The numodel package*

Paul Zuurbier
mail@paulzuurbier.nl

November 5, 2026

Abstract

A LuaLaTeX package for writing and rendering numerical models (Euler-integrated dynamical systems) directly inside LaTeX documents, aimed at physics teaching material. It provides a text-model pipeline (\mvar, \mrule, \computemodel), Forrester stock-and-flow diagrams, and optional plots of the computed time series via the sibling package numodel-plot.

Contents

1 Introduction **2**

2 First example: a free-falling ball **2**
2.1 \textmodel — rule table . . . 2
2.2 \graphicmodel — Forrester diagram . . . 3
2.2.1 How the elements are detected . . . 3
2.2.2 How the placement algorithm works . . . 4
2.2.3 Controlling the layout with the order of the \mvar lines . . . 6
2.3 \computemodel + \diagrammodel — numerical plot . . . 8

3 Configuration **9**
3.1 Diagram styles in practice . . . 11

4 Public API **12**
4.1 Variables and rules . . . 12
4.2 Expression reference . . . 13
4.3 Render commands . . . 15
4.4 Series accessors . . . 18
4.5 Namespace management . . . 20
4.6 Shared valves . . . 21

5 Variable types **21**

6 Generated accessors **21**

7 Exercises: pupil's and answer version **22**
7.1 Other replacements: the alias keys . . . 23

8 Multiple models in one document **24**
8.1 Changing and copying models . . . 24

9 Requirements **25**

*This document corresponds to numodel v0.13.0, dated November 5, 2026.

1 Introduction

numodel lets an author write a dynamical system (stocks, flows, helper variables, rules, and a stop condition) as a sequence of LaTeX macros and renders three complementary views of that model directly in the document:

- a *text model* — a typeset rule table with initial values (`\textmodel`);
- a *graphic model* — a Forrester stock-and-flow diagram with auto-layout (`\graphicmodel`);
- a *diagram* — a numerical Euler simulation plus PGFPlot of any pair of variables (`\computemodel` followed by `\diagrammodel`; the plot is rendered through the sibling package numodel-plot).

All three views are produced from a single set of declarations so the textbook description, the conceptual stock-and-flow diagram, and the numerical result of the same model are guaranteed to stay in sync. Variables and rules live in namespaces (*prefixes*) so a document can contain multiple independent models; `\newmodelprefix{P}` starts a fresh one. The model is computed in Lua (or, on request, with l3fp; Section 4.3), which also keeps the computed series; the rendering layer is expl3.

2 First example: a free-falling ball

A ball dropped from $h_0 = 100\,\mathrm{m}$ under constant gravitational acceleration. The complete model:

```
\usepackage[syntax=EN]{numodel}

\newmodelprefix{ball}
\mvar{T}{t}{0}{\s}{2}{system}
\mvar{Dt}{dt}{0.1}{\s}{2}{system}
\mvar{V}{v}{0}{\m\per\s}{2}{stock}
\mvar{Y}{y}{100}{\m}{3}{stock}
\mvar{G}{g}{-9.81}{\m\per\s\squared}{3}{aux}

\mrule{V}{\ballV + \ballG * \ballDt}
\mrule{Y}{\ballY + \ballV * \ballDt}
\mrule{T}{\ballT + \ballDt}
\mstop{\ballY <= 0}
```

After the declarations above, three render commands produce the three views shown below, each from the *same* model.

2.1 `\textmodel` — rule table

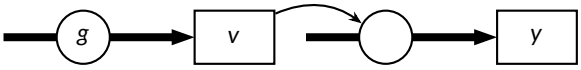
The verbatim source rendered by `\textmodel`:

	model	initial values
1	$v = v + g \cdot dt$	$t = 0\,\mathrm{s}$
2	$y = y + v \cdot dt$	$dt = 0.10\,\mathrm{s}$
3	$t = t + dt$	$v = 0\,\mathrm{m\,s^{-1}}$
4	IF $y \leqslant 0$ THEN STOP ENDIF	$y = 100\,\mathrm{m}$
		$g = -9.81\,\mathrm{m\,s^{-2}}$

Each `\mvar` with a non-empty start value contributes a row in the *initial values* column; each `\mrule` contributes a row in the *model* column. Symbols come from the second `\mvar` argument (the display text), values are formatted through siunitx. `<=` is rendered as $\leqslant$.

2.2 `\graphicmodel` — Forrester diagram

`\graphicmodel` draws the same model as a stock-and-flow diagram. You do not describe the drawing: the rules determine *which* elements the diagram has, and the order of the `\mvar` lines determines *where* they go. For the falling ball:



The velocity v grows through a flow with the rate g, which labels the valve. The height y changes with a rate that is a stock itself, so that valve has no name and a causal arrow comes from v. Section 2.2.1 shows how the elements are found in the rules, Section 2.2.2 how they are placed, and Section 2.2.3 how you steer the result.

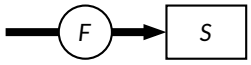
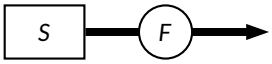
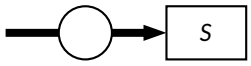
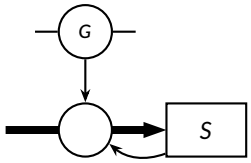
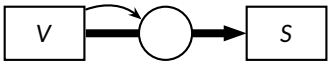
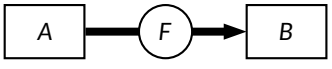
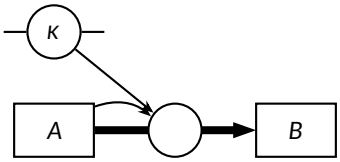
2.2.1 How the elements are detected

Nodes. Every `\mvar` gives one node, with a shape that follows from its type: a rectangle for a `stock`, a circle for an `aux` and a circle with two short horizontal dashes for a `constant`. A `system` variable is not drawn.

Flows. Flows are read from the rules of stocks only. Each rule is split into additive terms at the top-level `+` and `-`:

- the term that carries the stock over, `+\S`, is dropped;
- every other term with a `+` is a candidate inflow, every term with a `-` a candidate outflow;
- a difference in parentheses is split, so `(\A - \B)*\Dt` is an inflow and an outflow; a sum in parentheses, `(\A + \B)*\Dt`, stays one flow;
- a rule with `?:` contributes the terms of all its branches; a term that occurs with both signs counts as an inflow.

What a term becomes depends on the variables in it. System variables do not count, Δt among them; the stock itself does.

Terms	Result (default style `tight`)	Diagram
`+\F*\Dt`	One variable, not a stock: a valve named after *F*.	
`-\F*\Dt`	The same, as an outflow.	
`+0.5*\Dt`	No variable: a valve without a name.	
`+\G*\S*\Dt`	Several variables: a valve without a name and an arrow from each, here from the stock to its own valve (the feedback loop).	
`+\V*\Dt`	One variable that is a stock: a valve without a name and an arrow from that stock.	
`-\F*\Dt` in the rule of *A*, `+\F*\Dt` in the rule of *B*	The same term in two rules: one between-flow.	
`-\K*\A*\Dt` in the rule of *A*, `+\K*\A*\Dt` in the rule of *B*	The same, with several variables: a between-flow without a name, with an arrow from each.	

Between-flows. An outflow of one stock and an inflow of another become a single valve when the two terms are equal apart from spaces. Factors count: `2*\P*\Dt` pairs with `2*\P*\Dt`, not with `3*\P*\Dt`. Terms are compared whole, so `(\A - \B)*\K*\Dt` in two rules is one flow. The valve is named only when the term has one variable that is not a stock. The valve of `\K*\A*\Dt` stays empty, as a valve for such a term always does: a formula is not a name. To name the flow, give it a variable of its own: `\mrule{Q}{\K*\A}`, then `-\Q*\Dt` and `+\Q*\Dt`. A variable that feeds more than one stock gets a shared valve; see Section 4.6. When the variable already labels an inflow valve and the outflow term differs from it, as with `+3*\P*\Dt` and `-2*\P*\Dt`, the outflow gets a valve without a name and an arrow from *P*.

One valve per side. A stock has at most one inflow and one outflow valve. When a rule has more terms on one side, the last of the named flows is kept and the others are left out, so `\S + \A*\Dt + \B*\Dt` shows only *B*. In the style `tight` the variable *A* then disappears from the diagram altogether. Write `(\A + \B)*\Dt` for one valve with an arrow from each, or add an auxiliary for the total.

Causal arrows. Each variable used in a rule gets an arrow to the variable that the rule computes. There is no arrow from a system variable, and none from a flow variable to the stock it feeds or drains: there the valve shows the link. Arrows to a valve without a name come from the term itself, as in the table.

The diagram style. In the default style `tight` the variable of a named valve *is* the valve: the auxiliary or constant is not drawn as a separate node, and any arrows to it end at the valve. In the styles `forrester` and `edu` the variable stays a node on its own row, with a causal arrow to the valve. This matters for the placement below. Section 3 describes the `diagram-style` key, and the key `flowarrow-cloud-tip` for a cloud at the open end of a flow.

2.2.2 How the placement algorithm works

The elements sit in the cells of a grid. Columns are counted from the left, rows from the bottom, both from 0. The cell size is set by `graphscalex` and `graphscaley`. Stocks and valves fill row 0, auxiliaries row 1, constants row 2. The layout is computed afresh for every `\graphicmodel`, in these steps:

1. *Classify.* The flows of Section 2.2.1 are determined first, because they decide which variables need a place on the upper rows. In the style `tight` a valve variable does not; in the other styles it does.

2. *Count.* The algorithm counts c, the constants on their own row, and a, the auxiliaries on their own row.

3. *Stocks.* They are placed first, on row 0, in the order of their `\mvar` lines. A stock that is not yet placed brings along all stocks it is connected to by between-flows, and that *chain* is placed as one block in the direction of the flow:

   inflow valve, stock, between valve, stock, …, outflow valve

   A valve always stands directly left or right of its stock, so flows run from left to right. When a block starting with an inflow valve follows a block ending in an outflow valve, one column stays empty: two adjacent valves would read as a between-flow. With w the width of the whole row, it starts in column $\max(c, w) - w$, so a stock row narrower than the constants ends under the last constant.

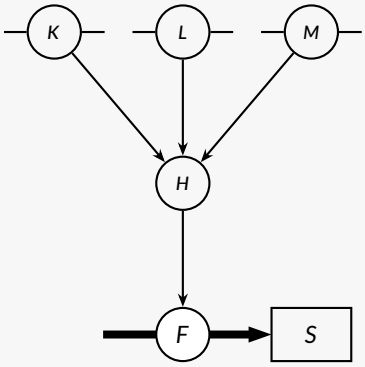
4. *Auxiliaries.* They go on row 1 in the order of their `\mvar` lines, skipping cells that are taken. If there are more constants than auxiliaries, the auxiliaries are centred under the constants: the first one goes in column $\lfloor (c - a + 1)/2 \rfloor$, and otherwise in column 0.

