will be constructed for each unique value of the split-by variable, with elements of the response variable divided according to their corresonding split value. See split(). You must type the name of the split variable (or an expression that computes an appropriate split variable) into the dialog element.

The remainder of the elements on the Basic tab control how the boxplots look. You can choose vertical (default) or horizontal boxes. You can choose to show outliers (default) or to simply have the whiskers extend to the extremes. If you show the outliers, you can control the symbols used to display them. You may also enter variables or expressions to control the location and width of the boxes. The location expression should evaluate to a real vector with length equal to the number of boxes. The width expression can be a real scalar or a vector with length equal to the number of boxes.

The "Appearance" tab collects information that affects the overall appearance of the plot. First, you can choose that the plot appear in a new window (default), or you can choose the number of the graph window where you would like it to appear. A window number of 0 indicates the most recently used graph window. Next, you can set the width and height of the plot. On the screen, these are in units of pixels. When printing using PostScript, these are in units of points (approximately 1/72 of an inch).

The second major set of choices are for labels. You can add a title and/or labels for the vertical and horizontal axes.

Finally, you can set where the border box and axis ticks will be drawn. By default, ticks and borders are drawn on all four sides.

The "Axes" tab allows you to control the appearance of the axes. First, you can choose to have a logarithmic scale by clicking the check box. Next, you may specify your own minimum and maximum values in each direction. Third, you may decide whether the x=0 or y=0 lines are drawn on the plot. Finally, you may set the appearance of the

```
ticks and labels.  Tick locations should be either a variable name
or an expression that evaluates to a vector of real values.  If
this is NULL, no ticks will be drawn.  Tick labels should be character
vectors with the same number of elements as the tick locations.
Finally, tick lengths should be real scalars >= -1.  Values less than
0 are outside the frame; values greater than zero are inside the
frame.  Values greater than 2 draw a grid all the way across the
plot.  The default value is -.5.

If you know the MacAnova commands, you may type in your options
directly on the "Direct Options" tab.
```

10.8 guifilepath()

Usage:

```
   guifilepath(type:charscalar) where charscalar is one of
   "setdatafile", "addmacrofile", "addhelpfile", "addpath"
```

Keywords:

```
guifilepath() is used to set various files, file lists,
and path lists.  It brings up dialogs to solicit file
information, which it then forms into a MacAnova command.
Options are
  type:"setdatafile"   does DATAFILE <- "..."
  type:"addmacrofile"  does addmacrofile("...")
  type:"addhelpfile"   does addhelpfile("...")
  type:"adddatapath"   does adddatapath("...")
where the ... is extracted from the dialogs
```

10.9 guihelp()

Usage:

```
   guihelp(topic) displays html help on topic
```

Keywords:

```
guihelp(topic) displays the MacAnova html help associated with topic.
Standard MacAnova help topics have html versions in files in the
SharedSupport/docs/html directory.  They have names of the form
prefix_topic.htm; for example, the html help on run is in base_run.htm,
and the html help on reml is in design_reml.htm .  guihelp() runs
through the various prefixes in an attempt to match an html help file,
and then calls doguihelp() when a match is found.  See also
'doguihelp()' and 'help()'.
```

10.10 guihist()

Usage:

```
guihist()
```

Keywords:

guihist brings up a dialog box to collect information that will be used to construct a histogram command. The dialog is a "tabbed" dialog, meaning that the information is collected on multiple panels that the user accesses via tabs.

The "Basic" tab collects the standard information for a basic histogram without any bells and whistles. At the bottom of the panel, you choose the variable to appear in the histogram.

The elements at the top of the Basic tab control how the histogram looks. On the left, you can choose between density (default) or frequency or relative frequency histograms. On the right, you set the bins. You may let MacAnova choose the bins, or you can choose the number of bins and let MacAnova choose their locations. The two remaining options allow you to specify the bin locations. The anchor/width specification will produce adjacent bins with your chosen width, with one bin edge placed at the anchor point. Both the anchor and the width must be numeric values. The final choice is to enter an expression that will evaluate to a vector of bin edges. It is usually an error to have data outside the bins, but you may choose to let that happen by checking the "Data outside bins" box. Finally, you choose the endpoint convention. By default, the right hand endpoint is in the interval, but you may choose to have the left hand endpoint in the interval by choosing that option.

The "Appearance" tab collects information that affects the overall appearance of the plot. First, you can choose that the plot appear in a new window (default), or you can choose the number of the graph window where you would like it to appear. A window number of 0 indicates the most recently used graph window. Next, you can set the width and height of the plot. On the screen, these are in units of pixels. When printing using PostScript, these are in units of points (approximately 1/72 of an inch).

The second major set of choices are for labels. You can add a title and/or labels for the vertical and horizontal axes.

Finally, you can set where the border box and axis ticks will be drawn. By default, ticks and borders are drawn on all four sides.

The "Axes" tab allows you to control the appearance of the axes. First, you can choose to have a logarithmic scale by clicking the check box. Next, you may specify your own minimum and maximum values in each direction. Third, you may decide whether the x=0 or y=0 lines

```
are drawn on the plot.  Finally, you may set the appearance of the
ticks and labels.  Tick locations should be either a variable name
or an expression that evaluates to a vector of real values.  If
this is NULL, no ticks will be drawn.  Tick labels should be character
vectors with the same number of elements as the tick locations.
Finally, tick lengths should be real scalars >= -1.  Values less than
0 are outside the frame; values greater than zero are inside the
frame.  Values greater than 2 draw a grid all the way across the
plot.  The default value is -.5.

If know the MacAnova commands, you may type in your options
directly on the "Direct Options" tab.
```

10.11 guilistxml()

Usage:

```
guilistxml()
```

Keywords:

```
This is a utility routine, not called by users
```

10.12 guilistctrl()

Usage:

```
guilistctrl()
```

Keywords:

```
This is a utility routine, not called by users.  It produces
the xml for a choose variable control that can be embedded into
a dialog.  See guilistdlg() for a description of the arguments.
```

10.13 guilistdlg()

Usage:

```
guilistdlg(many, many keyword arguments)
```

Keywords:

```
This routine brings up a dialog box allowing you to choose variables,
and returns a CHARACTER vector of the chosen names.  It takes
keyword arguments of several types: usexxxx:T, reqxxxx:T, and
```

```
attrxxxx:T|F (for example, attrlogic:T or reqreal:T).  Without any
arguments, all variables are included in the dialog.  If an reqxxx:T
argument is present, any variable must match the xxxx to be listed in
the dialog.  If one or more usexxxx:T arguments is present, then a
variable must match at least one of the xxxx descriptors to be
included in the dialog.  If an attrxxx: keyword is used, then a
checkbox will be added to the dialog to optionally allow variables
that match the xxx to be show.  The T or F of the attrxxx keyword
determines whether the checkbox in initially checked or unchecked.
Note: if any attrxxx:T arguments are present, then a variable must
match at least one of them to be listed.

There are two additional reqxxx keywords.  reqnrows:k means that a
variable must have leading dimension k; reqndims:k means that a
variable must have k dimensions.

Keyword maxselected:k sets that at most k variables can be selected.
0 is the default, and it indicates no limit.

usonly:vec says to use only the names in the character vector vec.
omit:vec says to omit any names in the character vector vec.

matchrows:T means that once a variable has been selected, only
variables that match the number of rows will be show.  This differs
from reqnrows:k in that matchrows:T allows different numbers of
rows to be shown before the first variable is selected.

Possible xxxx attributes are:
char         CHARACTER variable
graph        GRAPH variable
logic        LOGIC variable
macro        MACRO variable
real         REAL variable

scalr        Scalar variable
vect         Vector variable (a la isvector)
matrx        Matrix variable (a la ismatrix)
1d           Exactly one dimensional
2d           Exactly two dimensional
array        Array variable
struc        Structure variable

lockd        Locked variable
ulock        Unlocked variable

factr        Factor variable
vart         Variate (1 dimensional, real, not factor) variable
binary       0/1 variable
system       Name is all capitals, and either longer than
                        1 letter or not "E"

nonsys       A non-system variable
nsvart       A non-system variate
```

```
nsfact       A non-system factor
ns1d         A non-system 1d variable
ns2d         A non-system 2d variable
nsstrc       A non-system structure
```

10.14 guintrctplt()

Usage:

```
guintrctplt() does interaction plots via dialogs
```

Keywords:

```
guintrctplt brings up a dialog box to collect information that
will be used to construct an interaction plot command.  The
dialog is a "tabbed" dialog, meaning that the information is
collected on multiple panels that the user accesses via tabs.

Interaction plots graphically display a vector/matrix/array of
real numbers.  The coordinates of the first dimension are
indicated by the horizontal plotting position.  The levels of
any other dimensions are indicated by the plotting symbol; for
example, 2.4 indicates level 2 of the second factor and level 4
of the third factor.  Points with the same plotting symbol are
joined by lines.

The "Basic" tab determines the vector/matrix/array to be plotted.
First, you may directly select a matrix or array of plotting
positions by selecting a matrix of array in the variable selection
control at the bottom of the tab.  In this form, you only select
a single matrix or array.  Second, if there is an active model,
you may choose to plot least squares means from the active model.
To do this, you check the "use LS means" box in the "Control"
subdialog and select the desired factors from the model in the
variable selection control.  (Note: this form makes use of glmtable()
internally and thus will not work for balanced designs.  Use unbal:T
in the anova() command to enable the use of LS means in interaction
plots for balanced data.)  Finally, you may indicate the vector/
matrix/array to be plotted as the tabular means of a response
variable split according to one or more factor variables.  In this
form, you first select the response variable, and then select one
or more splitting variables.

You may optionally choose to plot error bars around each mean,
simply by checking the show error bars button in the "Control"
subdialog.  By default, the bars are plus or minus two SE, but you
may adjust that in the Control subdialog.  For LS means, standard
errors are taken directly from the model.  For the matrix/array
form, you must enter the name of a matrix of standard errors in
```

the "Options" subdialog. The tabular data form will compute SEs for the within-cell variances. Note: an SE of 0 will be used for cells with a single observation. Optionally, checking the "Pooled estimate of error" box in the Options subdialog will pool variance information from all cells into a single common estimate of variance, which will then be used to compute cell standard errors. Finally, you may simply specify a common error variance directly in the Options subdialog.

The "Appearance" tab collects information that affects the overall appearance of the plot. First, you can choose that the plot appear in a new window (default), or you can choose the number of the graph window where you would like it to appear. A window number of 0 indicates the most recently used graph window. Next, you can set the width and height of the plot. On the screen, these are in units of pixels. When printing using PostScript, these are in units of points (approximately 1/72 of an inch).

The second major set of choices are for labels. You can add a title and/or labels for the vertical and horizontal axes.

Finally, you can set where the border box and axis ticks will be drawn. By default, ticks and borders are drawn on all four sides.

The "Axes" tab allows you to control the appearance of the axes. First, you can choose to have a logarithmic scale by clicking the check box. Next, you may specify your own minimum and maximum values in each direction. Third, you may decide whether the x=0 or y=0 lines are drawn on the plot. Finally, you may set the appearance of the ticks and labels. Tick locations should be either a variable name or an expression that evaluates to a vector of real values. If this is NULL, no ticks will be drawn. Tick labels should be character vectors with the same number of elements as the tick locations. Finally, tick lengths should be real scalars >= -1. Values less than 0 are outside the frame; values greater than zero are inside the frame. Values greater than 2 draw a grid all the way across the plot. The default value is -.5.

If you know the MacAnova commands, you may type in your options directly on the "Direct Options" tab.

