

The numodel package*

Paul Zuurbier
mail@paulzuurbier.nl

October 11, 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	1
2	First example: a free-falling ball	2
2.1	<code>\textmodel</code> — rule table	2
2.2	<code>\graphicmodel</code> — Forrester diagram	2
2.3	<code>\computemodel</code> + <code>\diagrammodel</code> — numerical plot	3
3	Configuration	3
3.1	Diagram styles in practice	6
4	Public API	6
4.1	Variables and rules	6
4.2	Expression reference	8
4.3	Render commands	10
4.4	Series accessors	13
4.5	Namespace management	15
4.6	Shared valves	15
5	Variable types	15
6	Generated accessors	16
6.1	Pupil's and answer version: <code>blank</code>	17
6.2	Changing and copying models	18
7	Multiple models in one document	19
8	Requirements	19

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`);

*This document corresponds to `numodel` v0.11.0, dated October 11, 2026.

- 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 simulation engine runs in Lua (through `luacode`) for $\mathcal{O}(1)$ appends and cheap min/max tracking; the rendering layer is pure `expl3`.

2 First example: a free-falling ball

A ball dropped from $h_0 = 100$ 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	$v = v + g \cdot dt$	$t = 0 \text{ s}$
2	$y = y + v \cdot dt$	$dt = 0.10 \text{ s}$
3	$t = t + dt$	$v = 0 \text{ m s}^{-1}$
4	IF $y \leq 0$ THEN STOP ENDIF	$y = 100 \text{ m}$ $g = -9.81 \text{ m s}^{-2}$

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 $\leq$.

2.2 `\graphicmodel` — Forrester diagram

`\graphicmodel` draws the same model as a stock-and-flow diagram. Stocks (type `stock`) are rectangles; auxiliaries (`aux`) and constants (`constant`) are circles, with two short horizontal dashes flanking the constant ring. In the default `tight` diagram-style, an `aux` or `constant` that is the direct inflow or outflow of a stock is absorbed into the valve label and not drawn as a separate node; `diagram-style=forrester` or `edu` keep both (Section 3).

Flows are inferred per additive term on the right-hand side of each stock rule. After dropping the self-carry term, every `+T` is a candidate inflow and every `-T` a candidate outflow of the target stock;

the first non-system, non-Dt variable appearing inside T becomes the flow variable (aux and stock beat constant when several candidates coexist). When the same flow variable surfaces as an outflow term of stock A and a matching inflow term of stock B, the pair is promoted to a *between-flow*: one valve drains A into B instead of two separate valves. If the flow variable of an inflow term is itself a stock, the source stock takes the valve's place – with a matching outflow term on its own rule (a conserved-quantity between-flow) or, when no such match exists, with a cloud-fed phantom valve plus a causal link to the source stock:



Layout is automatic from the rule graph: stocks (with their inflow/outflow valves) sit on the bottom row, auxiliaries on the middle row, and constants on the top row, each filled left-to-right in declaration order. The order of `\mvar` calls therefore drives the horizontal reading order of the diagram – reordering, inserting or holding back a `\mvar` nudges the visible arrangement without touching any grid coordinates, which is the primary lever a document author has over the look of the diagram. Manual placement remains available through `gridx/gridy` keys on the `\mvar` call (see Section 4). Wide diagrams can be capped to a maximum number of grid columns with `\numodelsetup{grid_max=N}`: when a row reaches N, the auto layout shifts the affected items up one row and continues filling.