5. *Constants.* They go on row 2 from column 0, in the order of their `\mvar` lines. When row 1 has no auxiliaries and no return valve (see below) they go on row 1 instead, so that no row stays empty.

6. *Arrows.* Causal arrows are straight. An arrow bends when the centre of another element lies exactly on the line, and an arrow from a stock to a valve on its own row bends because it would otherwise run along the flow. All other crossings stay, and those are what the order of the `\mvar` lines can remove.

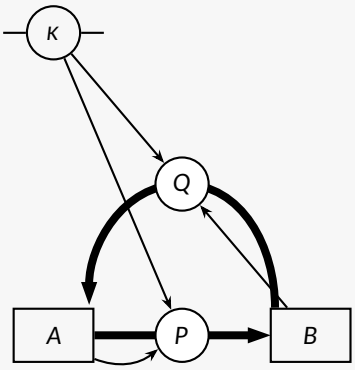
The next example shows the alignment; from here on the diagrams use `graphscalex=1.7` to fit next to their code. There are three constants and one auxiliary, so $c = 3$ and $a = 1$, and the auxiliary starts in column $\lfloor (3 - 1 + 1)/2 \rfloor = 1$. The stock row has a valve and a stock, so $w = 2$ and the row starts in column $3 - 2 = 1$.

```
\newmodelprefix{ala}
\mvar{Dt}{dt}{1}{}{2}{system}
\mvar{S}{S}{0}{}{2}{stock}
\mvar{F}{F}{1}{}{2}{aux}
\mvar{H}{H}{1}{}{2}{aux}
\mvar{K}{K}{1}{}{2}{constant}
\mvar{L}{L}{1}{}{2}{constant}
\mvar{M}{M}{1}{}{2}{constant}
\mrule{H}{\alaK + \alaL + \alaM}
\mrule{F}{2 * \alaH}
\mrule{S}{\alaS + \alaF * \alaDt}
\graphicmodel
```



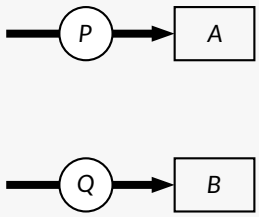
Closed loops. Between-flows can form a closed loop, as with two tanks that exchange water. The first declared stock of the loop starts the chain, and the flow from the last stock back to the first is the *return flow*. Its valve goes on the row above the chain, in the middle cell, and the pipe curves through it from one inner top corner to the other. The cells between the two outer stocks are taken; those above the outer stocks stay free for an auxiliary, which fits next to the pipe. Row 1 is not empty now, so constants go on row 2. Declaring *B* first would swap the stocks and make *P* the return flow.

```
\newmodelprefix{loo}
\mvar{Dt}{dt}{1}{}{2}{system}
\mvar{A}{A}{9}{}{2}{stock}
\mvar{B}{B}{0}{}{2}{stock}
\mvar{P}{P}{}{}{2}{aux}
\mvar{Q}{Q}{}{}{2}{aux}
\mvar{K}{K}{1}{}{2}{constant}
\mrule{P}{\looK * \looA}
\mrule{Q}{\looK * \looB}
\mrule{A}{\looA - \looP*\looDt + \looQ*\
    looDt}
\mrule{B}{\looB + \looP*\looDt - \looQ*\
    looDt}
\graphicmodel
```



Limiting the width. With `\numodelsetup{gridmaxx=N}` a row holds at most `N` columns. When the next element does not fit, everything already placed on that row and above moves up one row, and filling continues in column 0 of the original row. What you declare first therefore ends up on top, as in text. The alignment of steps 3 and 4 is switched off, and a chain is never split. A chain with a return flow starts a new row, with a free row above it for the valve.

```
\newmodelprefix{wra}
\mvar{Dt}{dt}{1}{}{2}{system}
\mvar{A}{A}{0}{}{2}{stock}
\mvar{B}{B}{0}{}{2}{stock}
\mvar{P}{P}{1}{}{2}{aux}
\mvar{Q}{Q}{1}{}{2}{aux}
\mrule{A}{\wraA + \wraP * \wraDt}
\mrule{B}{\wraB + \wraQ * \wraDt}
\numodelsetup{gridmaxx=3}
\graphicmodel
\numodelsetup{gridmaxx=0}
```



Fixed positions. With `\mvar[gridx=...,gridy=...]` a variable stays in that cell. With only `gridx` it goes on the row of its type; `gridy` alone has no effect. Automatic elements skip the occupied cell, a fixed variable does not count in *c* or *a*, and with `gridmaxx` rows move past it. The valves of a fixed stock go in the columns next to it, on its own row; for a stock in column 0 the whole diagram moves one column to the right.

2.2.3 Controlling the layout with the order of the `\mvar` lines

The order of the `\mvar` lines is the main thing you control, and it is safe to use: it does not change the computed results. For the `\mrule` lines it is the other way round: their order decides the calculation, but the diagram is the same in every order. Two side effects remain. The `\mvar` order is also the order of the *initial values* column of `\textmodel`, and a start value given as an expression can only use variables declared before it.

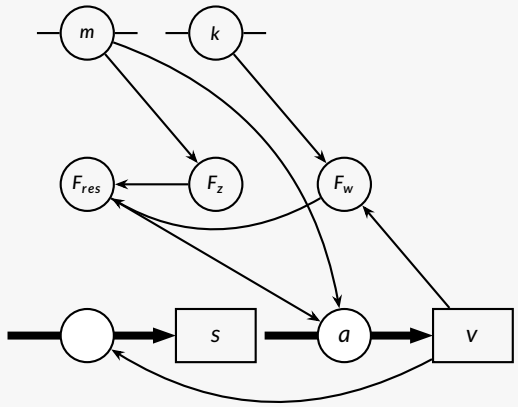
Only the order *within* a type counts:

Change	Effect in the diagram
Swap two `stock` lines	The blocks on row 0 change places.
Swap two `aux` lines	They change places on row 1, unless one is a valve in the style `tight`.
Swap two `constant` lines	They change places on the constants row, unless one is a valve in the style `tight`.
Move an `aux` past a `stock` or `constant`	Nothing.
Move a valve variable in the style `tight`	Nothing: the valve stays next to its stock.
Swap two stocks of one chain	Nothing inside the chain: it follows the flow. The chain stands where its first declared stock would stand.
Change the order of the terms in one rule	Which named flow keeps the valve, if a side has more than one.
Swap the stocks of a shared valve	An inflow valve goes before the first declared stock, an outflow valve after the last.
Declare another stock of a closed loop first	That stock starts the chain, which decides which flow is the return flow.

A worked example. A falling object with air resistance has two stocks and a chain of auxiliaries, which makes the order visible. A first version, with the variables in the order in which one might write them down. Time and the stop condition are left out here and below: they play no part in the diagram.

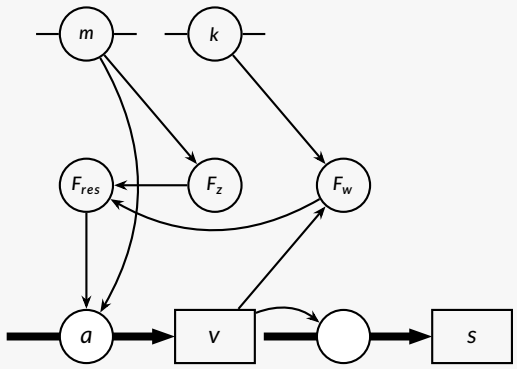
```
\newmodelprefix{dra}
\mvar{Dt}{\Delta t}{0.01}{\s}{2}{system}
\mvar{S}{s}{0}{\m}{3}{stock}
\mvar{V}{v}{0}{\m\per\s}{3}{stock}
\mvar{A}{a}{}{\m\per\s\squared}{3}{aux}
\mvar{Fres}{F_{res}}{}{\newton}{3}{aux}
\mvar{Fz}{F_z}{}{\newton}{3}{aux}
\mvar{Fw}{F_w}{}{\newton}{3}{aux}
```

```
\mvar{M}{m}{0.5}{\kg}{2}{constant}
\mvar{K}{k}{0.02}{\kg\per\m}{2}{constant}
\mrule{Fz}{\draM * 9.81}
\mrule{Fw}{\draK * \draV * \draV}
\mrule{Fres}{\draFz - \draFw}
\mrule{A}{\draFres / \draM}
\mrule{V}{\draV + \draA * \draDt}
\mrule{S}{\draS + \draV * \draDt}
\graphicmodel
```



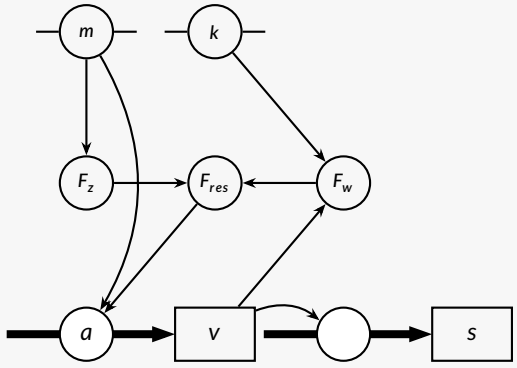
The calculation runs from the velocity to the distance, but *s* stands left of *v*, so the arrow from *v* to the valve of *s* points backwards. Swapping the two `stock` lines fixes that. The other lines stay as they are, so row 1 and row 2 do not change. (`\dragrules` stands for the six `\mrule` lines above, for the given prefix.)

```
\newmodelprefix{drb}
\mvar{Dt}{\Delta t}{0.01}{\s}{2}{system}
\mvar{V}{v}{0}{\m\per\s}{3}{stock}
\mvar{S}{s}{0}{\m}{3}{stock}
\mvar{A}{a}{}{\m\per\s\squared}{3}{aux}
\mvar{Fres}{F_{res}}{}{\newton}{3}{aux}
\mvar{Fz}{F_z}{}{\newton}{3}{aux}
\mvar{Fw}{F_w}{}{\newton}{3}{aux}
\mvar{M}{m}{0.5}{\kg}{2}{constant}
\mvar{K}{k}{0.02}{\kg\per\m}{2}{constant}
\dragrules{drb}
\graphicmodel
```