10.15 guipatterned()

Usage:

```
guipatterned()
```

Keywords:

guipatterned() will solicit information for building factors,

```
maximum level, the number of times each should be repeated
consecutively, and the number of times the whole pattern is then
repeated. You may optionally save as factor or nonfactor.
```

10.16 guiplotresid()

Usage:

```
guiplotresid() does residual plots via dialogs
```

Keywords:

```
guiplotresid brings up a dialog box to collect information that
will be used to construct a plotresids command for plotting
residuals.  The dialog is a "tabbed" dialog, meaning that the
information is collected on multiple panels that the user accesses
via tabs.

The "Basic" tab allows you to determine which plots you want,
which kind of residuals to use, and what plotting character to use.
On the left, you may select one or more of the plot types: residuals
versus fitted values, or residuals versus normal scores, or
residuals versus case numbers.  On the right, you may choose which
type of residuals: raw residuals, scaled residuals, standardized
residuals, or studentized residuals.  You may also choose the size
and style of the plotting character.

Scaled residuals are raw residuals divided by the root MSE.
Standardized residuals are residuals divided by an estimate of their
standard error (ie, this takes the HII values into account).
Studentized residuals are outlier-t residuals.  In all cases,
scaling involves the final error term of the model.

The "Appearance" tab collects information that affects the overall
appearance of the plot.  First, you can choose that the plot appear
in a new window (default), or you can choose the number of the
graph window where you would like it to appear. A window number of
0 indicates the most recently used graph window. Next, you can set
the width and height of the plot.  On the screen, these are in
units of pixels.  When printing using PostScript, these are in
units of points (approximately 1/72 of an inch).

The second major set of choices are for labels.  You can add a title
and/or labels for the vertical and horizontal axes.

Finally, you can set where the border box and axis ticks will be
drawn.  By default, ticks and borders are drawn on all four sides.

The "Axes" tab allows you to control the appearance of the axes.
First, you can choose to have a logarithmic scale by clicking the
check box.  Next, you may specify your own minimum and maximum values
```

```
in each direction. Third, you may decide whether the x=0 or y=0 lines
are drawn on the plot.  Finally, you may set the appearance of the
ticks and labels.  Tick locations should be either a variable name
or an expression that evaluates to a vector of real values.  If
this is NULL, no ticks will be drawn.  Tick labels should be character
vectors with the same number of elements as the tick locations.
Finally, tick lengths should be real scalars >= -1.  Values less than
0 are outside the frame; values greater than zero are inside the
frame.  Values greater than 2 draw a grid all the way across the
plot.  The default value is -.5.

If you know the MacAnova commands, you may type in your options
directly on the "Direct Options" tab.
```

10.17 guirandom()

Usage:

```
guirandom(dist:disttype)
```

Keywords:

```
guirandom() is a dialog based interface to the most common
ways to generate random data into MacAnova.  The possible disttypes
(as character scalars) are:
    "uniform","normal","binomial","poisson","F","beta",
    "gamma","chisq","student"
```

10.18 guireadfile()

Usage:

```
guireadfile(type:readtype,useclip:TF)
```

Keywords:

```
guireadfile() is a dialog based interface to the most common
ways to read data into MacAnova.  If useclip is T, then guireadfile()
will read from the clipboard; otherwise it will read from a file.
The possible readtypes (as character scalars) are:
    "matread"             read in matread format
    "matread/datafile"      read from DATAFILE in matread format
    "vecread"               read data as a vector
    "vecread/matrix"        read data but store as matrix
    "readdata/labelled"     read labelled columns
    "readdata/unlabelled"   read unlabelled columns
```

10.19 guirsample()

Usage:

```
guirsample() to subsample a variable through a dialog
```

Keywords:

```
guitypein() brings up a dialog into which you can enter information
to subsample from a variable.
```

10.20 guitypein()

Usage:

```
guitypein() to type in data through a dialog
```

Keywords:

```
guitypein() brings up a dialog into which you can type data,
and then have it read in.
```

10.21 setmenubar()

Usage:

```
setmenubar(res,menuName) replaces the current menubar with that
described in the xml resource res with name menuName; both res and
menuName are CHARACTER scalars
```

Keywords: xml

```
All menus in Carapace versions of MacAnova, including the default
menus, are set using XML resources.  These resources are based on the
XRC system in wxWidgets, with some MacAnova-specific extensions.
The resource variable is a CHARACTER scalar in XML format:
<?xml version="1.0"?>
  <resource>
    ...
  </resource>

Of course, all the action is in the ... where the menus are actually
specified.  The resource is a collection of objects.  Each object
has a type given by its class, and an identifier given by its name.
Within the object, you can have parameters that describe or modify
the object, and other objects that are called children of the containing
object.

To set up a menu bar, the resource contains one object of class
```

```
wxMenuBar.  The name for that menubar is the menuName used in the
setmenubar() call.  For example, setmenubar(resource,"sampleMenu")
based on this excerpted resource:
<?xml version="1.0"?>
  <resource>
    <object class="wxMenuBar" name="sampleMenu">
      ...
    </object>
  </resource>

The children of a menu bar are objects of class cpcMenu; these are the
menus seen on the menu bar.  The children of a menu are items (objects
of class cpcMenuItem), separators (objects of class separator), and/or
submenus (more objects of class cpcMenu).  Here is an example, with
comments and explanations interspersed in the XML.

      <object class="cpcMenu" name="File_menu">
        <label>File</label>
Start a menu. The name (File_menu) should be unique, but is otherwise
not used.  The label determines how the menu will be shown on the
menu bar; here we have the File menu.
          <object class="cpcMenuItem" name="CPC_FILE_OPEN">
            <label>Open\tCtrl+O</label>
            <accel>Ctrl+O</accel>
            <help>Open a text file</help>
          </object>
Here we have a menu item with name CPC_FILE_OPEN.  Many names that
begin CPC_ correspond to predetermined actions; here, CPC_FILE_OPEN
opens a text file in a new output window.  A list of these known
names is given below.  The label parameter determines how the item
will appear on the menu.  The accel parameter determines a keyboard
combination that is equivalent to selecting the menu item, here,
control plus O.  Finally, the help parameter is a message displayed
in the status bar at the bottom of the frame.
          <object class="separator"/>
This is a separator to produce a gap in the menu.
          <object class="cpcMenuItem" name="SaveWorkspace">
            <label>Save workspace\tCtrl+K</label>
            <accel>Ctrl+K</accel>
            <help>Save the workspace in a file</help>
            <action>save()</action>
          </object>
Here is a menu item with an action.  When a menu item with
an action is selected, the action value is sent to MacAnova as a
command, just as if you had typed it at the command line.  Please
note, if you add an action to one of the standard CPC_... named
items, the action will be ignored.
          <object class="cpcMenu" name="CPC_WINDOWS_TEXTWINDOW">
            <label>Output windows</label>
            <help>Select output window</help>
            <object class="cpcMenuItem" name="fake window">
              <label>fake</label>
            </object>
```

```
          </object>
Here is object that is another menu.  In this case, the menu named
CPC_WINDOWS_TEXTWINDOW has only a single fake item in the resource.
The menu with this name is updated automatically by Carapace to
reflect the command windows present.
          <object class="cpcMenuItem" name="CPC_HELP_HELP">\
            <label>$Help\tCtrl+L</label>\
            <accel>Ctrl+L</accel>\
            <selectionlabel>$Help on selection</selectionlabel>\
            <help>Help on MacAnova</help>\
            <action>guihelp()</action>\
            <hidecommand>1</hidecommand>\
            <selectionaction>guihelp(%s)</selectionaction>\
          </object>\
        </object>
You can make menus change slightly depending on whether or not anything
in the window is selected.  If there is a selectionlabel parameter,
that label will be used whenever something in the window is selected.
Similarly, the selectionaction will be used instead of the action
whenever something is selected.  In this case, the selected text will
replace the %s in the selectionaction parameter.  One additional
twist here is hidecommand.  Ordinarily, any action or selectionaction
command is printed in the output pane.  If hidecommand is 1, the
command will not be printed.  It is not shown here, but there is also
a hideoutput parameter. If hideoutput is 1, then any printing that
the command would do is suppressed.

Here are the standard names and their corresponding actions:
 CPC_FILE_OPEN            Open a file in an output window
 CPC_FILE_SAVEWINDOW      Save the contents of the output pane
 CPC_FILE_SAVEWINDOWAS    Save the contents, but change the name
 CPC_FILE_PAGESETUP       Set up for printing
 CPC_FILE_PRINTSELECTION  Print
 CPC_FILE_INTERRUPT       Interrupt execution
 CPC_FILE_QUIT            Quit
 CPC_FILE_FASTQUIT        No fooling around, just quit now

 CPC_EDIT_UNDO            Undo last edit/typing
 CPC_EDIT_CUT             Cut selection
 CPC_EDIT_COPY            Copy selection
 CPC_EDIT_PASTE           Paste selection
 CPC_EDIT_COPYTOEND       Copy selection to command line
 CPC_EDIT_EXECUTE         Execute the command line
 CPC_EDIT_UPHISTORY       Go back in command history
 CPC_EDIT_DOWNHISTORY     Go forward in command history

 CPC_WINDOWS_HIDE         Hide this window
 CPC_WINDOWS_CLOSE        Close this window
 CPC_WINDOWS_FASTCLOSE    Close this window, no chance to save
 CPC_WINDOWS_NEWWINDOW    Open a new command window
 CPC_WINDOWS_GRAPH        Nothing happens, this is a submenu
 CPC_WINDOWS_TEXTWFONT    Select command window font
 CPC_WINDOWS_GOTOTOP      Scroll to top of output pane
```

```
CPC_WINDOWS_GOTOEND      Scroll to bottom of output frame
CPC_WINDOWS_GOTOCOMMANDPOINT  Move focus to end of command pane
```

Chapter 11

User Function Help File

This Chapter contains help for users interested in producing user functions (compiled C or FORTRAN code that is loaded into MacAnova and executed). The material here is a reformatting of file Userfun.hlp.txt A suggested order for reading this material is

1. loadUser
2. User
3. user fun
4. arginfo fun
5. callback fun
6. c macros
7. compile dos compile mac compile unix compile win
8. type codes

11.1 arginfo_fun

Usage:

```
Type userfunhelp(user_fun) for information on the structure of user
  functions.
Type userfunhelp(callback_fun) for information on the structure of user
  functions making "call backs" to MacAnova.
Type userfunhelp(arginfo_fun) for information on how to enable automatic
  checking of arguments to a user function.
```

Keywords: user functions, coding, sample source

```
This topic presumes familiarity with topic User() and user_fun.  It
provides a brief introduction to the form of an arginfo function, that
is an externally compiled function that can be called by MacAnova to
```

obtain information about the arguments expected by a user function. If available, an arginfo function operates transparently to the user of MacAnova. Because of the inherent dependence on the computer and operating system, there are many details that are not covered here. Additional details may be found in topics compile_dos, compile_mac, compile_unx and compile_win.

This topic presumes familiarity with topics user_fun and callback_fun.

Arginfo functions are not currently possible when compiling for the protected mode DOS version (DJGPP).

Since we have no experience with writing arginfo functions in Fortran, no Fortran related information is provided here.

In the following, 'handle' is used in the Macintosh OS sense, as a pointer to a pointer.

To compile an arginfo function associated with user function foo, say, you need to include a function named 'arginfo_foo' in the source for foo. arginfo_foo should have no arguments and should return a pointer to a vector of long integers, that is it should be declared as

```
  long * arginfo_foo(void)
```

When compiling for Windows using Borland C/C++ 4.5, the declaration should be

```
  long * _export arginfo_foo(void)
```

The ending of the name of the arginfo function (here 'foo') must match the name of the user function.

arginfo_foo should return a pointer to a vector arginfo of Nargs + 2 long integers, where Nargs is the number of arguments expected by foo, excluding the list, if any, of call back functions (see callback_fun).

The first element of vector arginfo (arginfo[0]) must be Nargs >= 1.

The second element of vector arginfo (arginfo[1]) is composed of bit constants that specify various properties of the function (whether it makes call backs, whether it expects pointers or handles, whether its arguments should be data or symbols, and whether a required 68881 co-processor is absent (Macintosh only). Symbolic names for these are defined in header file dynload.h which is automatically included by header file Userfun.h.

