Package 'sdbuildR'

August 23, 2026

Title An Accessible Interface for Stock-and-Flow Modelling

Version 2.2.3

Description Stock-and-flow models are a computational method from the field of system dynamics. They represent how systems change over time and are mathematically equivalent to ordinary differential equations. 'sdbuildR' (system dynamics builder) provides an intuitive interface for constructing stock-and-flow models without requiring extensive domain knowledge. Models can quickly be simulated and revised, supporting iterative development. 'sdbuildR' simulates models in 'R' and 'Julia', and supports computationally intensive ensemble simulations. Additionally, 'sdbuildR' can import models created in 'Insight Maker' (<https://insightmaker.com/>).

License GPL (>= 3)

URL https://kcevers.github.io/sdbuildR/, https://github.com/KCEvers/sdbuildR

BugReports https://github.com/KCEvers/sdbuildR/issues

Depends R (>= 4.2.0)

Imports cli, data.table, deSolve, DiagrammeR, future, future.apply, htmltools, htmlwidgets, igraph, jsonlite, JuliaConnectoR, plotly, progressr, rlang, stringi, stringr, textutils, tools, utils, withr, xml2

Suggests callr, gt, DiagrammeRsvg, kableExtra, knitr, qgraph, rsvg, testthat (>= 3.0.0), this.path, webshot2, bslib

Config/Needs/website rmarkdown, lavaan, shiny

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.0.0

RoxygenNote 8.0.0

NeedsCompilation no

Author Kyra Caitlin Evers [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-6890-3482>),
STIX Fonts Project Authors [ctb, cph] (Bundled STIX Two Text font
(inst/fonts/), SIL Open Font License 1.1)

Maintainer Kyra Caitlin Evers <kyra.c.evers@gmail.com>

Repository CRAN

Date/Publication 2026-08-23 11:30:02 UTC

Contents

`as.data.frame.ensemble_stockflow`
Create data frame of simulation results

Description

Convert simulation results to a data.frame.

Usage

```
## S3 method for class 'ensemble_stockflow'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  which = c("summary", "sims")[1],
  direction = "long",
  sim = NULL,
  condition = NULL,
  vars = NULL,
  type = NULL,
  ...
)
```

Arguments

x	Output of simulate().
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	Ignored parameter.
which	Type of data to return. Either "summary" for a summary statistics, or "sims" for individual simulation trajectories. Defaults to "summary".
direction	Format of data frame, either "long" (default) or "wide".
sim	Indices of the individual trajectories to include if which = "sims". Defaults to NULL, which includes all trajectories. Including a high number of trajectories will create a large object.
condition	Indices of the conditions to include. Defaults to NULL, which includes all conditions.
vars	Variables to plot. Defaults to NULL to plot all variables.
type	Variable types to retain in the data frame. Must be one or more of 'stock', 'flow', 'constant', 'aux', 'lookup', or 'func'. Defaults to NULL to include all types.
...	Optional parameters

Value

A data.frame with simulation results. For direction = "long" (default), the data frame has three columns: time, variable, and value. For direction = "wide", the data frame has columns time followed by one column per variable.

See Also

ensemble(), stockflow()

Examples

```
sfm <- stockflow("sir") |>
  # Randomize initial values of all stocks to show variation in the ensemble
  update(c(susceptible, infected, recovered),
    eqn = runif(1, min = 20, max = 800)
  )

# Run ensemble simulation with 3 simulations,
# saving only 20 timepoints per simulation
sims <- ensemble(sfm, n = 3, save_length = 20, save_sims = TRUE)

# Get summary statistics in long format
df <- as.data.frame(sims)
head(df, n = 1)

# Get summary statistics in wide format
df_wide <- as.data.frame(sims, direction = "wide")
head(df_wide, n = 1)

# Get individual simulations in wide format
df_wide_sims <- as.data.frame(sims, which = "sims", direction = "wide")
head(df_wide_sims, n = 1)
```

---

as.data.frame.simulate_stockflow

Create data frame of simulation results

Description

Convert simulation results to a data.frame.

Usage

```
## S3 method for class 'simulate_stockflow'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  direction = "long",
  vars = NULL,
  type = NULL,
  ...
)
```

Arguments

x	Output of simulate().

row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	Ignored parameter.
direction	Format of data frame, either "long" (default) or "wide".
vars	Variable names to retain in the data frame. Defaults to NULL to include all variables.
type	Variable types to retain in the data frame. Must be one or more of 'stock', 'flow', 'constant', 'aux', 'lookup', or 'func'. Defaults to NULL to include all types.
...	Optional parameters

Value

A data.frame with simulation results. For direction = "long" (default), the data frame has three columns: time, variable, and value. For direction = "wide", the data frame has columns time followed by one column per variable.

See Also

simulate(), stockflow()

Examples

```
sfm <- stockflow("sir")
sim <- simulate(sfm)
df <- as.data.frame(sim)
head(df)

# Get results in wide format
df_wide <- as.data.frame(sim, direction = "wide")
head(df_wide)
```

---

as.data.frame.stockflow

Convert stock-and-flow model to data frame

Description

Create a data frame with properties of all model variables and functions. Specify the variable types, variable names, and/or properties to get a subset of the data frame.