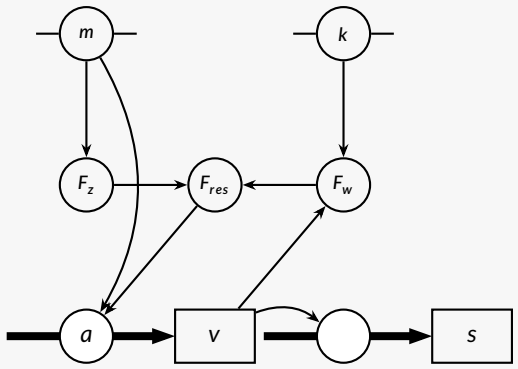
On row 1, F_{res} stands left of both its inputs, so the arrow from F_w has to bend over F_z. Putting F_{res} between F_z and F_w gives two straight arrows. The line of *a* can stay anywhere: in the style `tight`, *a* is the valve of *v*.

```
\newmodelprefix{drc}
\mvar{Dt}{\Delta t}{0.01}{\s}{2}{system}
\mvar{V}{v}{0}{\m\per\s}{3}{stock}
\mvar{S}{s}{0}{\m}{3}{stock}
\mvar{A}{a}{}{\m\per\s\squared}{3}{aux}
\mvar{Fz}{F_z}{}{\newton}{3}{aux}
\mvar{Fres}{F_{res}}{}{\newton}{3}{aux}
\mvar{Fw}{F_w}{}{\newton}{3}{aux}
\mvar{M}{m}{0.5}{\kg}{2}{constant}
\mvar{K}{k}{0.02}{\kg\per\m}{2}{constant}
\dragrules{drc}
\graphicmodel
```



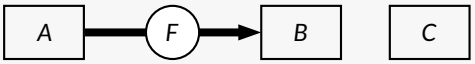
Order alone cannot leave a gap. Here k stands in column 1 and is only needed by F_w in column 2. This is where one fixed position helps; the rest stays automatic.

```
\newmodelprefix{drd}
\mvar{Dt}{\Delta t}{0.01}{\s}{2}{system}
\mvar{V}{v}{0}{\m\per\s}{3}{stock}
\mvar{S}{s}{0}{\m}{3}{stock}
\mvar{A}{a}{}{\m\per\s\squared}{3}{aux}
\mvar{Fz}{F_z}{}{\newton}{3}{aux}
\mvar{Fres}{F_{res}}{}{\newton}{3}{aux}
\mvar{Fw}{F_w}{}{\newton}{3}{aux}
\mvar{M}{m}{0.5}{\kg}{2}{constant}
\mvar[gridx=2, gridy=2]{K}{k}{0.02}{\kg\
    per\m}{2}{constant}
\dragrules{drd}
\graphicmodel
```



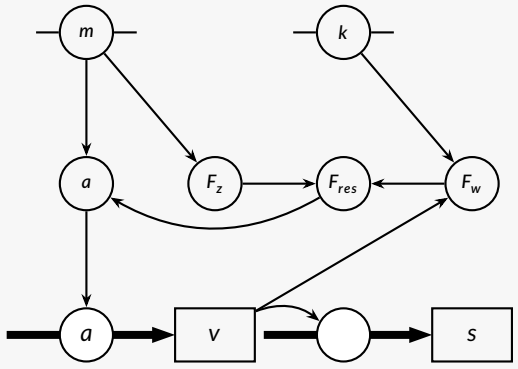
Chains follow the flow. Within a chain the order of the `\mvar` lines has no effect. Here B is declared first, but the flow runs from A to B, so A stands on the left. The chain as a whole takes the place of its first declared stock, so it comes before C.

```
\newmodelprefix{cha}
\mvar{Dt}{dt}{1}{}{2}{system}
\mvar{B}{B}{0}{}{2}{stock}
\mvar{C}{C}{0}{}{2}{stock}
\mvar{A}{A}{9}{}{2}{stock}
\mvar{F}{F}{1}{}{2}{aux}
\mrule{A}{\chaA - \chaF * \chaDt}
\mrule{B}{\chaB + \chaF * \chaDt}
\graphicmodel
```



The style changes what the order does. In the styles `forrester` and `edu` a valve variable needs a place on its row, so its `\mvar` line now counts. Below is the last worked example again, with the style `edu`: a is the first auxiliary and stands above its valve, but the arrow from F_{res} to a has to bend over F_z. In the style `tight` that arrow went straight to the valve. Choose the order for the style you use.

```
\newmodelprefix{dre}
\mvar{Dt}{\Delta t}{0.01}{\s}{2}{system}
\mvar{V}{v}{0}{\m\per\s}{3}{stock}
\mvar{S}{s}{0}{\m}{3}{stock}
\mvar{A}{a}{}{\m\per\s\squared}{3}{aux}
\mvar{Fz}{F_z}{}{\newton}{3}{aux}
\mvar{Fres}{F_{res}}{}{\newton}{3}{aux}
\mvar{Fw}{F_w}{}{\newton}{3}{aux}
\mvar{M}{m}{0.5}{\kg}{2}{constant}
\mvar[gridx=2, gridy=2]{K}{k}{0.02}{\kg\
    per\m}{2}{constant}
\dragrules{dre}
\graphicmodel[diagram-style=edu]
```



2.3 `\computemodel` + `\diagrammodel` — numerical plot

`\computemodel` iterates the rules forward in time using Euler integration with step size `\ballDt`, stopping when `\mstop`'s condition becomes true (or when the `maxiter` safety limit is reached, see Section 3). `\diagrammodel{xvar}{yvar}{label}` then plots one variable against another:

```
\computemodel
\diagrammodel{T}{Y}{ball-fall}
```

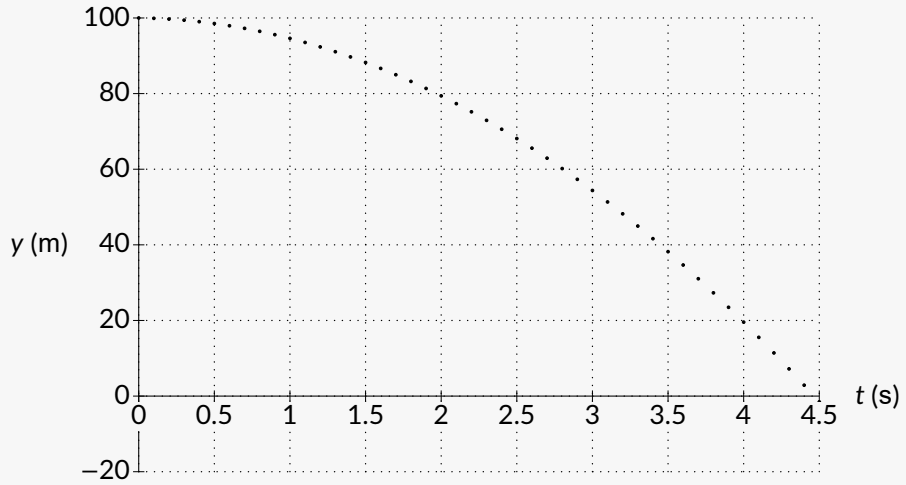


Figure 1: $y(t)$-diagram

Axis ranges, tick lattice, and labels are computed automatically from the simulated min/max of each variable; the plot inherits the numodel-plot style (see that package's documentation for configuration).

3 Configuration

`\numodelsetup` Runtime configuration:

```
\numodelsetup{syntax=NL, maxiter=50000}
```

The same keys can also be passed as package options: `\usepackage[syntax=NL]{numodel}`. Recognised keys:

`syntax` Language tag for the rule-table rendering. Built-in values:

- **`EN`** (default) XMILE-style ALL-CAPS keywords: `IF`/`THEN`/`ELSE`, `AND`, `OR`, `ABS`, `SIGN`, …
- **`NL`** Dutch CoachTaal keywords: `Als`/`Dan`/`Anders`, `EN`, `OF`, `Abs`, `Teken`, …
- **`FPEVAL`** (aliases `fpeval`, `l3fp`) shows each rule body verbatim in l3fp (`\fpeval`) syntax, exactly as typed in `\mrule` – lowercase function names, literal `*`, `^`, `&&`, `?:` – typeset monospace. Variable macros still resolve to their display names; the ternary is *not* expanded into `IF`/`THEN`/`ELSE` and the stop row reads `stop:` ⟨*cond*⟩. Decimal mark defaults to `point`.

Each language tag X corresponds to a file `numodel-X.def` located via `kpse` when the package processes the key. The package ships with `numodel-EN.def`, `numodel-NL.def` and `numodel-FPEVAL.def`; drop your own `numodel-FR.def` (or any other tag) in `TEXMFHOME/tex/latex/numodel/` and select it with `\usepackage[syntax=FR]{numodel}` – no package rebuild needed. The setting affects display only; the expression syntax in `\mrule` bodies is always `\fp_eval`-compatible.

`maxiter` Safety limit on the number of `\computemodel` iterations (default 20 000). When reached, the simulation stops with a warning to check the stop condition; the series up to that point are kept. numodel-coach uses it as Coach's number of iterations.

`graphscalex` Horizontal grid spacing in centimetres for `\graphicmodel`'s Forrester layout (default 2). Larger values spread the diagram out horizontally.

graphscaley Vertical grid spacing in centimetres for `\graphicmodel`'s Forrester layout (default 2). Larger values spread the diagram out vertically.