Name of bit	Meaning
DOESCALLBACK	Call backs to MacAnova functions will be made
NOCALLBACK	No call backs to MacAnova functions will be made
USESPOINTERS	Arguments should be pointers
USESHANDLES	Arguments should be handles
POINTERUSE	Same as USESHANDLES on Macintosh and same as USESPOINTERS on other systems
SYMBOLARGS	All arguments (except call back function list) are pointers or handles to Symbols
NOSYMBOLARGS	All arguments (except call back function list) are

```
                    pointers or handles to data
    COPROCESSOROK    Co-processor not needed or, if needed, is available
    COPROCESSORERROR A co-processor is needed but not available
```

For example, for a function with default pointer/handle usage that makes call backs and does not expect Symbol arguments, arginfo[1] should be DOESCALLBACK | POINTERUSE | NOSYMBOLARGS. When compiling for a Macintosh, this is equivalent to DOESCALLBACK | USERSHANDLES | NOSYMBOLARGS; when compiling for other computers it is equivalent to DOESCALLBACK | USESPOINTERS | NOSYMBOLARGS

Strictly speaking NOCALLBACK and NOSYMBOLARGS are not needed since they evaluate to 0, but their use can make for clearer code.

The remaining Nargs elements (arginfo[2], arginfo[3], ..., arginfo[Nargs+1]) of the vector are integers that specify the MacAnova types of the user function arguments, using symbolic constants defined in Userfun.h. Typical constants are REALMATRIX, CHARSCALAR, LOGICSCALAR, INTVECTOR, POSITIVEREALVECTOR, NONNEGATIVEINT, LONGVECTOR and SYMHVALUE. The qualifier INT means REAL with integer values; the qualifer LONG means actual long integers as produced by asLong(). See topic type_codes for a complete list of permissible constants.

In writing a function for a 68K Macintosh when compiling using Metrowerks CodeWarrior, to ensure correct compilation, all declarations of call back and arginfo functions must be bracketed by

```
  #pragma mpwc on
  ...
  #pragma mpwc off
```

Here is an example of a function to provide argument information for fooeval() listed under topic callback_fun and executed from MacAnova by, say,

```
  Cmd> User("fooeval", "sqrt(PI/2)")
```

Non-Macintosh version:

```
  #include "Userfun.h"

  static long Fooevalarginfo[] =
        {1, DOESCALLBACK | POINTERUSE | NOSYMBOLARGS, CHARSCALAR};

  long * arginfo_fooeval(void)
  {
      return(Fooevalarginfo);
  }
```

Macintosh version:

```
  #include "Userfun.h"

  #define info_main main
```

```
  static long Fooevalarginfo[] =
        {1, DOESCALLBACK | POINTERUSE | NOSYMBOLARGS, CHARSCALAR};

  #ifndef powerc
  #pragma mpwc on
  #endif
  long * info_main(void)
  {
      long         *arginfo;

      EnterCode();

      arginfo = Fooevalarginfo;
      /*add COPROCESSORERROR to arginfo[1] if appropriate*/
      CHECK68881(arginfo);

      ExitCode();
      return(arginfo);
  }
  #ifndef powerc
  #pragma mpwc off
  #endif

  #ifdef powerc
  RoutineDescriptor arginfo_fooeval =
     BUILD_ROUTINE_DESCRIPTOR(uppArgInfoEntryProcInfo, info_main);
  #endif /*powerc*/
```

When compiled for a 68K Macintosh, this must be compiled separately from fooeval. If the source is in the same file as source for fooeval, some form of conditional compilation should be used so that both arginfo_fooeval and fooeval don't both get compiled at once. The code resource produced should have name arginfo_fooeval and be included in the same resource file as fooeval.

When compiled for a Power PC Macintosh, arginfo_fooeval would normally be in the same source file as fooeval (C function main) and info_main would not be defined to be main. A single compilation would produce a resource file containing resource fooeval with entries fooeval and arginfo_fooeval. The actual entry points would be specified by RoutineDescriptors fooeval and fooeval_arginfo.

See topics compile_dos, compile_mac, compile_unix and compile_dos for information on compiling a user function on different types of computers.

See loadUser() and User() for information on how to load and execute a user function.

See topic user_fun for information on the structure of a user function not making call backs to MacAnova.

See topic callback_fun for information on the structure of a user

```
function making call backs to Macanova.

See topic c_macros and header file Userfun.h distributed with MacAnova
for C macros that are helpful in writing arginfo functions.
```

11.2 c_macros

Usage:

```
type userfunhelp(c_macros) for information on available C_macros for compiling
  user functions that may be compiled for more than one type of computer.
```

Keywords: user functions, coding, sample source

```
This topic presumes familiarity with topics User(), user_fun,
arginfo_fun and callback_fun.  It describes the use of the C macros in
header file Userfun.h in writing user functions in such a way that their
code may be compiled on a variety of computers with little or no change.
Some of the macros in Userfun.h are helpful even when writing a user
function to run on a single type of computer.  Among other things, use
of these macros ensures that all non-symbol arguments end up as pointers
and symbol arguments end up as handles.  They can also make it easier to
call back to MacAnova.  See callback_fun.

To make these macros available, the following should appear in your
source file
  #include "Userfun.h"
and both files Userfun.h and dynload.h (included by Userfun.h) should be
in the same directory as the file being compiled.

Userfun.h also includes macros for working directly with Macanova
symbols.  These are not discussed here.  However, example source
fooeval.c includes some example of their use.  See Userfun.h for more
information.

Userfun.h also defines constants for describing the type and shape of
user function arguments.  See topic type_codes for a complete list.

Here is a brief summary of the most important macros in Userfun.h.

Prefix each user and arginfo function name with EXPORTED:
  void EXPORTED foo(...)
or
  long * EXPORTED arginfo_fooeval(void)

If MACINTOSH is defined, the macros referencing arguments (THEARG,
THECOMMAND, THESYMBOL, CALLBACKFUN) normally assume the arguments are
handles; if MACINTOSH is not defined, they normally assume arguments are
pointers.

If, for some reason, you want to deviate from this convention, you can
override it by using one of the following
```

```
  #define POINTERARGS 1  /*arguments assumed to be pointers*/
or
  #define POINTERARGS 0  /*arguments assumed to be handles*/
```

If POINTERARGS is not defined, Userfun.h defines it to be 0 on a Macintosh or 1 otherwise.

Use DOUBLEARG(argx), CHARARG(argx), LONGARG(argx) and SYMBOLARG(argx) to declare arguments other than a list of call back functions. They expand to handles (double ** argx, char ** argx, long ** argx, Symbol ** argx) when POINTERARGS is 0 (Macintosh) and to pointers (double * argx, char * argx, long * argx, Symbol * argx) when POINTERARGS is 1 (everywhere else). If you do deviate and do not provide an arginfo function (see arginfo_fun), you will have to include either pointers:T (Macintosh) or pointers:F (otherwise) as an argument to User().

Use CALLBACKLIST(funlist) to declare the call back function list structure. It expands to MacAnovaCBSH funlist (a handle) when POINTERARGS is 0 and to MacAnovaCBSPtr funlist (a pointer) otherwise.

Use arg = THEARG(argx) to obtain a pointer to non-symbol argument. This expands to arg = *argx (dereferencing a handle) when POINTERARGS is 0 and to arg = argx otherwise. On a Macintosh, you should dereference any handle argument again after calling back to a function internal to MacAnova.

Use commandH = THECOMMAND(argx) to obtain a handle (char **) to a CHARACTER argument that is to be an argument to the mvEval() call back function.

Use symhArg = THESYMBOL(argx) to obtain a Symbolhandle (Symbol **) for a symbol type argument.

Use CALLBACKFUN(funlist, funName) to obtain a pointer to function funName in the list of call back functions.

In any user function making callbacks define C macro MVCALLBACKS before include Userfun.h. This results in the declaration of MvCallbackFuns, a global handle or pointer (depending on the value of POINTERARGS) to a call back function list. It this case, one of the first executable lines in the user function should be setMvFuns(funlist) to initialize MvCallbackFuns. Subsequently you can call the standard call back functions by mvPrint(msg), mvAlert(msg), mvEval(cmd), mvIsmissing(&x), mvSeterror(errorNumber), and mvFindfun(funName), where msg, cmd and funName have type char *, x has type double and errorNumber has type long. For example, mvPrint("Hello!") expands to CALLBACKFUN(MvCallbackFuns, print)("Hello").

When compiling for a 68K Macintosh, Userfun.h defines PRAGMAMPWC. This is to be used as follows:

```
  #ifdef PRAGMAMPWC
  #pragma mpwc on
```

```
  #endif

  Declaration of arginfo or call back function(s)

  #ifdef PRAGMAMPWC
  #pragma mpwc off
  #endif
```

This ensures the use of function calling conventions that are compatible with the 68K version of MacAnova which is compiled using MPW C. See the sample files goo.c and fooeval.c below for examples of the use of PRAGMAMPWC.

If the user function makes call backs and has more than one source file, define USERSUBFUNCTION in all but one source file. This assures that MvCallbackFuns will not be multiply defined.

In topic user_fun is C code not using these macros for user function goo and its arginfo function arginfo_goo. Separate versions are given there for Macintosh and non-Macintosh use. Here is C code for these functions that uses the macros. It should compile correctly on all platforms. When compiled for a 68K Macintosh, two compilation runs will be required, changing the value of WHICHFUN (1 for goo, 2 for arginfo_goo).

Use of these macros requires that MACINTOSH be defined when compiling for a Macintosh (with powerc also defined for PPC and MW_CW defined if using Metrowerks CodeWarrior compiler), DJGPP is defined when compiling for use with the protected mode DOS version, and WIN32 is defined when compiling for use with the Windows version.

The use of macros such as USERFUN and ARGINFO to define function names is needed to meet the requirements for Macintosh compilation for which an entry point must be named 'main'. The conditional compilation of the user function and arginfo function (depending on whether MAINFUN and/or INFOFUN is defined) is in response to limitations on compilations for a 68K Macintosh for which there can be only one entry point per resource.

We suggest the examples below, source for which is distributed with MacAnova) be used as templates, changing most of the executable code and the augument lists to meet your particular needs.

File goo.c:

```
  #include "Userfun.h"

  #ifndef MACINTOSH
  #define MAINFUN    /*main entry will be compiled*/
  #ifndef DJGPP
  #define INFOFUN    /*arginfo entry will be compiled*/
  #endif /*DJGPP*/
  #define USERFUN goo
  #define ARGINFO arginfo_goo
  #else /*!MACINTOSH*/
```

```
/*
    For PPC MAC, one function must be called main
    For 68K MAC, only one function is reachable per compilation
    project and it must be called main
*/
#ifdef powerc
#define MAINFUN    /*main entry will be compiled*/
#define INFOFUN    /*arginfo entry will be compiled*/
#define main_goo main
#else /*powerc*/

/* define which of the 2 functions will be compiled*/
#define WHICHFUN 1  /*must be 1 or 2 */
#if WHICHFUN == 1
#define MAINFUN
#else
#define INFOFUN
#endif

#if defined(MAINFUN)
#define main_goo    main
#else
#define info_goo    main
#endif

#endif /*powerc*/

#define USERFUN      main_goo
#define USERFUNENTRY goo
#define ARGINFO      info_goo
#define ARGINFOENTRY arginfo_goo

#endif /*!MACINTOSH*/

#ifdef MAINFUN
/*
  EXPORTED is _export for Windows compiled by Borland C/C++ 4.5
  DOUBLEARG(argx) expands as double * argx or double ** argx, and
  similarly for LONGARG(argn)
*/
void EXPORTED USERFUN(DOUBLEARG(argx), DOUBLEARG(argy), LONGARG(argn),
    DOUBLEARG(argresult))
{
    /* THEARG(argx) expands as argx or *argx */
    double      *x = THEARG(argx);
    double      *y = THEARG(argy);
    long        *n = THEARG(argn);
    double      *result = THEARG(argresult);
    int          i;

    EnterCode();
    *result = 0.0;
    for (i = 0; i < *n; i++)
```

```
    {
        *result += x[i]*y[i];
    }
    ExitCode();
}
#endif /*MAINFUN*/

#ifdef INFOFUN
static long Gooarginfo[] =
{
    4, NOCALLBACK | POINTERUSE | NOSYMBOLARGS,
    NONMISSINGREALVECTOR, NONMISSINGREALVECTOR, LONGSCALAR, REALSCALAR
};

#ifdef PRAGMAMPWC /*PRAGMAMPWC defined in Userfun.h only for 68K Mac*/
#pragma mpwc on
#endif /*PRAGMAMPWC*/
long * EXPORTED ARGINFO(void)
{
    long         *arginfo;

    EnterCode();

    arginfo = Gooarginfo;
    CHECK68881(arginfo);

    ExitCode();

    return(arginfo);
}
#ifdef PRAGMAMPWC
#pragma mpwc off
#endif /*PRAGMAMPWC*/
#endif /*INFOFUN*/

#ifdef powerc /*powerc defined means compiling for Power PC*/
RoutineDescriptor USERFUNENTRY =
    BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo04, USERFUN);
RoutineDescriptor ARGINFOENTRY =
    BUILD_ROUTINE_DESCRIPTOR(uppArgInfoEntryProcInfo, ARGINFO);
#endif /*powerc*/
```

In topic arginfo_fun is C code not using the macros in Userfun.h for user function fooeval and its arginfo function arginfo_foo. Here is C code using the macros that should compile correctly on all platforms using the C macros defined in Userfun.h. Since fooeval calls back to MacAnova, it must define MVCALLBACKS before including Userfun.h. This allows use of macros such as mvPrint and mvEval to call back to MacAnova. See topic callback_fun or header file Userfun.h for information on type sprintftype.