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}
```

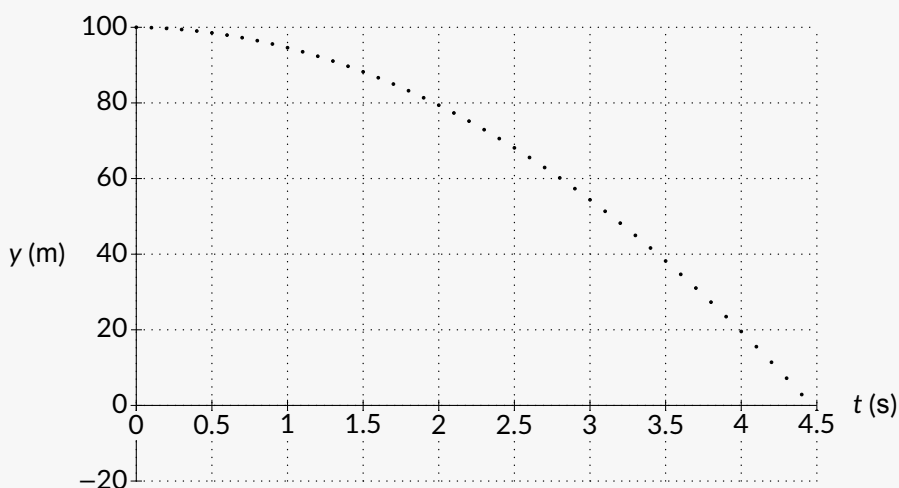


Figure 1: $y(t)$ -diagram

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` and `numodel-NL.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 aborts with a warning naming the unmet stop condition.

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, bottom-to-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-m-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 ($2*3$, $\text{sqrt}(\text{oscK}/\text{oscM})$) 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 ($F_z = \dots$, 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 6.1.

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( $\alpha$ )` 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 `$\pi/180 \cdot \text{Arcsin}(x)$` . 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`). Pick `tblr` or `talltblr` when `\textmodel` sits inside an enclosing environment that suppresses page breaks (`subfigure`, `minipage`, ...); the default `longtblr` would otherwise emit spurious "(Continued)" / *Continued on next page* markers in that setting. 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 for mat/use comma`).

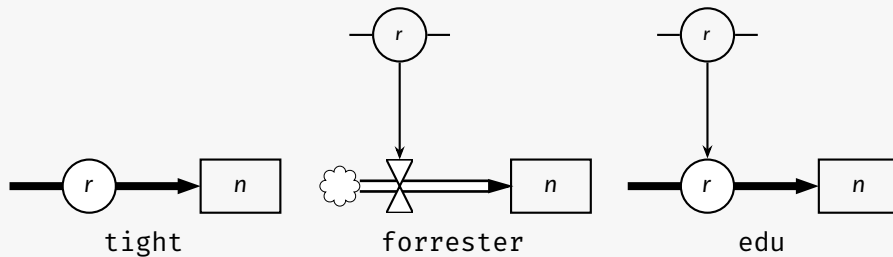
point use a full stop (`siunitx output-decimal-marker={.}`, `pgfplots/pgf/number for rmat/use period`).

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}{\s}{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}
\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 $\langle Name \rangle$ 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), $\langle text \rangle$ is the math-mode display symbol used in the rule table and diagram (e.g. F_{res}), $\langle start \rangle$ 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), $\langle unit \rangle$ is a bare `siunitx` unit macro sequence (e.g. `\m\per\s\squared`), $\langle sig \rangle$ is the number of significant figures used by `\<prefix><Name>`, `num/qty`, and $\langle type \rangle$ 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 6.1. `\mrule` and `\mstop` accept the same key.

`\mrule` Adds a rule of the form *<LHS>* $\leftarrow$ *<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. 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. Exactly one `\mstop` per model prefix is required before `\computemodel`. Without one, `\computemodel` issues a warning.

4.2 Expression reference

Table 2 lists all expression constructs for \mrule bodies. The **XMILE** column shows XMILE v1.0 syntax, **I3fp** 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 yet fully supported by numodel (the expression computes correctly, but the rendered keyword in the rule table is not yet translated). *Italic* cells contain a derived equivalent rather than a native keyword.

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

XMILE	I3fp (\fp_eval)	CoachTaal
<i>Arithmetic operators</i>		
+	+	+
-	-	-
*	*	*
/	/	/
^	^	^
MOD(x, y)	<i>x - trunc(x/y)*y</i>	<i>x - Entier(x/y)*y</i>
<i>Comparison operators</i>		
<	<	<