`graphscalex`/`graphscaley` control only the empty space between diagram elements: the nodes themselves (stocks, valves, auxiliaries, constants) have fixed dimensions expressed in `em` and are not individually configurable. Because `em` is relative to the current font size, the way to grow or shrink the nodes uniformly is to change the font size around the `\graphicmodel` call – e.g. wrap it in `{\small \graphicmodel}` or `{\large \graphicmodel}`. Pick spacing with `graphscalex`/`graphscaley`, pick node size with the surrounding font.

gridmaxx Maximum number of grid columns the auto layout may fill on any one row before wrapping (integer, default `0` = no limit). When the limit is reached, items already placed on the affected row (and everything above it for the stocks row, everything but stocks for the aux row, only the constants for the constants row) shift up by one row to free space, and placement continues from column 0. Manually positioned variables (`\mvar[gridx=...,gridy=...]`) are kept where they are. When wrapping is active the default centring of the aux row and the right-aligning of the stocks row are disabled, so the diagram fills left-to-right, top-to-bottom: what is declared first ends up on top.

diagram-style Rendering style for the case where a helper or constant is the direct inflow/outflow of a stock. Three values:

- **tight** (default) the valve takes the helper's/constant's label; the helper/constant itself is not drawn as a separate node. Compact and most LaTeX-native.
- **forrester** Forrester/Sterman convention: the valve is drawn without a label and the helper/constant remains as a separate node connected to the valve by a causal arrow.
- **edu** Didactic dual form: the valve carries the label *and* the helper/constant is drawn as a separate node with a causal arrow to the valve. Visually busy but pedagogically explicit.

flowarrow-style Visual style of the flow pipe. `hollow` renders the classic Forrester double-line pipe with an open arrow head; `filled` renders a thick solid arrow. The default tracks `diagram-style`: `forrester` picks `hollow`, the other styles pick `filled`. An explicit value overrides this coupling.

valve-style Visual style of the valve node. `valve` draws the bow-tie/butterfly icon (Forrester); `circle` draws an empty circle on the flow pipe; `edu` draws a labelled circle (the flow variable's display text inside). The default tracks `diagram-style`: `forrester` picks `valve`, the other styles pick `edu`.

flowarrow-cloud-tip Whether the open end of an inflow or outflow pipe is anchored to a cloud node, signalling the model boundary. Default tracks `diagram-style`: `forrester` picks `true`, the other styles pick `false`. May be set globally via `\numodelsetup`, per-render via `\graphicmodel`, or per-stock via `\mvar[flowarrow-cloud-tip=...]`. The most specific source wins.

start-format How a start value is shown by `\<prefix><Name>num/qty` and so in the *initial values* column of `\textmodel`: `rounded` (default) rounds it to the variable's significant figures, with a power of ten where needed ($h = 3{,}00 \cdot 10^3$ m); `input` shows it exactly as typed in `\mvar` ($h = 3000$ m, $k = 0{,}24$ kg/m, $e = 1{,}5 \cdot 10^{-3}$ m). Only a plain number has a typed form: a start value given as an expression (`2*3`, `sqrt(\oscK/\oscM)`) is always rounded. The setting is read when the value is shown, so it also applies to variables declared before it. Per variable: `\mvar[start-format=...]`.

engine How `\computemodel` computes: `lua` (default) compiles the rules to Lua, some hundreds of times faster than `fpeval`, which evaluates every rule with `\fp_eval` in decimal arithmetic; `compare` runs both, keeps the `\fp_eval` results and reports whether the Lua run agrees. `\computemodel[engine=...]` chooses for one model. See Section 4.3.

table-style Document-wide tabularray options for every `\tablemodel`, e.g. room to write in: `table-style={columns={c,colsep=8pt}, rows={rowsep=5pt}}`. A table's own `table={...}` comes after it. A macro holding the options may be given.

debug `true` writes `numodel`'s trace output (how variables, rules and tables are built) to the log; default `false`. Only useful when reporting a problem with `numodel` itself.

plot-points Maximum number of points per series that `\diagrammodel` plots (default 1000, 0 for all); see its `points` key.

blanks `open` (default) or `filled`: whether the rules, stop condition and start values declared with the `blank` key are shown as gaps ("$F_z = \cdots$", the pupil's version) or in full (the answer version). `\textmodel[blanks=...]` overrides it for one table; numodel-coach follows the same setting. See Section 7.

angles `radians` (default) or `degrees`: the angle unit of the program the model is written for, as Coach's "hoekeenheid". With `degrees` that program's `Sin`, `Cos` and `Tan` take degrees, so `\textmodel` and numodel-coach write `sind(\alpha)` as Sin(α) and `sin(x)` as Sin($^{180}/_{\pi} \cdot x$); with `radians` it is the other way round (Sin($^{\pi}/_{180} \cdot \alpha$) and Sin($x$)). The inverse functions (`Arcsin`, …) answer in the same unit, as in Coach: with `degrees` `asind(x)` is Arcsin(x) and `asin(x)` is $^{\pi}/_{180} \cdot$ Arcsin(x). The calculation does not change: in the rules, `sin` always takes radians and `sind` degrees. The text of a rule is fixed when it is declared, so set `angles` in the preamble (a warning reminds you otherwise).

units Whether the *initial values* cells in `\textmodel` display the SI unit alongside the value (`\qty`) or only the numeric value (`\num`). Boolean, default `true`. May also be supplied to `\textmodel[units=false]` as a per-table override; the global setting is restored after rendering.

tblrenv Which tabularray environment wraps the `\textmodel` table. Three values: `longtblr` (default, page-breakable); `tblr` (inline, no page breaks); `talltblr` (inline, no page breaks, supports `\caption`/notes). Inside a box (`minipage`, `subfigure`, a tcolorbox, a framed `solution` of exam) `longtblr` cannot break the page but could still split the table with a spurious *Continued on next page*; there `\textmodel` uses `tblr` instead. A box that breaks across pages has the same limit, so a long model table belongs outside it. May also be supplied per render as `\textmodel[tblrenv=...]`; the global setting is restored afterwards.

decimal-separator Decimal mark used by every number that `numodel` renders: the *initial values* column of `\textmodel`, decimal numbers inside the model rules (shown as typed, without rounding), and the tick labels of `\diagrammodel`. Two values:

- **comma** use a comma (siunitx `output-decimal-marker={,}`, pgfplots `/pgf/number format/use comma`).
- **point** use a full stop (siunitx `output-decimal-marker={.}`, pgfplots `/pgf/number format/use period`).

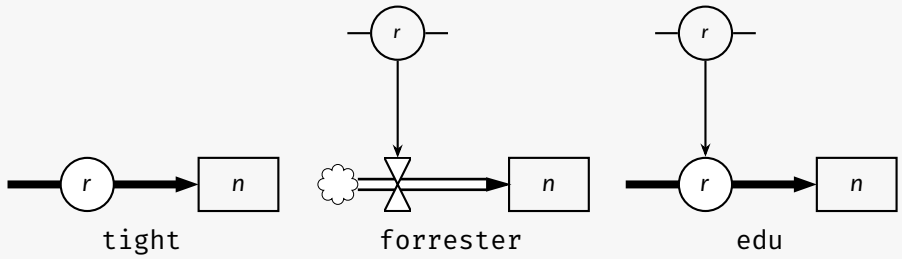
The default tracks `syntax`: `NL` picks `comma`, `EN` and `FPEVAL` pick `point`. Other language files can publish a default by defining `\__numodel_kw_<LANG>_dsep_default:` (expanding to `point` or `comma`); when the macro is absent the default is `point`. An explicit `decimal-separator` key locks that choice and overrides any future `syntax` change. The override is scoped: `numodel` applies it only inside its own renderers (siunitx's state is restored on group exit), so a document-wide `\sisetup` is not perturbed.

3.1 Diagram styles in practice

The same model rendered under each of the three `diagram-style` values. The model is the simplest case that distinguishes the styles: one stock N with a constant inflow R:

```
\newmodelprefix{flux}
\mvar{T}{t}{0}{\s}{2}{system}
\mvar{Dt}{dt}{1}{\s}{2}{system}
\mvar{N}{n}{0}{}{0}{stock}
\mvar{R}{r}{5}{\per\s}{2}{constant}
\mrule{N}{\fluxN + \fluxR * \fluxDt}
\mrule{T}{\fluxT + \fluxDt}
\mstop{\fluxT >= 5}
\begin{tabular}{@{}ccc@{}}
\graphicmodel[diagram-style=tight] &
\graphicmodel[diagram-style=forrester] &
\graphicmodel[diagram-style=edu] \\[2pt]
\texttt{tight} & \texttt{forrester} & \texttt{edu}
```

```
\end{tabular}
```



- `tight` collapses the constant *R* into the valve label, producing the most compact diagram.
- `forrester` keeps the canonical System-Dynamics convention: unlabelled bow-tie valve, the constant remains a separate node, the link from *R* to the valve is a thin causal arrow.
- `edu` is a didactic dual: the valve carries the label *and* the constant remains as a separate node with a causal arrow. Less compact but pedagogically explicit – useful when first introducing the stock/flow vocabulary.

4 Public API

4.1 Variables and rules

`\mvar` Declares a model variable. Signature:

```
\mvar[<keys>]{<Name>}{<text>}{<start>}{<unit>}{<sig>}{<type>}
```

where ⟨*Name*⟩ is a short alphabetic identifier (the prefix-qualified accessor becomes `\<prefix><Name>`, along with a family of suffixed accessor macros `\<prefix><Name><suffix>` documented in Section 6), ⟨*text*⟩ is the math-mode display symbol used in the rule table and diagram (e.g. `F_{res}`), ⟨*start*⟩ is the initial value (a number, or empty for helpers computed by a rule, or any expl3 `fp`-evaluable expression involving previously defined model variables), ⟨*unit*⟩ is a bare siunitx unit macro sequence (e.g. `\m\per\s\squared`), ⟨*sig*⟩ is the number of significant figures used by `\<prefix><Name>num/qty`, and ⟨*type*⟩ is one of `stock`, `aux`, `constant`, or `system`. Each English type also accepts a Dutch alias (`voorraad`, `hulp`, `constante`, `systeem`) for backwards compatibility with existing teaching material. See Section 5.

Naming caveat. All accessor macros of a given prefix share one flat TeX namespace: `\mvar{X}` generates `\<prefix>X` and every `\<prefix>X<suffix>` listed in Section 6. Pick ⟨*Name*⟩s so that no name equals another name followed by such a suffix, otherwise the two definitions silently overwrite each other. The canonical trap is declaring both `T` and `Tmax`: the `max`-suffix accessor of `T` (the extremum `\<prefix>Tmax`) collides with the no-suffix accessor of `Tmax`, so `\<prefix>Tmax` inside a rule expression ends up meaning whichever was declared last. The same hazard applies to prefix choices – avoid prefixes where one is a concatenation of another with a variable name (e.g. `ball` with a variable `Y` versus a prefix `ballY`). An `\mvar` name of `steps` would similarly collide with the per-prefix iteration accessor `\<prefix>steps` set by `\computemodel`.

`\mvar` detects these collisions. A full name that already belongs to a variable of *another* model is an error. A full name that is also an accessor of another variable (`T` next to `Tmax`, `V` next to `Vmin`) gives a warning naming both; documents that rely on the coincidence keep compiling, but the warning says which value wins. With the convention of a lower-case prefix and variable names that start with a capital, prefixes cannot collide with each other.

Optional ⟨*keys*⟩:

`prefix` Override the current prefix for this single call.

`gridx`, `gridy` Manual placement in the `\graphicmodel` grid; integers, −1 leaves the slot to auto-layout (default).

`alias` Math-mode token list that replaces the entire *initial values* cell.

`aliasleft`, `aliasright` Replace just the left symbol or right value half of the cell.

start-format `rounded` or `input` for this variable only, overriding the global `start-format` setting.

blank Show the start value as a gap in the pupil's version (`blanks=open`); see Section 7. `\mrule` and `\mstop` accept the same key.