File fooeval.c:

```
#define MVCALLBACKS   /*enables call backs; required before include*/
```

```
#include "Userfun.h"

#ifndef MACINTOSH
#define MAINFUN  /*if defined, compile fooeval*/
#ifndef DJGPP
#define INFOFUN  /*if defined, compile arginfo function for fooeval*/
#endif
#define USERFUN fooeval
#define ARGINFO arginfo_fooeval
#else /*MACINTOSH*/

/*
    For PPC MAC, one function must be called main
    For 68K MAC, only one function is reachable per compilation
    project and it must be called main
*/
#ifdef powerc
#define MAINFUN
#define INFOFUN
#define main_fooeval main
#else /*powerc*/
/* define which of the 2 functions this project is for*/
#define WHICHFUN 1

#if WHICHFUN == 1
#define MAINFUN
#else
#define INFOFUN
#endif

#if defined(MAINFUN)
#define main_fooeval main
#else
#define main_info    main
#endif

#endif /*powerc*/
#define USERFUN       main_fooeval
#define USERFUNENTRY  fooeval
#define ARGINFO       main_info
#define ARGINFOENTRY  arginfo_fooeval

#endif /*MACINTOSH*/

#ifdef MAINFUN
void EXPORTED USERFUN(CHARARG(commandarg), CALLBACKLIST(funlist))
{
    char              **commandH = THECOMMAND(commandarg);
    Symbolhandle        result;
    char                line[200];
    sprintftype         sprintf; /*type defined in Userfun.h*/

    EnterCode();
```

```
    setMvFuns(funlist); /*initializes global duplicate of funlist*/

    sprintf = (sprintftype) mvFindfun("sprintf");

    result = mvEval(commandH);

    if (result != (Symbolhandle) 0)
    {
        switch (TYPE(result))
        {
          case CHAR:
            sprintf(line, "STRINGPTR(result) = '%s'", STRINGPTR(result));
            break;

          case REAL:
            if (!mvIsmissing(&DATAVALUE(result,0)))
            {
                sprintf(line, "DATAVALUE(result,0) = %.17g",
                      DATAVALUE(result,0));
            }
            else
            {
                sprintf(line, "DATAVALUE(result,0) = MISSING");
            }

            break;

          case LOGIC:
            sprintf(line, "DATAVALUE(result,0) = %c",
                    (DATAVALUE(result,0)) ? 'T' : 'F');
            break;

          case NULLSYM:
            sprintf(line, "Result is NULL");
            break;

          default:
            sprintf(line,
              "Type %ld of result not CHARACTER, REAL, LOGICAL, or NULL",
              TYPE(result))
        }
        mvPrint(line);
    }
    else
    {
        mvAlert("ERROR: Command produced error");
      /*tell User() error occurred but no message should be printed*/
        mvSeterror(silentCallbackError);
    }
    ExitCode();
} /*fooeval()*/
#endif /*MAINFUN*/
```

```
#ifdef INFOFUN
static long Fooevalarginfo[] =
      {1, DOESCALLBACK | POINTERUSE | NOSYMBOLARGS, CHARSCALAR};

#ifdef PRAGMAMPWC
#pragma mpwc on
#endif /*PRAGMAMPWC*/

long * EXPORTED ARGINFO(void)
{
    long         *arginfo;

    EnterCode();

    arginfo = Fooevalarginfo;
    CHECK68881(arginfo);
    ExitCode();
    return(arginfo);
}
#ifdef PRAGMAMPW
#pragma mpwc off
#endif /*PRAGMAMPW*/

#endif /*INFOFUN*/

#ifdef powerc
RoutineDescriptor USERFUNENTRY =
   BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo02, USERFUN);
RoutineDescriptor ARGINFOENTRY =
   BUILD_ROUTINE_DESCRIPTOR(uppArgInfoEntryProcInfo, ARGINFO);
#endif /*powerc*/
```

11.3 callback_fun

Usage:

```
Type userfunhelp(user_fun) for information on the structure of user
  functions.
Type userfunhelp(callback_fun) for information on the structure of user
  functions making "call backs" to MacAnova.
Type userfunhelp(arginfo_fun) for information on how to enable automatic
  checking of arguments to a user function.
```

Keywords: user functions, coding, sample source

```
This topic provides a brief introduction to the form of a user function
that makes "call backs" (executes functions internal to MacAnova).
Because of the inherent dependence on the computer and operating system,
there are many details that are not covered here.  Additional details
may be found in topics compile_dos, compile_mac, compile_unx and
compile_win.
```

```
It presumes familiarity with topic user_fun which describes the
structure of user functions not making call backs.

See headerfile Userfun.h distributed with MacAnova for C macros that are
helpful in writing user functions.

See loadUser() and User() for information on how to load and execute a
user function.

See topic arginfo_fun for information on how to make it possible for
MacAnova to obtain information about a user function for automatic
argument checking.

Since we have no experience in making call backs from a Fortran routine,
no Fortran tips are given.

In the following, 'handle' is used in the Macintosh OS sense, as a
pointer to a pointer.

          Structure of user functions calling back to MacAnova
In addition to regular arguments (pointers or handles to data or
symbols; see topic user_fun), a user function that calls back to
MacAnova must have an extra argument providing a list of functions that
can be called.  This is either a pointer (non-Macintosh) or handle
(Macintosh) to a MacAnovaCBS structure (defined in header file
dynload.h, included by header file Userfun.h).

Example of non-Macintosh declaration
  void fooclbck(char * m, MacAnovaCBS * funlist)
or
  void fooclbck(char * m, MacAnovaCBSPtr funlist)

Example of Macintosh declaration
  void fooclbck(char ** m, MacAnovaCBS ** funlist)
or
  void fooclbck(char ** m, MacAnovaCBSH funlist)

Here is the current definition of a MacAnovaCBS structure taken from
header file dynload.h:

  typedef struct MacAnovaCBS
  {
      void          (*print)(char *);
      void          (*alert)(char *);
      Symbol **     (*eval)(char **);
      long          (*ismissing)(double *);
      void          (*seterror)(long);
      void *        (*findfun)(char *);
  } MacAnovaCBS, *MacAnovaCBSPtr, **MacAnovaCBSH;

All the components are pointers to single argument functions internal to
MacAnova.
```

'print' points to mvPrint which expects a pointer to null terminated character vector (a "string") as argument. It inserts the string in the MacAnova output stream, usually the screen or command/output window. Virtually all MacAnova output is printed with mvPrint. If output is being spooled to a file (see spool()), mvPrint correctly handles it.

'alert' points to mvAlert() which expects a pointer to a string as argument. In a windowed version (Macintosh, Windows, Motif), this displays the string in a dialog box. In other versions, 'alert' is equivalent to 'print'.

'eval' points to mvEval which expects a handle to a character string (type char **) as argument. mvEval evaluates this string as if it were input to MacAnova, almost as if it were the text of a macro, and returns a handle of type Symbolhandle as value. Just as in a macro, this is the value of the last expression evaluated. C macros in dynload.h allow access to the type (REAL, CHARACTER, ...), dimension and value of the value returned by mvEval. The argument to mvEval must be a handle (char **) even in a non-Macintosh user function.

'ismissing' points to mvIsmissing which expects a pointer to double (double *) as argument. If x is a pointer to a double vector, mvIsmissing(&x[i]) (or mvIsmissing(x+i)) returns 1 if x[i] is MISSING and 0 otherwise.

'seterror' points to mvSeterror which expects a long integer as argument. mvSeterror(code) sets a variable that will be checked by User() on return. If the value is non-zero User() treats it as an error. Unless code = silentCallbackError (defined whenever MacAnovaCBS is defined), User will print the value.

'findfun' points to mvFindfun which expects a pointer to a string containing the name of an internal MacAnova function as argument. mvFindfun returns a pointer to void (C type void *) which must be cast to a function pointer of the appropriate type. A NULL return value indicates the function could not be found.

On some systems, you may be able to access any function known to MacAnova; on others, the available functions are limited to those in a short list. The functions available always include the following functions that can be used to allocate and de-allocate memory and to create and decode character strings. Additional functions may be available on other systems.

`char ** mygethandle(long n)`	Allocate n bytes of memory and return handle to the space allocated
`void mydisphandle(char ** x)`	De-allocate memory referenced by handle x
`char ** mygrowhandle(char **x, long n)`	Allocate n bytes, copy at most n bytes of x to it and then de-allocate x.

```
  int sprintf(char * bf, char * fmt, ...) Formatted "print" to buffer bf
  int sscanf(char * bf, char * fmt, ...)  Formatted "scan" of buffer bf
```

C types for these functions are defined in header file Userfun.h so that you can use the following to declare local pointers to them:

```
  mygethandletype      mygethandle;
  mydisphandletype     mydisphandle;
  mygrowhandletype     mygrowhandle;
  sprintftype          sprintf;
  sscanftype           sscanf.
```

See below for an example.

Memory management functions mygethandle, mydisphandle and mygrowhandle work with handles (pointers to pointers) in all versions.

The char ** arguments to mydisphandle and mygrowhandle must have been allocated by mygethandle.

On a Macintosh, although memory is allocated using Macintosh OS function NewHandle, the values returned by mygethandle and mygrowhandle cannot be used as handle arguments to Macintosh OS functions such as DisposHandle.

It appears that calling back to these sprintf and sscanf is the only way to use them in the protected mode DOS version; in other versions, you can probably use them directly.

In writing a 68K Macintosh user function, to ensure correct compilation, all declarations of call back and arginfo functions must be bracketed by

```
  #pragma mpwc on
  ...
  #pragma mpwc off
```

This is because the released 68K versions of MacAnova are compiled using MPW C.

Here is an example of a function that calls back to MacAnova. It uses mvEval to evaluate its first argument as a command and then uses call back functions to print a message describing the result of the evaluation. A typical use might be User("fooeval","sqrt(2*PI)").