<=	<=	<=
>	>	>
>=	>=	>=
=	=	=
<>	!=	<>
<i>Boolean operators</i>		
AND	&&	EN
OR		OF
NOT	!	NIET
<i>Control flow</i>		
IF c THEN a ELSE b	c ? a : b	Als c Dan a Anders b EindAls
<i>General math functions</i>		
ABS(x)	abs(x)	Abs(x)
SIGN(x)	sign(x)	Teken(x)
SQRT(x)	sqrt(x)	Sqrt(x)
INT(x)	<i>trunc(x)</i>	<i>Entier(Abs(x))*Teken(x)</i>
INT(x+0.5)	<i>round(x)</i>	<i>Round(x)</i>
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)
—	<i>fact(x)</i>	<i>Fac(x)</i>
INT(LOG10(ABS(x)))	<i>logb(x)</i>	<i>Entier(Log(Abs(x)))</i>
<i>Exponential and logarithmic</i>		
EXP(x)	exp(x)	Exp(x)
LN(x)	ln(x)	Ln(x)
LOG10(x)	<i>ln(x)/ln(10)</i>	<i>Log(x)</i>
<i>Trigonometry (radians)</i>		
SIN(x)	sin(x)	Sin(x)
COS(x)	cos(x)	Cos(x)
TAN(x)	tan(x)	Tan(x)

Continued on next page

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

XMILE	l3fp (\fp_eval)	CoachTaal
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)
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)
<i>Trigonometry (degrees)</i>		
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)
<i>Constants</i>		
PI	pi	Pi
e	exp(1)	Exp(1)
INF	inf	—
NAN	nan	—
Pi/180	deg	Pi/180
1	true	Aan
0	false	Uit
<i>Simulation-specific functions (XMILE only)</i>		
TIME	user variable (\mvar)	user variable
DT	user variable (\mvar)	user variable
STEP(h, t ₀)	T >= t ₀ ? h : 0	Als T >= t ₀ Dan h Anders 0 EindAls
RAMP(s, t ₀)	T > t ₀ ? s*(T-t ₀) : 0	Als T > t ₀ Dan s*(T-t ₀) 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`, and `tblrenv=tblrlongtblrtalltblr`. Each per-call key overrides the global `\numodelsetup` setting for this single render only; the global state is restored afterwards.

`tblrenv` selects which `tabularray` environment wraps the table: `longtblr` (default) breaks across pages, `tblr` renders inline without page breaks, `talltblr` renders inline but with caption and note support. Use `tblr` or `talltblr` when `\textmodel` sits inside an enclosing environment that suppresses page breaks (`subfigure`, `minipage`, ...); the default `longtblr` would otherwise emit spurious continuation markers there.

Row spacing. `numodel` loads `tabularray` and sets `\SetTblrInner{rowsep=0pt}` globally so the rule listing renders compactly. This applies to *every* `tabularray` table in the document; if you want the default spacing back in your own tables, issue `\SetTblrInner{rowsep=2pt}` (or whatever value you prefer) somewhere in your document.

`\graphicmodel` Renders the Forrester stock-and-flow diagram. Variables of type `stock` become rectangles, `c` constant become circles, `aux` become identifier nodes; 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) and `diagram-style=tightforresteredu`, which overrides 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 $t \geq 1$ 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 $t = 1, 2, \dots$, 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 (10³)"). 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. $U_{3\text{ptc}}$ and $U_{3\text{pw}}$. 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}
```