`\mrule` Adds a rule of the form ⟨*LHS*⟩ ← ⟨*expr*⟩. Signature:

```
\mrule*[<keys>]{<LHS>}{<expr>}
```

Both forms add the rule to *both* the rule table (`\textmodel`) and the simulation (`\computemodel`); execution is identical. The star only changes the typeset layout when ⟨*expr*⟩ is a ternary `cond ? a : b`. Without the star the ternary is rendered on a single table row,

```
IF cond THEN lhs = a ELSE lhs = b ENDIF
```

which is compact but wide. With the star (`\mrule*`) the same ternary is broken across rows,

```
IF cond THEN
    lhs = a
ELSE
    lhs = b
ENDIF
```

keeping the table column narrow so a `\graphicmodel` can sit alongside it, and making the source itself easier to read. For non-ternary expressions the star has no effect. ⟨*expr*⟩ may use the full `\fp_eval` expression grammar.

Optional ⟨*keys*⟩ of `\mrule`:

prefix Another model than the current one.

blank Show the rule as "⟨*symbol*⟩ = ⋯" in the pupil's version (Section 7).

alias Replaces the whole row in `\textmodel`; `aliasright` only the right-hand side ("*v* = ⋯"). The model still computes with ⟨*expr*⟩.

before, after, add, replace Where the rule goes, and whether it replaces the rule for the same target; see Section 8.1.

See Table 2 (Section 4.2) for a complete overview of supported operators and functions with their XMILE and CoachTaal equivalents.

`\mruletext` Inserts a free-text row in the rule table without registering a rule with the simulator. Signature: `\mruletext[<keys>]{<text>}`. Useful for inserting comments or section dividers in long rule tables.

`\mstop` Sets the simulation stop condition. Signature: `\mstop[<keys>]{<expr>}`. The simulation halts at the first step where ⟨*expr*⟩ evaluates true. A model needs one `\mstop` before `\computemodel`, which otherwise stops with an error; a second `\mstop` replaces the first (Section 8.1). Keys: `prefix` and `blank`.

4.2 Expression reference

Table 2 lists all expression constructs for `\mrule` bodies. The **XMILE** column shows XMILE v1.0 syntax, **l3fp** shows the `\fp_eval`-compatible form used inside `\mrule`, and **CoachTaal** shows the Coach 7 equivalent (function names from the standard CoachTaal math reference; note that argument separators in CoachTaal are semicolons). Orange entries are not translated by numodel: the construct is missing on one side (no l3fp function, no Coach equivalent), or a translation would change its meaning (the quadrant of a two-argument arctangent). *Italic* cells contain a derived equivalent rather than a native keyword. The trigonometric rows show the default `angles=radians`; with `angles=degrees` the conversion factors move to the radian functions (see `angles` in Section 3).

Table 2: Expression reference: XMILE, `\fp_eval`, and CoachTaal

XMILE	l3fp (`\fpeval`)	CoachTaal
Arithmetic operators		
`+`	`+`	`+`
`-`	`-`	`-`
`*`	`*`	`*`
`/`	`/`	`/`
`^`	`^`	`^`
`MOD(x,y)`	*x - trunc(x/y)*y*	*x - Entier(x/y)*y*
Comparison operators		
`<`	`<`	`<`
`<=`	`<=`	`<=`
`>`	`>`	`>`
`>=`	`>=`	`>=`
`=`	`=`	`=`
`<>`	`!=`	`<>`
Boolean operators		
`AND`	`&&`	`EN`
`OR`	`||`	`OF`
`NOT`	`!`	`NIET`
Control flow		
`IF c THEN a ELSE b`	`c ? a : b`	`Als c Dan a Anders b EindAls`
General math functions		
`ABS(x)`	`abs(x)`	`Abs(x)`
`SIGN(x)`	`sign(x)`	`Teken(x)`
`SQRT(x)`	`sqrt(x)`	`Sqrt(x)`
*SIGN(x)*INT(ABS(x))*	`trunc(x)`	*Teken(x)*Entier(Abs(x))*
ROUND(x)	`round(x)`	`Round(x)`
*ROUND(x*10^n)/10^n*	`round(x,n)`	*Round(x*10^n)/10^n*
INT(x)	`floor(x)`	`Entier(x)`
-INT(-x)	`ceil(x)`	*-Entier(-x)*
`MIN(x,y)`	`min(x,y)`	`Min(x;y)`
`MAX(x,y)`	`max(x,y)`	`Max(x;y)`
FACT(x)	`fact(x)`	`Fac(x)`
INT(LOG10(ABS(x)))	`logb(x)`	*Entier(Log(Abs(x)))*
Exponential and logarithmic		
`EXP(x)`	`exp(x)`	`Exp(x)`
`LN(x)`	`ln(x)`	`Ln(x)`
`LOG10(x)`	*ln(x)/ln(10)*	`Log(x)`
Trigonometry (radians)		
`SIN(x)`	`sin(x)`	`Sin(x)`
`COS(x)`	`cos(x)`	`Cos(x)`
`TAN(x)`	`tan(x)`	`Tan(x)`
`ARCSIN(x)`	`asin(x)`	`Arcsin(x)`
`ARCCOS(x)`	`acos(x)`	`Arccos(x)`
`ARCTAN(x)`	`atan(x)`	`Arctan(x)`
`ARCTAN2(y,x)`	`atan(y,x)`	*Arctan(y/x)*
1/TAN(x)	`cot(x)`	*1/Tan(x)*
1/SIN(x)	`csc(x)`	*1/Sin(x)*
1/COS(x)	`sec(x)`	*1/Cos(x)*

Continued on next page

Table 2: Expression reference: XMILE, \fp_eval, and CoachTaal (Continued)

XMILE	l3fp (\fpeval)	CoachTaal
ARCSIN(1/x)	acsc(x)	*Arcsin(1/x)*
ARCCOS(1/x)	asec(x)	*Arccos(1/x)*
ARCTAN(1/x)	acot(x)	*Arctan(1/x)*
ARCTAN2(x,y)	acot(y,x)	*Arctan(x/y)*
Trigonometry (degrees)		
*SIN(PI/180*x)*	sind(x)	*Sin(Pi/180*x)*
*COS(PI/180*x)*	cosd(x)	*Cos(Pi/180*x)*
*TAN(PI/180*x)*	tand(x)	*Tan(Pi/180*x)*
*180/PI*ARCSIN(x)*	asind(x)	*180/Pi*Arcsin(x)*
*180/PI*ARCCOS(x)*	acosd(x)	*180/Pi*Arccos(x)*
*180/PI*ARCTAN(x)*	atand(x)	*180/Pi*Arctan(x)*
*180/PI*ARCTAN2(y,x)*	atand(y,x)	*180/Pi*Arctan(y/x)*
*1/TAN(PI/180*x)*	cotd(x)	*1/Tan(Pi/180*x)*
*1/SIN(PI/180*x)*	cscd(x)	*1/Sin(Pi/180*x)*
*1/COS(PI/180*x)*	secd(x)	*1/Cos(Pi/180*x)*
*180/PI*ARCSIN(1/x)*	acscd(x)	*180/Pi*Arcsin(1/x)*
*180/PI*ARCCOS(1/x)*	asecd(x)	*180/Pi*Arccos(1/x)*
*180/PI*ARCTAN(1/x)*	acotd(x)	*180/Pi*Arctan(1/x)*
*180/PI*ARCTAN2(x,y)*	acotd(y,x)	*180/Pi*Arctan(x/y)*
Constants		
PI	pi	Pi
e	*exp(1)*	*Exp(1)*
INF	inf	—
NAN	nan	—
PI/180	deg	*Pi/180*
1	true	Aan
0	false	Uit
Simulation-specific functions (XMILE only)		
TIME	*user variable (\mvar)*	*user variable*
DT	*user variable (\mvar)*	*user variable*
STEP(h,t_0)	*T >= t0 ? h : 0*	*Als T >= t0 Dan h Anders 0 EindAls*
RAMP(s,t_0)	*T > t0 ? s*(T-t0) : 0*	*Als T > t0 Dan s*(T-t0) Anders 0 EindAls*
DELAY(x,dt)	*not supported*	*not supported*
SMOOTH(x,t)	*not supported*	*not supported*
INIT(x)	*not supported*	*not supported*
PREVIOUS(x)	*not supported*	*not supported*
RANDOM(0,1)	rand()	Rand
RANDOM(lo,hi)	*lo + (hi-lo)*rand()*	*lo + (hi-lo)*Rand*
not supported	randint(n)	*not supported*
not supported	randint(m,n)	*not supported*

4.3 Render commands

\textmodel Renders the rule-and-startvalue table. Optional [<keys>] accepts prefix=<name> (render a non-current model), units=true or units=false, blanks=open or filled, and tblrenv=tblr, longtblr or talltblr (see \numodelsetup; inside a box longtblr becomes tblr). Each per-call key overrides the global \numodelsetup setting for this single render only; the global state is restored afterwards.

\graphicmodel Renders the Forrester stock-and-flow diagram. Variables of type stock become rectangles, aux circles and constant circles with two dashes; system variables are not drawn. Flow arrows connect stocks to constant or helper sources/sinks based on which variables appear in the right-hand side of

stock-updating rules.

Optional [<keys>] accepts prefix=<name> (render a non-current model), diagram-style=tight, forrester or edu, flowarrow-style, valve-style and flowarrow-cloud-tip, which override the global \numodelsetup setting for this single render only. The global state is restored afterwards, so multiple \graphicmodel calls can each pick their own style without re-issuing \numodelsetup.

\computemodel Runs the Euler simulation. Each iteration step evaluates the stop condition and then every \mrule body in declaration order; by default in Lua, with engine=fpeval in LaTeX itself through expl3's \fp_eval (the engine behind the LaTeX2e \fpeval), writing each new value into the accessor macro \<prefix><Name>. The per-step values are kept in Lua tables, so even a 20 000-step run stays linear in total work. After \computemodel returns, \<prefix><Name>min / \<prefix><Name>max hold the extrema (for a variable without start value from step 1: in step 0 it is still 0, as in Coach's first table row), \<prefix><Name> holds the final-step value, \<prefix>steps expands to the number of recorded samples (i.e. the iteration count *N*, step 0 included), and the full time-series can be retrieved with \mcoords / \mstep / \mstepat.