Non-Macintosh version:

```
  #include "Userfun.h"

  void fooeval(char * commandarg, MacAnovaCBSPtr funlist)
  {
      char              **commandH = &commandarg;
      Symbolhandle        result;
      char                line[200];
      void              (*mvPrint)(char *) = funlist->print;
```

```
void              (*mvAlert)(char *) = funlist->alert;
long              (*mvIsmissing)(double *) = funlist->ismissing;
Symbolhandle      (*mvEval)(char **) = funlist->eval;
void              (*mvSeterror)(long) = funlist->seterror;
void             *(*mvFindfun)(char *) = funlist->findfun;
sprintftype         sprintf;

/* the code from here to END is the same for any version */
sprintf = (sprintftype) mvFindfun("sprintf");

result = mvEval(commandH); /* have MacAnova evaluate the command*/

if (result != (Symbolhandle) 0)
{
    /*
      C macros TYPE, STRINGPTR, DATAVALUE and constants
      CHAR, REAL, LOGIC, and NULLSYM are defined in Userfun.h
      along with other macros for working with Symbols
      */
    switch (TYPE(result))
    {
      case CHAR:
        sprintf(line,
                "STRINGPTR(result) = '%s'", STRINGPTR(result));
        break;

      case REAL:
        if (!mvIsmissing(&DATAVALUE(result,0)))
        {
            sprintf(line,
                    "DATAVALUE(result,0) = %.17g", DATAVALUE(result,0));
        }
        else
        {
            sprintf(line, "DATAVALUE(result,0) = MISSING");
        }

        break;

      case LOGIC:
        sprintf(line, "DATAVALUE(result,0) = %c",
                (DATAVALUE(result,0)) ? 'T' : 'F');
        break;

      case NULLSYM:
        sprintf(line, "Result is NULL");
        break;

      default:
        sprintf(line,
         "Type of result not CHARACTER, REAL, LOGICAL, or NULL");
    }
    mvPrint(line);
```

```
      }
      else
      {
          mvAlert("ERROR: Command produced error");
  /*Tell User an error has occurred but code should not be printed*/
          mvSeterror(silentCallbackError);
      }
      /* END */
  }

Macintosh version:

  #include "Userfun.h"
  /*
    On a Macintosh, the main entry must be called main;  the function
    is found by the name of its code resource
  */
  void main(char ** commandarg, MacAnovaCBSH funlist)
  {
      char              **commandH = commandarg;
      Symbolhandle        result;
      char                line[200];
  #ifndef powerc /*if compiled for 68K Macintosh*/
  #pragma mpwc on
  #endif
      void              (*mvPrint)(char *) = funlist->print;
      void              (*mvAlert)(char *) = funlist->alert;
      long              (*mvIsmissing)(double *) = funlist->ismissing;
      Symbolhandle      (*mvEval)(char **) = funlist->eval;
      void              (*mvSeterror)(long) = funlist->seterror;
      void             *(*mvFindfun)(char *) = funlist->findfun;
  #ifndef powerc
  #pragma mpwc off
  #endif
      EnterCode();
  /* the code from here to END is the same for any version */
      . . . . . see above . . . . .
  /* END */
      Exitcode();
  }

  #ifdef powerc /*powerc defined means compiling for Power PC*/
   RoutineDescriptor fooeval =
      BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo02,main);
  #endif /*powerc*/
```

11.4 compile_dos

Usage:

```
Type userfunhelp(compile_dos) for information on how to compile a user
  function for use with the protected mode DOS version of MacAnova.
```

Keywords: user functions, compiling

```
This topic provides some details about compiling a user function for use
with the protected mode DOS version of MacAnova.  User functions are not
implemented in the real mode version (BCPP).  Compilation uses version
2.0 of the DJGPP compiler.

DJGPP uses the dxe format for loading.  This is a simple but restricted
method of loading.  In particular, you can access only one entry point
(function) in a file. You should compile the file with gcc as usual as
in
  gcc -c goo.c subs.c
and then run dxegen (supplied with DJGPP 2.0) as in
  dxegen goo.dxe _goo goo.o subs.o -lm -lc

The first two arguments to dxegen are the output file and the entry
point to be made visible for loading (note the prepended underscore).
These are followed by the compiled (*.o) files and arguments specifying
libraries to be searched.

There are some restrictions on your source file.  Not all library
functions may be used.  Excluded functions include input/output
functions and their relatives such as sprintf and sscanf.  In addition
there are some naming restrictions.  For example, you can't have
functions foo and foo2 and try to load entry point _foo, but you could
have foo and dofoo (it seems that the leading string must be unique).

Because sprintf and sscanf are frequently needed, (sprintf is often
used to build output lines or error messages), they are included in the
list of functions known to mvFindfun.

Because you can have only one entry point using dxe files, you cannot
also provide arginfo_foo() to check the number and types of arguments.
Moreover, you must use 'callback:T' and/or 'symbols:T' on User() when
executing a user function that makes call backs and/or expects
Macanova symbols as arguments.
```

11.5 compile_mac

Usage:

```
Type userfunhelp(compile_mac) for information on how to compile a user
  function for use with Macintosh versions of MacAnova.
```

Keywords: user functions, compiling

```
This topic provides some details about compiling a user function for use
with the Macintosh versions of MacAnova.  It assumes the Metrowerks
CodeWarrior compiler is used.

The Macintosh is a bit more complicated than other platforms, since
```

there are two kinds of processors (PPC and 68K) to support. The 68K case is further complicated by the fact that code may or may not compiled to use a 68881 math coprocessor.

In coding a Macintosh user function, if you make a pointer by dereferencing a handle argument, you should dereference it again after calling back to a function internal to MacAnova since its location in memory may have been changed by the call back.

User functions and arginfo functions are compiled into code resources in files of type 'rsrc'. PPC resources must have resource type 'MVPP'; 68K resources not requiring a 68881 coprocessor must have resource type 'MV6n'; and 68K resources requiring a coprocessor must have resource type 'MV6c'.

User() accesses the resources themselves by name. The resource for a function should have the name, say 'foo', you will give in your User() call or 'arginfo_foo. All resources of the same type should in a file should have distinct resource numbers, say 4000, 4001, ... A PPC user function may be in a resource with a different name, in which case you have to provide the name of the resource using User(funName, resource:resName, ...).

Source files for both PPC and 68K user functions must have a function named 'main', plus possibly other functions.

PPC code resources, but not 68K ones, can have additional entry points, usually an arginfo function, but occasionally other user functions.

Macintosh 68K user functions

For 68K user functions, it is necessary to set 68000 register A4 so that global variables will be found. Using CodeWarrior, this is accomplished by including

```
EnterCodeResource();
```

immediately after declaring local variables and before any reference to global variables, and including

```
ExitCodeResource();
```

immediately before returning. When C macro MACINTOSH is defined but powerc is not, macros EnterCode() and ExitCode() defined in header file Userfun.h expand to EnterCodeResource() and ExitCodeResource(). Otherwise they expand to nothing.

68K code resources have just one entry point which must be named 'main', so a single resource cannot include both a user function and its arginfo function. However, the Codewarrior compiler has a "Merge to file" option that allows you to add to an existing resource file the resource created when compiling a function.

PPC User Functions

PPC code resources may have multiple entry points which are taken from the names of global RoutineDescriptor variables in the source. Their names should similar to 'goo' and 'arginfo_goo' or 'fooeval' and 'arginfo_fooeval'. The name given to the function that actually codes

the user function should be 'main' and the name given to the function with the arginfo function code should be something like 'main_info' different from the name given to the arginfo. Here are typical RoutineDescriptor declarations.

```
  RoutineDescriptor goo =
      BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo04, main);
  RoutineDescriptor arginfo_goo =
     BUILD_ROUTINE_DESCRIPTOR(uppArgInfoEntryProcInfo, info_main);
```

Argument uppMainEntryProcInfo04 is appropriate for a user function expecting 4 arguments, including the call back function list. For a function expecting 5 arguments, use uppMainEntryProcInfo05, and so on.

Since you will normally use the funName given in the User("funName",...) call for both the name of the resource and the name of the RoutineDescriptor entry point, you will ordinarily include only one user function (and its arginfo function) in a resource. If you include more than one user function in a resource, you must use keyword phrase 'resource:Resname' to specify the resource name. An arginfo function must be in the same resource as its user function.

Setting up Macintosh 68K projects

Here is how to set things up to create a 68K code file with resources 'fooeval' and 'arginfo_fooeval' based on file fooeval.c listed in topic c_macros. This has been written in such a way that only one of fooeval or arginfo_fooeval will be compiled, depending on the value of C macro WHICHFUN.

(1) Create two MacOS 68K CodeWarrior project files, one for fooeval and the other for arginfo_fooeval. They should both have source files fooeval.c, Userfun.h and dynload.h. See below for library files needed.

(1.a) Specify Code Resource for the project type for both projects.

(1.a) For both projects specify resource type 'rsrc' and the same resource file, say, Fooeval.rez as Project options. The fooeval project should specify 'fooeval' and 4000 as the resource name and number. The arginfo_fooeval project should specify 'arginfo_fooeval' and 4001 as the resource name and number, and should have Merge to File checked. The resource numbers are arbitrary but should be different. The resource type should be 'MV6c' or 'MV6n', depending on whether you are compiling to use a 68881 math coprocessor.

(1.b) Both projects should specify processor options 68020 Codegen, 4 byte ints, 8 byte doubles and far data. If you are compiling to use a math coprocess, also specify 68881 Codegen,

(1.c) Set linker options Link Single Segment.

(1.d) For a user function making call backs (as does fooeval), C/C++ language option MPW Newlines should be checked.

(1.e) If you reference any C library functions such as strcpy (fooeval does not) you will need library file 'ANSIFa(N/4i/8d)C.A4.68K.Lib' and possibly 'MathLib68K Fa(4i/8d).A4.Lib' ('ANSIFa(N/4i/F/8d)C.A4.68K.Lib' and 'MathLib68K Fa(4i/f/8d).A4.Lib' if compiling to use a 68881 math comprocessor). You may also need MacOS.lib. For example, all three libraries are needed if you use the library version of sprintf. None is required for fooeval.c as written.

(1.f) Both projects should specify a prefix file, say Userfun.pch, containing at least the following:
```
 #define MACINTOSH
 #define MW_CW
```

(2) Edit fooeval.c to define C macro WHICHFUN as 1 and compile the fooeval project.

(3) Re-edit fooeval.c to define WHICHFUN as 2 and compile the arginfo_fooeval project.

You end up with one resource file 'Fooeval.rez' containing resources 'fooeval' and 'arginfo_fooeval'.

Setting up Macintosh PPC projects

Here is how to set up a CodeWarrior project to compile a PPC version of fooeval using source file fooeval.c listed in topic c_macros.

(1) Create a MacOS PPC project with source files fooeval.c, Userfun.h and dynload.h and library files MPCRuntime.Lib and InterfaceLib. See below for other library files.

(1.a) Specify Code Resource for the project type

(1.b) Specify resource file 'fooevalppc.rez' of type 'rsrc'. The resource type must be 'MVPP'. The resource name should be 'fooeval'. The resource number should be different from any other PPC user functions you might be using simultaneously.

(1.c) Specify Main entry 'main' for the PPC linker.

(1.d) For a user function making call backs (as does fooeval), C/C++ language option MPW Newlines should be checked.

(1.e) If you reference any C library functions such as strcmp you should add libraries 'ANSI C.PPC.Lib' and 'MathLib' and file 'console.stubs.c'.

(2) Compile the fooeval PPC project.

You end up with a resource file fooevalppc.rez containing resource 'foo'. You load it into MacAnova by loadUser("fooevalppc.rez") and execute it by, say, User("fooeval","exp(-x^2/2)/sqrt(2*PI)").