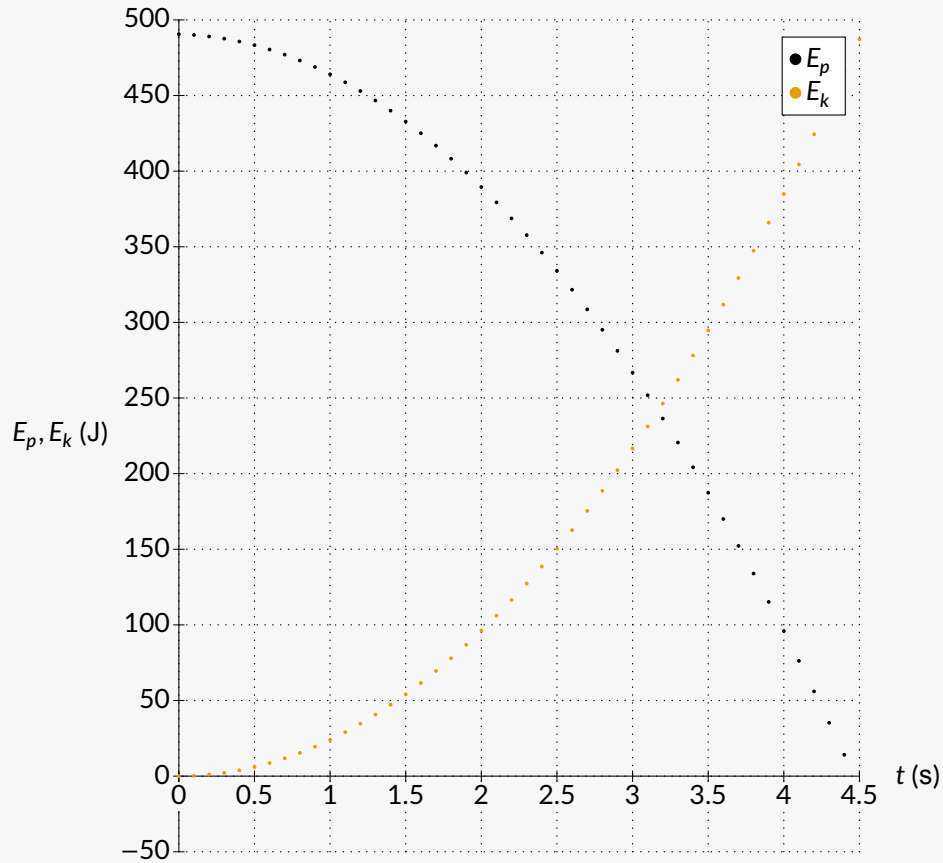


Figure 2: $E_p, E_k(t)$ -diagram

4.4 Series accessors

`\mcoords` Returns a comma-separated PGFPlots coordinate list of the simulated series for two variables. 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_p{<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 ($0 \dots \langle \text{prefix} \rangle \text{steps} - 1$ for non-negative indices, or $-\langle \text{prefix} \rangle \text{steps} \dots -1$ 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 `\computemodel`) – 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 nummodel's rounding: `num={T={round-mode=places,round-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 `\numodelsetup`), so they override it. A macro holding the options may be given.

prefix Another model than the current one.

```
\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 $\langle Var \rangle$ never reaches $\langle value \rangle$.

“Reaches” follows the direction in which $\langle Var \rangle$ starts: from below the value it is the first step at or above it, from above the first step at or below it. So $\langle Var \rangle$ 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}
```