Two engines. By default (engine=lua) the rules are translated into Lua functions – by the same parser that renders them as Coachtaal – and the loop above runs in Lua. A model of 8 000 steps takes a few hundredths of a second; with engine=fpeval every rule goes through \fp_eval and the same model takes some seconds. You keep writing rules in l3fp syntax; TeX macros other than the model's own variables (a value defined with \edef, \fpeval{...}, a variable of another model) are expanded once, before the run, exactly as \fp_eval would see them. A model that cannot be translated – an l3fp function the translator does not know – is computed with \fp_eval, with an info message naming the reason, so engine=lua never breaks a document.

l3fp computes in decimal, Lua in binary. Values agree to about 15 significant digits; the final values are handed back to TeX rounded to 12, which hides the binary noise (162,6 rather than 162,600 000 000 009 4). A comparison on a boundary could tip, though: ten steps of 0,1 give 0,999 999 999 999 999 89 in binary, so $t \geqslant 1$ would hold one step late. The Lua engine therefore compares with a relative tolerance of 10^{-9}, which stops every model at the same step as \fp_eval. Use engine=compare once for a document to check this for your own models: it warns for every model where the number of steps or a final value differs.

Why `fpeval` is still there. l3fp computes in *decimal* with 16 significant digits, as you would by hand: 0,1 + 0,2 is exactly 0,3, and a time step of 0,1 lands exactly on $t = 1, 2, \ldots$, with no tolerance needed. The results are the same, digit for digit, as any \fpeval or \num[evaluate-expression] elsewhere in the document, so a value you compute in the text from the model's data agrees exactly with the model. Every l3fp function and construct is available, also the rare ones the Lua translator does not know (a model using one falls back to fpeval by itself). engine=fpeval is the reference for engine=compare, and it computes exactly as numodel 0.10 and earlier did. The price is speed: choose it when exact decimal arithmetic matters more than compilation time, for instance a short model whose step-by-step values are discussed digit by digit.

What a step contains. Every round first *records* the current value of every variable, then checks the stop condition, and then executes the rules in order. Step 0 therefore holds the start values, and step *n* the values after *n* rounds. For a stock that is simply its value after round *n*. An auxiliary variable, however, is recorded with the value its rule produced in round *n* – computed from the stocks as they were *before* that round, i.e. from step $n - 1$ when its rule comes before the stock updates (the usual order). A table row "t, v, $F_w(v)$, a" thus takes t and v from step n and F_w and a from step $n + 1$. At step 0 an auxiliary variable holds its start value, or 0 when it has none; give it a start-value expression (as in the energy example below) when step 0 should show the physically meaningful value.

\diagrammodel Convenience wrapper that produces a complete figure with caption and label. Signature:

\diagrammodel[<keys>]{<xvar>}{<yvar>⟨,...⟩}[<extra>]{<label>}

Reads min/max and display text from \<prefix><xvar> etc., delegates to \drawplot from numodel-plot, and emits \caption{$<prefix><yvar><text>(<prefix><xvar><text>)$-diagram}\label{fig:<label>}. The optional ⟨*extra*⟩ argument is appended to the axis body, useful for additional \addplot lines (annotations, theoretical curves). Must be called after \computemodel.

The ⟨*yvar*⟩ argument accepts a comma-separated list of variables: every entry whose unitraw matches the first one's is drawn into the same diagram as discrete model points (no connecting lines), with the y-axis range scaled to the joint min/max of the kept series. Entries with a non-matching unit are dropped and a warning is issued. The seven colours cycle through Okabe & Ito's colour-blind safe palette (yellow omitted), ordered so consecutive series differ in luminance – which keeps them distinguishable on a greyscale printout as well. The legend lists each kept variable's display text. A

diagram with a single series has no legend by default; `legend=true` adds one with the text from the keyword table of the current `syntax` ("modelpunt" for `NL`, "model point" otherwise).

Keys of ⟨*keys*⟩, besides `prefix`:

`legend` `true` or `false`, or a list with one label per series (in series order), which also switches the legend on: `legend={$\Delta t = 1$\,s, $\Delta t = 0{,}1$\,s}`. A missing label falls back to the default text.

`style` `marks` (default: the model points), `line` (a line through them), or `both`.

`float` `false` sets the diagram where it stands, centred, instead of in a `figure` float – so it works inside a `minipage`, a tcolorbox or the `solution` environment of the exam class. The caption still numbers it as a figure. Inside a list (`enumerate`, exam's `questions`) diagram and caption take the width of the list's lines.

`caption` Caption text instead of the automatic "$v(t)$-diagram".

`nocaption` No caption, and so no figure number and no label.

`axis`, `plot`, `series` pgfplots options passed on unchanged, after `numodel`'s own, so they override them: `axis={...}` for the axis (`legend pos=north west`, `legend style={...}`, …), `plot={...}` for every series (`mark size=1.6pt`), and `series={{...},{...}}` for one series each, in series order (`series={{}, {mark size=0.5pt}}`). `numodel` keeps its own keys for what concerns the model and the layout; drawing details are left to pgfplots.

`points` Maximum number of points plotted per series, as the `plot-points` setting (default 1000; 0 plots all). A longer series is thinned by distance in the diagram: a sample is left out when it lies very close to the last one shown, so flat stretches, where thousands of marks overlap, become lighter while every step of a fast change stays visible. No point is moved or invented; only the plot is affected (`\mcoords`, `\mstep` and exports see all samples). An 8 000-step series plots in a tenth of the time.

`xcmmax`, `ycmmax` Maximum axis length in cm for this diagram, as the keys of the same name of `\numodelplotsetup`; a smaller value gives a smaller diagram.

`equal` Equal scales on both axes (a centimetre is the same amount on the x- and the y-axis), for a path such as y against x. `numodel` chooses the axis ranges and ticks with it, so the diagram stays within `xcmmax` and `ycmmax`; do not use pgfplots' `axis equal` for this. Also a `\numodelplotsetup` key.

A variable without unit gets no parentheses in its axis label ("X" instead of "X ()"), unless the axis is scaled ("X (10^3)"). A count can also get a unit of its own, `\DeclareSIUnit{\animals}{animals}`; numodel-coach exports it as that word.

The `prefix=` key likewise accepts a comma-separated list, so the same y-variables are plotted across several models in one diagram – useful for direct visual comparison between alternative model variants of the same physical setup. Series order is prefix-major (all y-variables of the first prefix come before any of the second), the colour cycle continues across the cross product, and axis ranges reduce over the union of all (prefix, yvar) pairs. The unit filter is applied against the first prefix's first y-variable, so the same short names are assumed to share unit and display text across prefixes. When more than one prefix is supplied the prefix is appended in parentheses after each y-variable's display text in the legend, e.g. `$U_3$ (ptc)` and `$U_3$ (pw)`. Single-prefix calls render exactly as before.

```
\diagrammodel[prefix={ptc,pw}]{T}{U_3,U_par}{compare-models}
```

Example – two quantities in one diagram. The free-fall ball from the introduction is extended with gravitational potential energy $E_p = m\,g\,h$ (written as $-m\,G\,Y$ because $G = -9{,}81$ already encodes the downward direction) and kinetic energy $E_k = \frac{1}{2}\,m\,v^2$. Both share the unit `\joule`, so the multi-series filter keeps both and scales the y-axis to their combined range. At $t = 0$ all energy sits in E_p; as the ball falls, E_p drops and E_k grows. The two `aux` variables receive a start-value expression so that step 0 is recorded with the physically meaningful initial energies rather than empty values.

```
\newmodelprefix{nrg}
\mvar{T}{t}{0}{\s}{2}{system}
```

```
\mvar{Dt}{dt}{0.1}{\s}{2}{system}
\mvar{V}{v}{0}{\m\per\s}{2}{stock}
\mvar{Y}{y}{100}{\m}{3}{stock}
\mvar{m}{m}{0.5}{\kg}{3}{constant}
\mvar{G}{g}{-9.81}{\m\per\s\squared}{3}{constant}
\mvar{Ep}{E_p}{-\nrgm * \nrgG * \nrgY}{\joule}{3}{aux}
\mvar{Ek}{E_k}{0.5 * \nrgm * \nrgV * \nrgV}{\joule}{3}{aux}
\mrule{V}{\nrgV + \nrgG * \nrgDt}
\mrule{Y}{\nrgY + \nrgV * \nrgDt}
\mrule{Ep}{-\nrgm * \nrgG * \nrgY}
\mrule{Ek}{0.5 * \nrgm * \nrgV * \nrgV}
\mrule{T}{\nrgT + \nrgDt}
\mstop{\nrgY <= 0}
\computemodel
\diagrammodel{T}{Ep,Ek}{ball-energy}
```

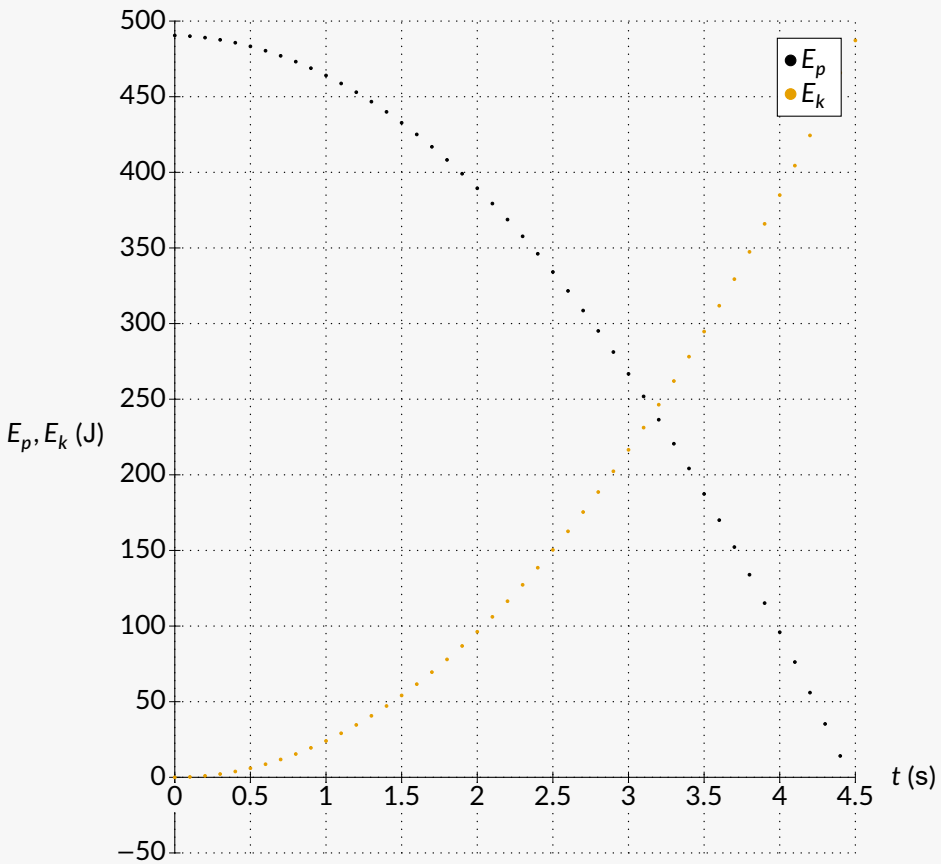


Figure 2: $E_p, E_k(t)$-diagram

4.4 Series accessors

`\mcoords` Returns the simulated series of two variables as a PGFPlots coordinate list, `(x,y) (x,y) ...`, with all samples. Fully expandable. Signature: `\mcoords{<xvar>}{<yvar>}` (current prefix); `\mcoordsp{<prefix>}{<xvar>}{<yvar>}` (explicit prefix). Both forms are usable inside `\addplot coordinates{ ... }`.