11.6 compile_unix

Usage:

```
Type userfunhelp(compile_unix) for information on how to compile a user
  function for use with a Unix version of Macanova, including Motif.
```

Keywords: user functions, compiling

```
This topic provides some details about compiling a user function to be
used with a Unix version of MacAnova (including Unix Motif).

                      Hewlett-Packard UX (HPUX)
The file to load must be a shared library.  This can be constructed, for
example, by
  cc -c +z -Aa fooeval.c
  ld -b fooeval.o -o fooeval.sl
It should be loaded by loadUser("fooeval.sl").

At present (version 4.05 release 1), there may be unsatisfied external
problems when linking with system libraries.

                         Other Unix Versions
Compilation and linking commands may be different.  MacAnova will have
to have been compiled and linked with functions for using a shared
library.  The actual loading of shared libraries and execution of
routines in them is done in file dynload.c.  Currently (July 1997) this
has been coded only for HPUX (using shl_load() and shl_findsym()).  Most
Unix versions will have similar functions.  For example, on IRIX, the
corresponding functions are dlopen() and dlsym().
```

11.7 compile_win

Usage:

```
Type userfunhelp(compile_win) for information on how to compile a user
  function for use with the Windows version of MacAnova.
```

Keywords: user functions, compiling

```
This topic provides some details about compiling a user function using
Borland C/C++ for use with the Windows version of MacAnova.

Set up the project to construct a 32 bit DLL.

Change the default Project Options as follows:
  Add WIN32 to the list of defines
  Set 32-bit Compiler Processor Data Alignment to Quad word (8 bytes).
  Set Resources Target Windows Version to Win32

In the source, add the modifier "_export" to the functions in the
project, as in
  void _export goo(double *x, double *y, long *n, double *result)
```

```
  long * _export goo_arginfo(void)

This will be accomplished automatically if you include header file
Userfun.h, and preface routine names with EXPORTED instead of _export.

Entry names will be prefixed by the compiler with '_'.  For this reason,
User("_foo", ... ) and User("foo", ... ) are equivalent.

Files goo.c and fooeval.c listed in topic c_macros are examples of user
functions that compile for Windows.
```

11.8 loadUser

Usage:

```
   loadUser(fileName [,reload:T or clear:T]), CHARACTER scalar fileName.
```

Keywords: user functions, loading

```
loadUser(FileName) loads a user function (separately compiled routine)
into MacAnova.  FileName should be a quoted string or CHARACTER scalar
giving the name of the file containing the user function to be loaded.
Once loaded, a user function can be executed by function User().  As
usual, in windowed versions (Macintosh, Windows, Motif), FileName can be
"".  If the file has been previously loaded, it is not reloaded, but it
may be put at the start of the entry search list for the next use of
User().

loadUser(FileName, reload:T) does the same, except that the file will be
reloaded, even if it has been previously loaded into MacAnova.

loadUser(FileName, clear:T) does the same, except all previously loaded
files will be forgotten.

On some systems, the user function can be written in Fortran, although
some features such as call back functions and argument checking may not
be available.

Functions loadUser() and User() are inherently specific to a particular
computer system although it is possible to write user functions that can
be compiled on multiple systems without change.

Unix:
  FileName must be the name of a shared library.
Windows:
  FileName must be the name of a DLL.
Protected mode DOS (DJGPP):
  FileName must be the name of a dxe file.
Macintosh:
  FileName must be the name of a file containing one or more code
  resources.  The PPC version of MacAnova can call both 68K and PPC code
```

```
  resources, but a 68K version of MacAnova can call only 68K code
  resources.  Resource types must be one of 'MVPP' (PPC), 'MV6n' (68K
  without coprocessor) or 'MV6c' (68K with coprocessor).

See also topics User() and user_fun.  Type userfunhelp(User) or
userfunhelp(user_fun).
```

11.9 type_codes

Usage:

```
   Type userfunhelp(type_codes) for a complete list of argument type and
     shape codes to be returned by an arginfo function.
```

Keywords: user functions, coding

```
This topic lists the type and shape codes that may be used by an arginfo
function to provide information about the arguments expected by a user
function (see topic arginfo_fun).  They are all integer constants
defined in header file Userfun.h.

Scalar argument (all dimensions 1)
  CHARSCALAR, LOGICSCALAR, REALSCALAR, NONMISSINGREAL, POSITIVEREAL,
  NONNEGATIVEREAL, INTSCALAR, POSITIVEINT, NONNEGATIVEINT, LONGSCALAR,
  POSITIVELONG, NONNEGATIVELONG

Vector argument (all dimensions beyond first, if any, are 1)
  CHARVECTOR, LOGICVECTOR, REALVECTOR, NONMISSINGREALVECTOR,
  POSITIVEVECTOR, NONNEGATIVEVECTOR, INTVECTOR, POSITIVEINTVECTOR,
  NONNEGATIVEINTVECTOR, LONGVECTOR, POSITIVELONGVECTOR,
  NONNEGATIVELONGVECTOR

Matrix argument (no more than 2 dimensions >= 1)
  CHARMATRIX, LOGICMATRIX, REALMATRIX, NONMISSINGREALMATRIX,
  POSITIVEMATRIX, NONNEGATIVEMATRIX, INTMATRIX, POSITIVEINTMATRIX,
  NONNEGATIVEINTMATRIX, LONGMATRIX, POSITIVELONGMATRIX,
  NONNEGATIVELONGMATRIX

Square matrix argument
  REALSQUAREMATRIX

Array argument
  CHARARRAY, LOGICARRAY, REALARRAY, NONMISSINGREALARRAY, POSITIVEARRAY,
  NONNEGATIVEARRAY, INTARRAY, POSITIVEINTARRAY, NONNEGATIVEINTARRAY,
  LONGARRAY, POSITIVELONGARRAY, NONNEGATIVELONGARRAY

Arbitrary Symbol argument
  SYMHVALUE

The qualifiers INT, POSITIVE and NONNEGATIVE imply all elements must be
non-MISSING.
```

These codes are applicable both for user functions whose arguments are pointers or handles to data, and for user functions whose arguments are pointers or handles to MacAnova symbols. SYMHVALUE should only be used when symbol arguments are expected and then only when the argument is not restricted to one type and shape or may has type different from REAL, LOGICAL, CHARACTER or LONG.

11.10 User

Usage:

```
User(funName [,resource:resName][,control keyword phrases],arg1 [,...]),
  funName and resName CHARACTER scalars; control keyword phrases are any
  of callback:T, symbols:T, pointers:T and quiet:T; arg1, ... arguments
  to a user function; if argument is keyword phrase other than
  'protect:arg', it is returned, possibly modified.
```

Keywords: user functions, executing

User(FuncName, arg1, arg2, ...) executes a user function, that is, a compiled routine external to MacAnova. Quoted string or CHARACTER scalar FuncName specifies the name of a user function whose code is in a file previously loaded by loadUser(). arg1, arg2, ... are arguments that will be passed to the function. You must have a least one argument in addition to FuncName and no more than 20 (13 in the Macintosh PPC version). Depending on the compiler and system, you may be required to include leading or trailing underscore characters '_' in FuncName, say User("foo_",...) or User("_foo", ...) instead of User("foo", ...).

A 68K version of MacAnova cannot execute a user function compiled for a PPC.

User(FuncName, quiet:T, arg1, ...) does the same except warning messages, if any, are suppressed.

User(FuncName, callback:T, arg1, ...) specifies the function is known to "call back" to MacAnova, that is, to execute functions internal to MacAnova. On a Macintosh, the type of user function (PPC or ordinary 68K) must match the version of MacAnova. See topic 'callback_fun' in file userfun.hlp (type userfunhelp(callback_fun)).

If the MacAnova version requires a 68881 math coprocessor, there can be problems if a user function that makes call backs does not require a coprocessor.

User(FuncName, symbols:T, arg1, ...) specifies that all the arguments are to be passed as complete MacAnova "symbols", including all dimension information. This should be used only with a user function specifically written to make use of MacAnova symbols.

The PPC Macintosh version cannot pass symbol arguments to a 68K user function.

User(FuncName, pointers:T or F, arg1, ...) changes the default way arguments are passed, either as "pointers" (pointers:T) or as "handles" (pointers:F). On all but Macintosh computers, the default is pointers:T. You are unlikely ever to need to use this keyword.

User(FuncName, resource:ResName, arg1, ...) specifies the name of the PPC Macintosh resource containing the user function. This option is not needed in other versions and needed on a PPC only when the resource name differs from the function name.

You can use more than one of the preceding keywords phrases together (User("goo", resource:"foo",quiet:T,callback:T,x,result:0)).

On most systems, it is possible to include with a user function an "arginfo" function that MacAnova can call to obtain information about the user function. The information includes the number of arguments expected, their types and shapes expected (for example, CHARACTER scalar, REAL matrix), and whether the function makes call backs or expects "symbol" arguments (see above). This allows automatic argument checking. If the function is compiled without an arginfo function, using the wrong number or type of arguments will usually result in a crash or other undesirable behavior. In particular, a user function will not be able to handle MacAnova symbol arguments unless it has been specially written to be able to understand their structure.

You normally do not need to use keywords 'callback', 'symbols' and 'pointers' if the user function has an associated arginfo function which is possible on all systems except protected mode DOS.

See topics user_fun and arginfo_fun in help file userfun.hlp for information about the form of a user function and an arginfo function (type userfunhelp(user_fun), say).

Interpretation of FuncName

Unix, Motif and Windows:
  FuncName should be the name of the function being called, possibly modified by a leading or trailing '_' (leading '_' when compiling for Windows with Borland C 4.5). In Windows and Unix versions where it is known entry names start with '_', when the function is not found using the name as provided, a second search is made after prepending '_' to the name. Thus if User("_foo", ...) would be successful, so will be User("foo", ...).

Extended memory DOS (DJGPP):
  FuncName should be the same as the name of the .dxe file loaded by loadUser() that contains the code except that directory information and the extension ".dxe" may be omitted. Thus after loadUser("../foo.dxe"), you can use any of User("../foo.dxe",...), User("foo.dxe",...) or User("foo",...). When there is more than one file with the same name attached (for example, "/a/foo.dxe" and "/b/foo.dxe", you should use the complete path name.

PPC Macintosh user function
  FuncName is the name of the user function. This will usually also be the name of the PPC code resource containing the user function. If it is not, you need to include keyword phrase 'resource:ResName' as an argument, where ResName is a quoted string or CHARACTER scalar specifying the resource name.

68K Macintosh user function:
  FuncName should be the name of the 68K code resource containing the user function (only one user function per resource). If 'resource:ResName' is an argument, ResName must be identical with FunName. It is an error to attempt to call a 68K user function that requires a 68881 or 68882 math coprocessor on a Macintosh without one.

User function arguments

All arguments except the function name and 'callback', 'quiet', 'symbols', 'pointers' and 'resource' keyword phrases are passed to the user function as its arguments

Only copies of keyword phrase arguments are passed to a user function. This means that the user function can modify these arguments without danger of changing any MacAnova variable.

Non-keyword phrase arguments to User() (except the user function name) are passed to the user function without being copied. When the argument is a named MacAnova variable and the user function modifies it, the value of the variable itself is changed. When the argument is a literal number (User("foo", 1, 2, 0)) or expression (User("foo", sqrt(2)+3, log(4), 19^2)) the function can safely change the argument without danger to any variable.

Example:
Suppose fooadd expects three arguments, and modifies the third by assigning the sum of the first two. Then

```
Cmd> c <- 0; User("fooadd", 1, 2, c)
```

returns no value (actually a NULL; see NULL), but c has been changed to 3 (= 1+2). However,

```
Cmd> c <- 0; User("fooadd", 1, 2, protect:c)
```

will not change c itself, but only a copy.

In addition to being copied before use, all keyword phrase user function arguments are returned, possibly modified, as the value of User(). When there are two or more such keyword arguments, a structure is returned with component names taken from the keyword names.