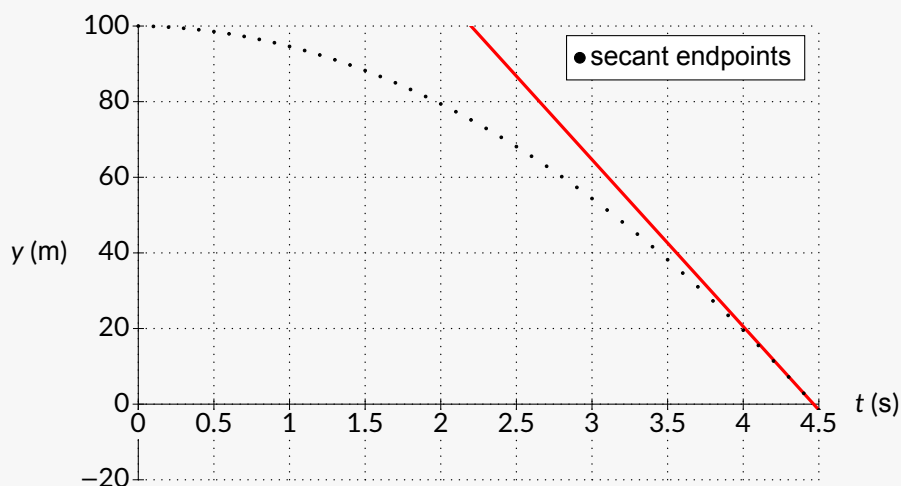


Figure 3: $y(t)$ -diagram

4.5 Namespace management

<code>\newmodelprefix</code>	Creates a new model namespace and switches to it. Signature: <code>\newmodelprefix{<name>}</code> . Subsequent <code>\mvar</code> / <code>\mrule</code> / <code>\mstop</code> calls bind to this prefix. All generated accessors are prefixed (so <code>\mvar{Y}{y}{...}{...}{...}{...}</code> under prefix <code>ball</code> produces <code>\ballY</code> , <code>\ballYtext</code> , etc.).
<code>\switchmodelprefix</code>	Switches to a previously created prefix. Useful when a document defines several models early and renders them later out of order. Signature: <code>\switchmodelprefix{<name>}</code> .
<code>\NumodelForEachVar</code>	Iterates a token list over every registered variable across every known prefix. Signature: <code>\NumodelForEachVar{<code with #1>}</code> ; inside the body, #1 is the full prefixed name (e.g. <code>ballY</code>). 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=3, 0]` – 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 $R \rightarrow V_a$, and a curved branch $R \rightarrow V_b$ arcing over V_a . (Note: TeX control words do not accept digits, so the macro names V_a/V_b 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.

Two flow-classification refinements support these layouts:

- The flow-variable heuristic prefers `aux` and `stock` variables over constants when both appear in the same additive term, so an inflow valve carries the meaningful rate variable instead of an earlier-declared scaling constant.
- Top-level parenthesised inflow terms of the form $(A - B) * C$ are distributed into $+A * C$, $-B * C$ before classification, so A surfaces as the inflow valve and B (typically a radiative or dissipative loss involving a constant) surfaces as the outflow valve.

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. Receives an *initial values* row.

aux An auxiliary variable computed from other variables on each step. Drawn as a plain identifier node, no rectangle. No *initial values* row (start value is normally left empty).

system System-level bookkeeping (time T , step size Dt , terminal time, ...). Drawn separately, no flow arrows. 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. Empty before `\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.

Example – the alias keys exist primarily as a worksheet hook: the same model can be rendered as a fully-specified reference *and* as a fill-in-the-blanks exercise. `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 $y = \dots$ (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	$v = v + g \cdot dt$	$t = 0\text{ s}$
2	$y = y + v \cdot dt$	$dt = 0.50\text{ s}$
3	$t = t + dt$	$v = 0\text{ m s}^{-1}$
4	IF $y \leq 0$ THEN STOP ENDIF	$y = \dots$

Continued on next page

(Continued)

	model	initial values
		?

6.1 Pupil's and answer version: blank

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 “ $\langle symbol \rangle = \dots$ ” (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 “ $\langle symbol \rangle = \dots$ ”. 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`.

```

\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	$v = \dots$	$t = 0\text{ s}$
2	$t = t + dt$	$dt = 0.50\text{ s}$
3	IF ... THEN STOP ENDIF	$v = 0\text{ m s}^{-1}$ $g = \dots$

	model	initial values
1	$v = v + g \cdot dt$	$t = 0\text{ s}$
2	$t = t + dt$	$dt = 0.50\text{ s}$
3	IF $t \geq 3$ THEN STOP ENDIF	$v = 0\text{ ms}^{-1}$ $g = -9.81\text{ ms}^{-2}$

Compared with the alias keys, `blank` needs no replacement text and switches between both versions; the alias keys remain for any other replacement (a question mark, a hint).

6.2 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_1=<Name>` puts it: before or after the rule for that variable. `\mrule text` 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.

\mremove rule[<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>_1=<value>` sets a start value; `blank={<Name>, ...}` gives the rules for these targets the `blank` key (Section 6.1). 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.

`\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. 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.

7 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 Requirements

numodel requires LuaLaTeX (the engine, for the Lua runtime) and TeX Live 2022 or later. Mandatory dependencies: `expl3`, `xparse`, `l3keys2e`, `amsmath`, `amssymb`, `xfrac`, `tikz`, `luacode`, `siunitx`, `float`, 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`.