`\mstep` Returns the value of one variable at a chosen iteration. Fully expandable. Signature: `\mstep{<Name>}{<i>}`; `\mstepp{<prefix>}{<Name>}{<i>}`. The current prefix is prepended automatically by `\mstep`. Step indexing is 0-based: step 0 is the initial-values row, step $N-1$ is the final recorded step after `\computemodel`. Negative indices count from the end (Python-style), so `\mstep{Y}{-1}` is the last recorded y and `\mstep{Y}{-2}` is the penultimate one. Raises a `nu_model` error when $\langle i \rangle$ falls outside the recorded range (`0..\<prefix>steps - 1` for non-negative

indices, or `-\<prefix>steps..-1` for the Python-style negative form), which surfaces typo'd indices and "ran *N* steps but indexed step *N*" off-by-one mistakes rather than expanding to nothing and propagating empty arguments into the surrounding plot.

`\tablemodel` `\tablemodel[<keys>]{<Name>,<Name>,...}` sets a table of computed values: one column per variable, headed "symbol (unit)", one row per step. The numbers are rounded to each variable's significant figures, with the decimal marker of the syntax. Keys:

`steps` Which steps: `0-5` (default), `{0,1,2,10-12}`.

`row` `step` (default): every value from step *n*. `state`: the state of step *n* with the auxiliary values computed from it, which are recorded in step $n+1$ (see "What a step contains" under `\compu⌋temodel`) – the row a pupil computes by hand.

`blank` Empty cells in the pupil's version (`blanks=open`, the default), except the first column and the first `given` rows; with `blanks=filled` the table is complete.

`given` Number of rows filled in anyway (an example row).

`blanks` `open` or `filled` for this table.

`num` siunitx options per column, after `numodel`'s rounding: `num={T={round-mode=places,rou⌋nd-precision=0}}` shows the time as whole numbers (the options may also be a macro).

`table` tabularray options added to the table, after the document-wide `table-style` (see `\numod⌋elsetup`), so they override it. A macro holding the options may be given.

`prefix` Another model than the current one.

Example – the first steps of the energy model above, once complete and once as an exercise with one example row:

```
\tablemodel[steps=0-4]{T,V,Y,Ek}

\medskip
\tablemodel[steps=0-4, blank, given=1]{T,V,Y,Ek}
```

t (s)	v (m s^{-1})	y (m)	E_k (J)
0	0	100	0
0.10	–0.98	99.9	0.241
0.20	–2.0	99.7	0.962
0.30	–2.9	99.4	2.17
0.40	–3.9	99.0	3.85

t (s)	v (m s^{-1})	y (m)	E_k (J)
0	0	100	0
0.10			
0.20			
0.30			
0.40			

For a model with an auxiliary force, `row=state` gives the rows a pupil computes by hand, and `num` shows the time as whole numbers:

```
\tablemodel[steps=0-5, row=state, blank, given=1,
num={T={round-mode=places, round-precision=0}}]{T,V,Fw,Fres,A,Dv}
```

`\mstepat` Returns the value of a variable at the first step where another variable – usually the time – has

reached a given value, so the text does not depend on the time step. Fully expandable. Signature: `\mstepat{<Name>}{<Var>}{<value>}`; `\mstepatp{<prefix>}{<Name>}{<Var>}{<value>}`. Example: `\mstepat{V}{T}{1}` is v at $t = 1$ s, whether Δt is 0,1 or 0,01 s. Round-off of an accumulated time step is tolerated ($t = 0{,}999\,999\,999$ counts as 1). Raises an error when ⟨*Var*⟩ never reaches ⟨*value*⟩.

"Reaches" follows the direction in which ⟨*Var*⟩ starts: from below the value it is the first step at or above it, from above the first step at or below it. So ⟨*Var*⟩ need not be the time: `\mstepat{T}{C}{30}` is the time at which C reaches 30 mg/L, `\mstepat{T}{Vy}{0}` the time at which a falling v_y becomes zero (the top of a throw), and `\mstepatp{flu}{T}{Z}{\fluZmax}` the time of the peak of Z. Such a number always matches the model, also after a change of the time step.

Example – a red secant line through the last two simulated (t, y) points of the free-fall ball (the model from the first example), extrapolated across the whole t-domain and labelled in the legend. The slope is the rise-over-run of the last two steps, the intercept is the final y-value. Inside the optional `[<extra>]` argument `\mstep` expands as a literal number into PGFPlots' math parser, so each `\mstep` is evaluated only once (when the expression is constructed) rather than per sample. Applied to the `ball` model:

```
\diagrammodel{T}{Y}[%
  \addplot[red, very thick, domain=\ballTmin:\ballTmax]
    {\mstep{Y}{-1}
      + (\mstep{Y}{-1} - \mstep{Y}{-2})
      / (\mstep{T}{-1} - \mstep{T}{-2})
      * (x - \mstep{T}{-1})};
  \addlegendentry{secant endpoints}
]{ball-fall-secant}
```

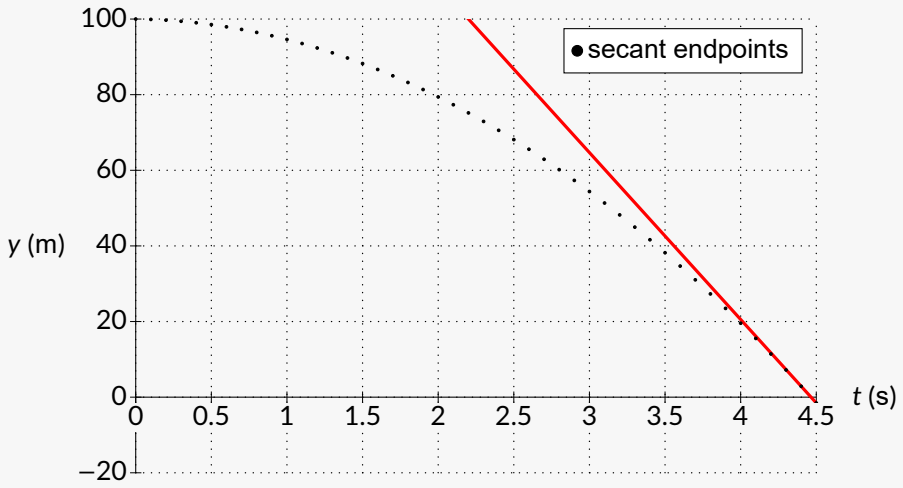


Figure 3: $y(t)$-diagram

4.5 Namespace management

`\newmodelprefix` Creates a new model namespace and switches to it. Signature: `\newmodelprefix{<name>}`. Subsequent `\mvar` / `\mrule` / `\mstop` calls bind to this prefix. All generated accessors are prefixed (so `\mvar{Y}{y}{...}{...}{...}{...}` under prefix `ball` produces `\ballY`, `\ballYtext`, etc.).

`switchmodelprefix` Switches to a previously created prefix. Useful when a document defines several models early and renders them later out of order. Signature: `\switchmodelprefix{<name>}`.

`NumodelForEachVar` Iterates a token list over every registered variable across every known prefix. Signature: `\NumodelForEachVar{<code with #1>}`; inside the body, `#1` is the full prefixed name (e.g. `ballY`). Used by external tools (e.g. a worksheet system) that need to expand every model accessor.

4.6 Shared valves

When the same auxiliary variable appears as the inflow of more than one stock, `\graphicmodel` draws a single valve next to the first such stock and threads a curved branch (`to[bend left=30]` – the same bend as the curved causal arrows) from that valve over the primary stock into each additional one. The auto-layout leaves the involved stocks side by side on the same row so the branch stays compact. A minimal example:

```
\mvar{R}{r}{5}{\m\cubed\per\s}{2}{aux}
\mvar{Va}{V_1}{0}{\m\cubed}{3}{stock}
\mvar{Vb}{V_2}{0}{\m\cubed}{3}{stock}
\mrule{Va}{\Va + \R * \Dt}
\mrule{Vb}{\Vb + \R * \Dt}
```

renders one `R` valve, a straight pipe `R` → `V_a`, and a curved branch `R` → `V_b` arcing over `V_a`. (Note: TeX control words do not accept digits, so the macro names `Va`/`Vb` are used internally while the display texts `V_1`/`V_2` appear in the rendered diagram and table.)

Shared *outflows* are handled symmetrically. When one variable drains more than one stock, the valve is placed to the right of the last-declared source stock, and every earlier source attaches with a curved branch arcing over the intervening stocks into the shared valve. This matters in particular for physical models where a single Stefan–Boltzmann constant or friction coefficient drains several reservoirs into the same sink.

A valve can also be shared around a between-flow: an extra stock with the same inflow term, or an extra source with the same outflow term, gets a curved branch to the valve of the between-flow.

Two rules of the flow classification matter for these layouts:

- A term with more than one variable has no flow variable. In the rules of two stocks with opposite signs it is a between-flow, but it cannot share a valve with a third stock (Section 2.2.1). To share a flow, give it a variable of its own.
- Top-level parenthesised terms of the form `(A - B)*C` are distributed into `+A*C, -B*C` before classification, so a difference gives an inflow and an outflow, typically a gain and a loss. With $C = \Delta t$ the valves are named after `A` and `B`. A sum such as `(A + B)*C` stays one flow.

5 Variable types

The sixth `\mvar` argument (⟨*type*⟩) tags a variable with a role. The type drives both `\textmodel` layout (whether the row appears in *initial values*) and `\graphicmodel` node shape. Each type has a canonical English name and a Dutch alias; the two are interchangeable.

`stock` A stock that accumulates over time. Drawn as a rectangle in the Forrester diagram; its rule must be of the form stock ← stock + ... so that the integrator can detect inflows and outflows. Receives an *initial values* row.