Keyword 'protect' is special, in that its only effect is to cause its value to be copied before being passed to the user function; its value is not returned by User(). Thus the use of 'protect' in the example

```
makes the user function useless: c does not get changed because it is
protected by a keyword, but the modified value is not returned either.
The following both protects c and causes the modified value of the last
argument to be returned as the value of User().

  Cmd> c <- 0;User("fooadd", 1, 2, result:c)

This returns 3 = 1 + 2, but c would be unchanged.  In place of variable
c for result, you could use any REAL scalar (result:0).  This serves to
provide space for the answer.

  Cmd> User("fooadd", left:1, right:2, result:0)

returns a structure with components 'left', 'right' and 'result'
containing the possibly modified values of the original arguments.

It is essential that the size of any argument that is to be modified
matches the size that is expected by the user function.  Thus, if foocat
is a user function that concatenates its first two arguments into a
third argument, the length of the third argument should be the combined
length:

  Cmd> User("foocat", run(4), run(7), combined:rep(0,11)

In this example, for the third argument to have fewer than 11 elements
would lead to unpredictable results, possibly even a crash of MacAnova.

Except when symbols:T is an argument, all user function arguments are
either REAL, LOGICAL, CHARACTER or LONG variables.  REAL arguments are
passed as double precision data, as are LOGICAL arguments (True = 1.0,
False = 0.0).  When an argument is a matrix or array, the values are
ordered such that the first subscript changes fastest.  See topic
'user_fun' in help file userfun.hlp for more information (type
userfunhelp(user_fun)).

LONG is a special MacAnova type whose values are long integers between
-2147483647 and 2147483647 = 2^31 - 1.  A LONG argument can be created
only by function asLong().  A LONG argument that is returned
(result:asLong(x)), is turned into an REAL quantity before being
returned.  See asLong().

  Cmd> User("goo", run(10), run(10), asLong(10), result:0)

invokes user function goo with three REAL arguments and one long
argument.

For virtually unlimited flexibility, when keyword phrase 'symbols:T' is
an argument to User(), all user function arguments are passed as symbols
-- MacAnova objects which encapsulate the data, type, and dimensions of
a variable.  Thus

  Cmd> User("foo", symbols:T, x, y, result:z)
```

```
passes x, y and z to 'foo' as symbols.  You cannot have some user
function arguments be symbols and some just data; all must be symbols or
none.
```

11.11 userfunhelp()

Usage:

```
userfunhelp(topic1 [, topic2 ...] [,usage:T] [,scrollback:T])
userfunhelp(key:Key), CHARACTER scalar Key
userfunhelp(index:T [,scrollback:T])
```

Keywords: general

```
userfunhelp(Topic1 [, Topic2, ...]) prints help on topics Topic1,
Topic2, ... related to user functions.  The help is taken from file
Userfun.mac.

userfunhelp(Topic1 [, Topic2, ...] , usage:T) prints usage information
related to these topics.

userfunhelp(index:T) or simply userfunhelp() prints an index of the
topics available using userfunhelp.

In all three usages, you can also include help() keyword phrase
'scrollback:T' as an argument to userfunhelp.  In windowed versions,
this directs the output/command window will be automatically scrolled
back to the start of the help output.

userfunhelp(key:key) where key is a quoted string or CHARACTER scalar
lists all topics cross referenced under Key.  userfunhelp(key:"?")
prints a list of available cross reference keys for topics in the file.

userfunhelp is implemented as a predefined macro.

See help() for information on direct use of help() to retrieve
information from Userfun.hlp.
```

11.12 user_fun

Usage:

```
Type userfunhelp(user_fun) for information on the structure of user
  functions.
Type userfunhelp(callback_fun) for information on the structure of user
  functions making "call backs" to MacAnova.
Type userfunhelp(arginfo_fun) for information on how to enable automatic
  checking of arguments to a user function.
```

Keywords: user functions, coding, sample source

This topic provides a brief introduction to the form of a user function (routine compiled separately from MacAnova) that can be loaded by loadUser() and executed by User(). Because of the inherent dependence on the computer and operating system, there are many details that are not covered here. Additional details may be found in topics compile_dos, compile_mac, compile_unx and compile_win.

See headerfile Userfun.h distributed with MacAnova for C macros that are helpful in writing user functions.

See loadUser() and User() for information on how to load and execute a user function.

See topic callback_fun for information on the structure of a user function that makes call backs to Macanova. It presumes familiarity with this topic (user_fun).

See topic arginfo_fun for information on how to make it possible for MacAnova to obtain information about a user function for automatic argument checking.

On some systems, if you are willing to forego automatic argument checking, creating a user function file may be as simple as recompiling and linking existing code with certain options set. On others you may need to modify the code to include C header file Userfun.h which defines various constants and C macros. You will almost certainly need to use Userfun.h if you write a user function that makes call backs (executes routines internal to MacAnova) or provides argument checking capability.

Structure of a user function

Most of this discussion assumes the user function is written in C although a few tips are given for user functions written in Fortran. If you write a user function in Fortran, you need to be aware that all its arguments are pointers, that is the function receives the location in computer memory of each argument, not its value.

Header file Userfun.h should normally be included in the C source, especially if you expect that the user function will be compiled for more than one computer type. Userfun.h not only contains information that may be essential for compilation (including type declaration for symbols; see above), but also contains many C macros that make coding easier. In particular, it contains macros allowing you to write a user function that may be compiled with little or no change on Unix, Macintosh and Windows. However, to make clear the principles, the structure of a user function is illustrated without using these macros.

Note: Header file Userfun.h itself includes header dynload.h which is also distributed with MacAnova. Both need to be available when compiling a user function.

Value returned:

The user function should not return a value (C type void, Fortran subroutine).

Non-Macintosh argument types
  Each user function argument must be declared as a pointer. The legal types are double * (REAL or LOGICAL data), char * (CHARACTER data), long * (LONG data), or Symbol * (symbol argument). For Fortran, these are double precision, character and integer*4 (symbol not possible).

Example of non-Macintosh declaration:
  C:
```
 void goo(double * x, double * y, long * n, double * result)
```

  Fortran:
```
 subroutine goo(x, y, n, result)
 integer*4  n
 double precision  x(n), y(n), result
```

Macintosh argument types
  Each argument must be declared as a pointer to a pointer, known to Macintosh programmers as a "handle". Thus the legal types are double ** (REAL or LOGICAL data), char ** (CHARACTER data), long ** (LONG data), or Symbol ** (symbol argument. Type Symbolhandle declared in Userfun.h is equivalent to Symbol **.

  It is probably not possible to do this directly in Fortran. A C interface to the Fortran subroutine will be required.

Example of Macintosh declaration:
```
  void goo(double ** x, double ** y, long ** n, double ** result)
```

  When "handle" is used below, it always means a pointer to a pointer, not any of the various handles used when programming for Windows.

  The reason for the use of handles as arguments to functions is that MacAnova functions may allocate memory. On the Macintosh, this can move the contents of previously allocated memory, so that a pointer would no longer be valid. However, the handle remains valid even if the pointer it points to changes.

  If you make a pointer by dereferencing a handle argument, you should dereference it again after calling back to a function internal to MacAnova since its location in memory may have changed.

Executable statements
  There should be no direct input or output statements (you can do output using a callback function) or any direct memory allocation (also possible using a callback function). On a Macintosh, code must take into account the extra level of indirection of the handle arguments.

Here is C code for an example user function that computes the inner product of two real vectors.

Non-Macintosh version:

```
  C:
  #include "Userfun.h" /*not needed here as coded*/

  void goo(double * x, double * y, long * n, double * result)
  {
      int         i;

      *result = 0.0;
      for (i = 0; i < *n; i++)
      {
          *result += x[i]*y[i];
      }
  }

  Fortran:
        subroutine goo(x, y, n, result)
        integer*4        n
        double precision x(n), y(n), result
        integer*4        i

        result = 0.0d0
        do 2 i = 1, n
          result = result + x(i)*y(i)
  2     continue
        return
        end
```

In Windows, when compiling using Borland C/C++ 4.5, you need to replace "void goo" by "void _export goo".

Macintosh (both PPC and 68K) version:

```
  #include "Userfun.h" /* required */
  #define main_goo main /*entry must have internal name 'main'*/

  void main_goo(double ** argx, double ** argy, long ** argn,
           double ** argresult)
  {
      double     *x = *argx, *y = *argy, *z = *argz;
      long       *n = *argn;
      int         i;

      EnterCode();
      *result = 0.0;
      for (i = 0; i < *n; i++)
      {
          *result += x[i]*y[i];
      }
      ExitCode();
  }

  #ifdef powerc /*powerc defined means compiling for Power PC*/
   RoutineDescriptor goo =
      BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo04,main_goo);
```

```
  #endif /*powerc*/
```

The first executable statement in a 68K Macintosh user function must be EnterCodeResource(), and the last before the return must be ExitCodeResource(). EnterCode() and ExitCode() are C macros defined in Userfun.h that expand to these statements in a 68K Macintosh compilation and to nothing in a PPC, DOS, Windows, Motif, Unix or other compilation.

When coding for a PPC Macintosh, an additional statement declaring and initializing a RoutineDescriptor is required for each function. Constant uppMainEntryProcInfo04 is defined in dynload.h (automatically included by Userfun.h) and is appropriate for a function with 4 arguments.

Chapter 12

Search Key Tables

The tables in this Chapter relate the search keys used in each help file to the help topics that reference them. Note that not all help files contain search keys.

Table 12.1: Search Keys and References from MacAnova.hlp.txt

Search Key	References
ANOVA	anova(), anovapred(), cellstats(), coefs(), contrast(), design, designhelp(), factor(), fastanova(), glm, glm_keys, glmfit(), glmpred(), glmtable(), makefactor(), manova(), models, popmodel(), power(), power2(), predtable(), pushmodel(), regcoefs(), robust(), samplesize(), secoefs(), wtanova(), wtmanova(), yates()
Categorical Data	bin(), glm, glm_keys, glmfit(), glmpred(), ipf(), logistic(), poisson(), probit(), tabs()
CHARACTER Variables	array(), cat(), CLIPBOARD, compnames(), delete(), fromclip(), getascii(), inforead(), ischar(), makefactor(), makesymbols(), match(), nameof(), putascii(), replacestr(), syntax, toclip(), variables, varnames(), vector()
Combining Variables	array(), cat(), hconcat(), makecols(), makestr(), matrix(), rep(), rotate(), run(), select(), split(), strconcat(), structure(), trilower(), triunpack(), triupper(), vconcat(), vector()
Comparisons	contrast(), cumdunnett(), cumstudrng(), invdunnett(), invstu(), invstudrng(), propinterval(), proptest(), t2val(), tinterval(), ttest(), tval(), twotailt(), zinterval(), ztest()

Continued from previous page

Search Key	References
Complex Arithmetic	cconj(), cdivc(), cdivcj(), cft(), cimag(), cmplx(), complex, cpolar(), cprdc(), cprdcj(), creal(), crect(), ctoh(), hconj(), hdivh(), hdivhj(), hft(), himag(), hpolar(), hprdh(), hprdhj(), hreal(), hrect(), htoc(), polyroot(), rft(), unwind()
Confidence Intervals	invchi(), invF(), invnor(), invstu(), invstudrng(), regcoefs(), regresshelp(), secoefs(), t2int(), tint()
Control	arginfo_fun, batch(), break, breakall, breakif(), callback_fun, customize, else, elseif, evaluate(), for, getoptions(), if, interrupt, loadUser(), macro(), macro_syntax, macros, next, options, printoptions(), redo(), return, setoptions(), shell(), syntax, user_fun, while
Descriptive Statistics	boxplot(), cellstats(), cor(), describe(), descriptive(), halfnorm(), hist(), lowess(), max(), min(), propinterval(), proptest(), rankits(), stemleaf(), sum(), t2int(), t2val(), tabs(), tint(), tinterval(), ttest(), tval(), twotailt(), vboxplot(), zinterval(), ztest()
Files	addhelpfile(), adddatapath(), addmacrofile(), asciisave(), batch(), console(), data_files, DATAPATHS, file_names, files, findfile(), fprint(), fwrite(), getdata(), getfilename(), gethelp(), getmacros(), graph_border, graph_files, inforead(), launching, loadUser(), macro_files, macroread(), macrowrite(), matprint(), matread(), matread_file, matwrite(), read(), readcols(), readdata(), restore(), restorenames(), save(), spool(), user_fun, vecread(), vecread_file, vecread_keys, write()
GLM	anova(), anovapred(), bcprd(), bit_ops, coefs(), contrast(), design, designhelp(), factor(), fastanova(), glm, glm_keys, glmfit(), glmpred(), glmtable(), ipf(), isfactor(), logistic(), makefactor(), manova(), modelinfo(), models, modelvars(), nbits(), poisson(), popmodel(), power(), power2(), predtable(), probit(), pushmodel(), regcoefs(), regpred(), regress(), regresshelp(), robust(), samplesize(), screen(), secoefs(), swp(), varnames(), wtanova(), wtmanova(), wtregress(), xrows(), xvariables(), yates()