`constant` A constant parameter (mass, gravitational acceleration, spring stiffness, ...). Drawn as a circle with two dashes. Receives an *initial values* row.

`aux` An auxiliary variable computed from other variables on each step. Drawn as a circle. Usually without start value, and then without *initial values* row; a start value (or start-value expression) gives it one and fixes its value in step 0.

`system` System-level bookkeeping (time `T`, step size `Dt`, terminal time, ...). Not drawn in the Forrester diagram. Receives an *initial values* row.

6 Generated accessors

Each `\mvar` call generates a family of accessor macros named `\<prefix><Name><suffix>`. The full set:

(no suffix) Current numeric value (post-rule-update if inside a step, post-final-step after `\computemodel`).

`text` The display symbol passed as the second argument of `\mvar`.

unit `\unit{...}` applied to the unit argument (siunitx-formatted).

unitraw The unit argument verbatim, without the `\unit{}` wrapper, suitable for use as a building block (e.g. `\xlabelunit` in numodel-plot).

num The *start value* formatted via siunitx's `\num{}`: rounded to the variable's significant figures, or as typed with `start-format=input`. Used by `\textmodel` in the *initial values* column when `units=false`. It is fixed by `\mvar` and does not follow `\computemodel`: for a computed value use the bare accessor (the final value), `\mstep` or `\mstepat`, formatted with `\num` or `\qty` (or a document macro of your own built on them).

qty As `num`, but including the unit (`\qty{value}{unit}`). Used by `\textmodel` in the *initial values* column when `units=true` (the default).

pre As `qty` (so also the start value), but with engineering prefix mode (e.g. `1500 W` renders as `1{,}5\,`kW).

sign Significant-figure count (raw integer).

type Variable type (raw string).

min, max Extrema over the simulated series; for a variable without start value from step 1, as step 0 holds 0. Defined by `\computemodel`.

gridx, gridy Manual placement coordinates in the Forrester grid; −1 if left to auto-layout.

alias, aliasleft, aliasright Override tokens for the *initial values* cell layout. `aliasright` replaces just the value part of the cell (keeping the symbol and equals sign); `aliasleft` replaces just the symbol; `alias` replaces the entire cell.

In addition, `\computemodel` writes one per-*prefix* accessor (no variable name in the middle):

\<prefix>steps The number of recorded samples after the most recent `\computemodel` call for this prefix, i.e. the iteration count N. Step 0 (the start values) is included, so the model went through $N - 1$ rounds. Empty (undefined) before the first `\computemodel`. Picking `steps` as an `\mvar` name would collide with this accessor – see the naming caveat in Section 4.

7 Exercises: pupil's and answer version

The `blank` key makes one declaration serve both versions of an exercise. On `\mvar`, `\mrule` (also `\mrule*`) and `\mstop` it marks what the pupil has to supply. With `blanks=open` – the default, the pupil's version – `\textmodel` shows such a rule as "⟨*symbol*⟩ = ⋯" (a conditional rule collapses to that single row, so its structure is part of the question), the stop condition as "IF ⋯ THEN STOP ENDIF", and the start value as "⟨*symbol*⟩ = ⋯". With `blanks=filled` everything is shown in full. `\computemodel` always uses the full rules, so diagrams and computed answers are the same in both versions. Switch once for the answer version, e.g. with the exam class: `\ifprintanswers\numodelsetup{blanks=filled}\fi`. The same setting governs `\tablemodel[blank]` (empty cells to fill in) and the Coach file of numodel-coach (the gaps become comments).

```
\newmodelprefix{gap}
\mvar{T}{t}{0}{\s}{2}{system}
\mvar{Dt}{dt}{0.5}{\s}{2}{system}
\mvar{V}{v}{0}{\m\per\s}{2}{stock}
\mvar[blank]{G}{g}{-9.81}{\m\per\s\squared}{3}{constant}
\mrule[blank]{V}{\gapV + \gapG * \gapDt}
\mrule{T}{\gapT + \gapDt}
\mstop[blank]{\gapT >= 3}
\textmodel
\textmodel[blanks=filled]
```

	model	initial values
1	$v = \cdots$	$t = 0\,\mathrm{s}$
2	$t = t + dt$	$dt = 0.50\,\mathrm{s}$
3	IF $\cdots$ THEN STOP ENDIF	$v = 0\,\mathrm{m\,s^{-1}}$
		$g = \cdots$

	model	initial values
1	$v = v + g \cdot dt$	$t = 0\,\mathrm{s}$
2	$t = t + dt$	$dt = 0.50\,\mathrm{s}$
3	IF $t \geqslant 3$ THEN STOP ENDIF	$v = 0\,\mathrm{m\,s^{-1}}$
		$g = -9.81\,\mathrm{m\,s^{-2}}$

7.1 Other replacements: the alias keys

`blank` needs no replacement text and switches between both versions. For any other replacement – a question mark, a hint – the alias keys of `\mvar` and `\mrule` replace what `\textmodel` shows. `aliasright=\cdots` blanks the value while keeping the symbol visible (asking *what value* belongs there), `alias={?}` blanks the entire cell (asking the student to supply both the symbol and the value from physical knowledge). Crucially, `\computemodel` keeps using the original numeric start value, so the diagram or computed answer for the rest of the model remains correct – only the rendered table changes. The row for y now reads $y = \cdots$ (asking the student to identify the initial height from a graph or photograph) and the row for g shows just ? (asking the student to supply the gravitational acceleration from memory). The rule column is unaffected – `\mrule` has its own `alias`/`aliasright` keys for blanking rule bodies if needed.

```
\newmodelprefix{quiz}
\mvar{T}{t}{0}{\s}{2}{system}
\mvar{Dt}{dt}{0.5}{\s}{2}{system}
\mvar{V}{v}{0}{\m\per\s}{2}{stock}
\mvar[aliasright=\cdots]{Y}{y}{50}{\m}{3}{stock}
\mvar[alias={?}]{G}{g}{-9.81}{\m\per\s\squared}{3}{constant}
\mrule{V}{\quizV + \quizG * \quizDt}
\mrule{Y}{\quizY + \quizV * \quizDt}
\mrule{T}{\quizT + \quizDt}
\mstop{\quizY <= 0}
\textmodel
```

	model	initial values
1	$v = v + g \cdot dt$	$t = 0\,\mathrm{s}$
2	$y = y + v \cdot dt$	$dt = 0.50\,\mathrm{s}$
3	$t = t + dt$	$v = 0\,\mathrm{m\,s^{-1}}$
4	IF $y \leqslant 0$ THEN STOP ENDIF	$y = \cdots$
		?

8 Multiple models in one document

Each `\newmodelprefix` starts a fresh namespace. This lets a document contain several unrelated models without name clashes:

```
\newmodelprefix{ball}
\mvar{Y}{y}{100}{\m}{3}{stock}
% ... ball model ...

\newmodelprefix{spring}
\mvar{X}{x}{0.1}{\m}{3}{stock}
% ... spring model ...

% Render the ball model:
\switchmodelprefix{ball}\textmodel\computemodel\diagrammodel{T}{Y}{fall}

% Render the spring model:
\switchmodelprefix{spring}\textmodel\computemodel\diagrammodel{T}{X}{spring}
```

Each prefix carries an independent rule list, stop condition, and recorded series.

8.1 Changing and copying models

A model is a list of declarations, and it can be changed after it has been written: useful for a series of models that differ in one point (the time step, one wrong rule, one extra force).

`\mvar` on a name the model already has replaces that variable *in place*: same position in the *initial values* column, new value, unit or type.

`\mrule` *adds* a rule, also for a target that already has one; `[add]` says so explicitly. It goes before the stop condition when there is one (else at the end), or where `before=`⟨*Name*⟩ or `after=`⟨*Name*⟩ puts it: before or after the rule for that variable. `\mruletext` takes the same position keys.

`\mrule[replace]` replaces the one rule for its target, in place (or at `before=`/`after=`). It is an error when the target has no rule or several; the key makes visible in the source that a rule is being replaced.

`\mremoverule[`⟨***prefix=...***⟩**`]{`**⟨***Name*⟩**`}`** removes every rule for ⟨*Name*⟩.

`\mstop` a second time replaces the stop condition.

`\copymodel` `\copymodel[<keys>]{<source>}{<copy>}` creates the model ⟨*copy*⟩ with all declarations of ⟨*source*⟩; references in rules are renamed (`\sprongV` becomes `\foutV`). ⟨*copy*⟩ becomes the current model, so the ordinary commands, with the copy's names, change it further. Keys: ⟨*Name*⟩=⟨*value*⟩ sets a start value; `blank={`⟨*Name*⟩`,...}` gives the rules for these targets the `blank` key (Section 7). The copy is a snapshot: later changes to ⟨*source*⟩ do not reach it. Not copied: computed results (call `\computemodel`), diagrams, and numodel-coach's instruction text.

```
\copymodel[Dt=0.5]{fall}{fallhalf}          % another time step
\copymodel[blank={Fz,Fw}]{fall}{template}   % pupil's template
\copymodel{fall}{wrong}                     % one wrong rule
\mrule[replace]{Fres}{\wrongFz + \wrongFw}
\copymodel[Dt=0.02]{fall}{chute}            % an extension
\mvar{Kpar}{k_{\text{par}}}{30}{\kg\per\m}{2}{constant}
\mvar{K}{k}{0.24}{\kg\per\m}{2}{aux}         % K was a constant
\mrule*[before=Fz]{K}{\chuteH < 600 ? \chuteKpar : 0.24}
```

Each change rebuilds the model from its list of declarations, so the table, the computation and any export always agree. Variables are declared before the rules when a model is rebuilt, so a rule may use a variable added to a copy after the rule's original position.

`\copymodel` is also the way to declare several models that differ only in a value. A home-made macro that builds the model for a prefix given as argument, with rules such as `\mrule{V}{\csname #1V\endcsname + ...}`, computes correctly but has a pitfall: `\csname` is expanded when the rule is declared, so `\textmodel` shows the numbers instead of the names ($t = 50 + 0{,}1$) and numodel-coach reports unknown variables. Declare one model in full and copy it.

9 Requirements

numodel requires LuaLaTeX (the engine, for the Lua runtime) and TeX Live 2024 or later. Mandatory dependencies: expl3, xparse, l3keys2e, amsmath, amssymb, xfrac, tikz, luacode, siunitx, float, tabularray, and the sibling package numodel-plot (which itself pulls in pgfplots). The companion Lua module `numodel.lua` must be installed alongside the `.sty` in a directory searched by `kpse`.