Continued from previous page

Search Key	References
General	addhelpfile(), appendnotes(), arginfo_fun, arimahelp(), asciisave(), attachnotes(), callback_fun, carapace, copyright, customize, designhelp(), dos_windows, edit(), equal(), evaluate(), gethelp(), gethistory(), getlabels(), getnotes(), gettime(), getusage(), goodfactors(), graphicshelp(), haslabels(), hasnotes(), help(), interrupt, isarray(), ischar(), isdefined(), isfactor(), isfunction(), isgraph(), islocked(), islogic(), ismacro(), ismatrix(), ismissing(), isname(), isnull(), isnumber(), isreal(), isscalar(), isstruc(), isvector(), labels, launching, list(), listbrief(), loadUser(), locks, lockvars(), macintosh, mac_classic, macrousage(), mathhelp(), memory, memoryinfo(), more(), mulvarhelp(), notes, options, primefactors(), printoptions(), quitting, regresshelp(), rename(), restore(), restorenames(), save(), sethistory(), setlabels(), setodometer(), shell(), syntax, tserhelp(), unix, unlockvars(), usage(), userfunhelp(), user_fun, workspace
Input	adddatapath(), CLIPBOARD, clipreaddata, console(), data_files, DATAPATHS, enter(), enterchars(), file_names, files, fromclip(), getdata(), getfilename(), getmacros(), inforead(), macro_files, macroread(), matread(), matread_file, read(), readcols(), readdata(), vecread(), vecread_file, vecread_keys
LOGICAL Variables	alltrue(), anytrue(), islogic(), logic, syntax, variables
NULL Variables	anymissing(), cat(), dim(), hconcat(), ismissing(), isnull(), length(), ndims(), NULL, save(), syntax, variables, vconcat(), vector()
Macros	addmacrofile(), appendnotes(), argvalue(), attachnotes(), evaluate(), getkeywords(), getmacros(), isarray(), ischar(), isdefined(), isfactor(), isfunction(), isgraph(), islogic(), ismacro(), ismatrix(), ismissing(), isname(), isnull(), isnumber(), isreal(), isscalar(), isstruc(), isvector(), keyvalue(), macro(), macro_files, macro_syntax, macroread(), macros, macrousage(), macrowrite(), notes, read(), return, setodometer()

Continued from previous page

Search Key	References
Matrix Algebra	bcprd(), cholesky(), det(), diag(), dmat(), eigen(), eigenvals(), mathhelp(), matrices, matrix(), outer(), qr(), releigen(), releigenvals(), svd(), swp(), t(), toeplitz(), trace(), transpose(), trideigen(), trilower(), triunpack(), triupper()
Missing Values	anymissing(), arithmetic, bit_ops, files, ismissing(), logic, matprint(), matread(), number, options, paste(), print(), read(), setoptions(), syntax
Multivariate Analysis	cluster(), glm, glm_keys, kmeans(), manova(), mulvarhelp(), popmodel(), pushmodel(), rotation()
Operations	arithmetic, bit_ops, logic, matrices, nbits(), precedence, t(), transpose()
Ordering	grade(), halfnorm(), match(), rank(), rankits(), sort(), unique()
Output	asciisave(), CLIPBOARD, clipwritedat(), data_files, DATAPATHS, error(), file_names, files, fprint(), formatpval(), fwrite(), getfilename(), graph_border, graph_files, labels, macro_files, macrowrite(), matprint(), matread_file, matwrite(), more(), options, paste(), print(), putascii(), setoptions(), spool(), toclip(), vecread_file, vecread_keys, write(), writedata()
Plotting	addchars(), addlines(), addpoints(), addstrings(), boxplot(), chplot(), colplot(), graphicshelp(), graphs, GRAPHWINDOWS, graph_assign, graph_border, graph_files, graph_keys, graph_ticks, hist(), lineplot(), lowess(), makesymbols(), Mouse(), plot(), rowplot(), showplot(), stemleaf(), stringplot(), tek(), tekx(), vboxplot(), vt(), vtx()
Probabilities	cumbeta(), cumbin(), cumchi(), cumdunnett(), cumF(), cumgamma(), cumnor(), cumpoi(), cumstu(), cumstudrng(), invbeta(), invchi(), invdunnett(), invF(), invgamma(), invnor(), invstu(), invstudrng(), power(), power2(), propinterval(), proptest(), samplesize(), t2int(), t2val(), tint(), tinterval(), ttest(), tval(), twotailt(), zinterval(), ztest()
Random Numbers	getseeds(), invbeta(), invchi(), invF(), invgamma(), invstu(), options, rbin(), rnorm(), rpoi(), rsample(), runi(), setoptions(), setseeds()

Continued from previous page

Search Key	References
Regression	coefs(), design, designhelp(), glm, glm_keys, glmfit(), glmpred(), logistic(), lowess(), models, poisson(), popmodel(), power2(), probit(), pushmodel(), regcoefs(), regpred(), regress(), regresshelp(), robust(), screen(), secoefs(), wtregress()
Residuals	popmodel(), pushmodel()
Structures	changestr(), compnames(), isstruc(), makestr(), ncomps(), restorenames(), split(), strconcat(), structure(), structures
Syntax	alltrue(), anytrue(), argvalue(), arithmetic, assignment, batch(), break, breakall, breakif(), CLIPBOARD, comments, else, elseif, evaluate(), for, getkeywords(), GRAPHWINDOWS, graph_assign, if, interrupt, keyvalue(), keywords, logic, macro(), macro_syntax, macros, next, number, precedence, return, scalars, structures, subscripts, syntax, variables, vectors, while
Time Series	arimahelp(), autoreg(), cconj(), cdivc(), cdivcj(), cft(), cimag(), cmplx(), complex, convolve(), cpolar(), cprdc(), cprdcj(), creal(), crect(), ctoh(), hconj(), hdivh(), hdivhj(), hft(), himag(), hpolar(), hprdh(), hprdhj(), hreal(), hrect(), htoc(), movavg(), padto(), partacf(), polyroot(), reverse(), rft(), rotate(), time_series, trideigen(), tserhelp(), unwind(), yulewalker()
Transformations	abs(), acos(), asin(), asLong(), atan(), atanh(), boxcox(), ceiling(), cos(), cosh(), digamma(), exp(), floor(), halfnorm(), hypot(), lgamma(), log(), log10(), log2(), nbits(), polygamma(), rankits(), rational(), round(), sin(), sinh(), sqrt(), tan(), tanh(), transformations

Continued from previous page

Search Key	References
Variables	appendnotes(), array(), arrays, asLong(), attachnotes(), cat(), data_files, delete(), diag(), dim(), dmat(), equal(), getdata(), getlabels(), getnotes(), haslabels(), hasnotes(), hconcat(), isarray(), ischar(), isdefined(), isfactor(), isfunction(), isgraph(), islocked(), islogic(), ismacro(), ismatrix(), ismissing(), isname(), isnull(), isnumber(), isreal(), isscalar(), isvector(), labels, length(), locks, lockvars(), logic, match(), matread_file, matrices, matrix(), nameof(), ncols(), ncomps(), ndims(), notes, nrows(), NULL, number, rename(), rep(), run(), save(), scalars, select(), setlabels(), shapeof(), structures, syntax, trilower(), triunpack(), triupper(), typeof(), unique(), unlockvars(), variables, varnames(), vconcat(), vecread_file, vector(), vectors

Table 12.2: Search Keys and References from Arima.mac.txt

Search Key	References
ARIMA models	acfarma(), arima(), arimares(), ARSIGN, detarma(), hannriss(), innovations(), innovest(), MASIGN, moveoutroots(), neg2logLarma()
Autocovariance	acfarma()
Complex numbers	
Frequency domain	specarma()
General	arimahelp()
Nonlinear fitting	arima(), arimares()
Preliminary estimation	hannriss(), innovations(), innovest()
Spectrum analysis	specarma()
Time domain	acfarma(), arima(), arimares(), hannriss(), innovations(), innovest(), moveoutroots(), neg2logLarma(), rhatcovar(), rhatvar()

Table 12.3: Search Keys and References from Design.hlp.txt

Search Key	References
Aliasing	aberration2(), aliases2(), aliases3(), allaliases2(), choosegen2(), doff2(), ffdesign2()
Analysis	boxcoxvec(), ems(), interblock(), mixed(), pairwise(), quadmax(), randsign(), randt2(), randt(), reml(), rscanon(), varcomp(), yatesplot()
ANOVA	all3anova(), all4anova(), boxcoxvec(), buildfactor(), ems(), interblock(), mixed(), pairwise(), varcomp()
Confounding	choosedef2(), confound2(), confound3(), doconfound2()
Design	aberration2(), aliases2(), aliases3(), allaliases2(), choosedef2(), choosegen2(), confound2(), confound3(), doconfound2(), doff2(), ffdesign2(), findncp(), findpower(), findsampsize()
Factorial	aberration2(), aliases2(), aliases3(), allaliases2(), buildfactor(), choosedef2(), choosegen2(), confound2(), confound3(), doconfound2(), doff2(), ems(), ffdesign2(), interactplot(), mixed(), stdordlabels(), varcomp(), yatesplot()
Permutation test	randsign(), randt2(), randt()
Plots	interactplot(), sidebyside(), yatesplot()
Random effects	mixed(), varcomp()

Table 12.4: Search Keys and References from Graphics.mac.txt

Search Key	References
Bar graphs	bargraph(), hist(), panelhist()
Contour graphs	contour(), contourplot(), findcontour()
Distribution graphs	boxplot5num(), hist(), piechart(), sampcdf(), vboxplot()
General	graphicshelp(), news
Interaction graphs	colplot(), rowplot()
Line graphs	colplot(), ellipse(), plotpanes(), rowplot(), sampcdf()
Multivariate graphs	plotmatrix()

Continued from previous page

Search Key	References
Panel graphs	panelhist(), panelplot(), panel_graphs, plotmatrix(), plotpanes()
Residual graphs	plotresids()
Shapes	ellipse()

Table 12.5: Search Keys and References from Regress.mac.txt

Search Key	References
ANOVA	anovapred(), regcoefs(), resid(), resvsindex(), resvsrankits(), resvsyhat(), yhat()
Confidence limits	anovapred(), betalimits(), estimlimits(), regcoefs()
General	regresshelp()
GLM	anovapred(), regcoefs(), regs(), resid(), resvsindex(), resvsrankits(), resvsyhat(), yhat()
Hypothesis test	testbeta(), testestim()
Nonlinear fitting	nlreg()
Plotting	resvsindex(), resvsrankits(), resvsyhat()
Prediction limits	anovapred(), predlimits()
Regression	betalimits(), entervar(), estimlimits(), nlreg(), predlimits(), regcoefs(), regs(), removevar(), resid(), resvsindex(), resvsrankits(), resvsyhat(), steplook(), stepsetup(), stepstatus(), testbeta(), testestim(), yhat()
Residuals	resid(), resvsindex(), resvsrankits(), resvsyhat()
Standard error	anovapred(), regcoefs()
Stepwise regression	entervar(), removevar(), steplook(), stepsetup(), stepstatus()

Table 12.6: Search Keys and References from Tser.hlp.txt

Search Key	References
ARIMA models	arspectrum()
Autocorrelation	autocor(), crosscor(), crosscov()

Continued from previous page

Search Key	References
Autocovariance	autocov(), crosscor(), crosscov()
Complex numbers	complex_data, complex_fun, ffplot(), fourier, hermitian
Fourier transforms	complex_fun, fourier, hermitian, testnfreq()
Frequency domain	arspectrum(), bandwidth, burg(), compfa(), complex_data, complex_fun, compza(), costaper(), crsspectrum(), dpss(), ffplot(), fourier, hermitian, multitaper(), spectrum()
General	gettsmacros()
Plotting	ffplot(), tsplot()
Spectrum analysis	arspectrum(), bandwidth, burg(), compfa(), compza(), costaper(), crsspectrum(), dpss(), multitaper(), spectrum()
Time domain	autocor(), autocov(), burg(), costaper(), crosscor(), crosscov(), detrend(), tsplot()

Table 12.7: Search Keys and References from Gui.hlp.txt

Search Key	References
dialogs	alert()
xml	getmenubar(), setmenubar()

Table 12.8: Search Keys and References from Userfun.hlp.txt

Search Key	References
Coding	arginfo_fun, c_macros, callback_fun, type_codes, user_fun
Compiling	compile_dos, compile_mac, compile_unix, compile_win
Executing	User
Loading	loadUser
Sample source	arginfo_fun, c_macros, callback_fun, user_fun
User functions	arginfo_fun, c_macros, callback_fun, compile_dos, compile_mac, compile_unix, compile_win, loadUser, type_codes, User, user_fun