Usage

```
## S3 method for class 'stockflow'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  vars = NULL,
  type = NULL,
  properties = NULL,
  ...
)
```

Arguments

x	A stock-and-flow model object of class stockflow.
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	Ignored parameter.
vars	Variable names to retain in the data frame. Defaults to NULL to include all variables.
type	Variable types to retain in the data frame. Must be one or more of 'stock', 'flow', 'constant', 'aux', 'lookup', or 'func'. Defaults to NULL to include all types.
properties	Variable properties to retain in the data frame. Defaults to NULL to include all properties.
...	Optional arguments

Value

A data.frame with one row per model component. Common columns include type (component type), name (variable name), eqn (equation), and label (descriptive label). Additional columns may include to, from, non_negative, and others depending on variable types. The exact columns returned depend on the type and properties arguments. Returns an empty data.frame if no components match the filters.

Examples

```
as.data.frame(stockflow("sir"))

# Only show stocks
as.data.frame(stockflow("sir"), type = "stock")

# Only show specific variables
as.data.frame(stockflow("sir"), vars = c("susceptible", "infected"))

# Only show equation and label
as.data.frame(stockflow("sir"), properties = c("eqn", "label"))
```

`as.data.frame.verify_stockflow` *Convert verify() results to a data frame*

Description

Converts a `verify_stockflow` object to a data frame.

Usage

```
## S3 method for class 'verify_stockflow'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  which = c("tests", "sims")[1],
  direction = "long",
  test = NULL,
  label = NULL,
  ignore_case = TRUE,
  status = c("pass", "fail", "error", "skip"),
  condition = NULL,
  vars = NULL,
  type = NULL,
  ...
)
```

Arguments

`x`	A `verify_stockflow` object (output of `verify()`).
`row.names`	`NULL` or a character vector giving row names (optional).
`optional`	Ignored; present for compatibility.
`which`	Character. `"tests"` (default) or `"sims"`. Partial matching supported.
`direction`	Character. `"long"` (default) or `"wide"`. Only used when `which = "sims"`.
`test`	Integer vector of test number(s) to display (1-based). Defaults to `NULL` (show all tests). Can be combined with `label` (intersection).
`label`	Character vector of regex patterns for partial, case-insensitive label matching. A test is included if its label matches *any* pattern. E.g., `c("non-neg", "beta")` returns tests matching either fragment. Can be combined with `test` (intersection).
`ignore_case`	Logical; whether `label` matching is case-insensitive. Default `TRUE`.
`status`	Optional character vector of statuses to include (e.g., `c("fail", "error")`). Defaults to all statuses. • `which = "tests"`: filters rows by test status.

- `which = "sims"`: filters to conditions that have at least one test with a matching status.

`condition` Optional integer vector of condition numbers to filter by. For `which = "sims"`, keeps only the matching condition simulations. For `which = "tests"`, keeps only tests belonging to those conditions.

`vars` Variable names to retain in the data frame. Only applies when `which = "sims"`. Defaults to `NULL` to include all variables.

`type` Variable types to retain in the data frame. Must be one or more of 'stock', 'flow', 'constant', 'aux', 'lookup', or 'func'. Only applies when `which = "sims"`. Defaults to `NULL` to include all types.

`...` Additional arguments (unused).

Details

`which = "tests"` **(default)** returns one row per unit test with columns `test`, `label`, `status`, `outcome`, `expr_str`, `conditions`, and `message`. Use `test`, `label`, and `status` to filter.

`which = "sims"` returns the underlying simulation time-series in long format with columns `test` (test number(s) that used this simulation, as a comma-separated string), `conditions` (specified conditions per test, if any), `time`, `variable`, and `value`. Each unique condition generates one simulation; if multiple tests share a condition their numbers are combined in `test` (e.g. `"1, 3"`). When filtering with `test`, the displayed `test` value shows only the requested matching test number(s) for the retained simulation row(s). Use `direction = "wide"` to pivot variables into columns.

Value

A `data.frame`. Column set depends on `which`:

- `"tests"`: `test`, `label`, `status`, `outcome`, `expr_str`, `condition`, `conditions`, `message`.
- `"sims"` (long): `test`, `condition`, `conditions`, `time`, `variable`, `value`.
- `"sims"` (wide): `test`, `condition`, `conditions`, `time`, then one column per variable.

Examples

```
# Create model with 2 unit tests
sfm <- stockflow("sir") |>
  unit_test(expr = all(susceptible >= 0)) |>
  # Add test with conditions
  unit_test(
    label = "lower infection rate",
    expr = all(susceptible >= 0),
    conditions = list(infection_rate = 0.1)
  )
res <- verify(sfm)

# Test results (default)
as.data.frame(res)

# Simulation time-series (long format)
as.data.frame(res, which = "sims") |> head()
```

```
# Simulation time-series (wide format)
as.data.frame(res, which = "sims", direction = "wide") |> head()

# Filter to simulation for test 2 only
as.data.frame(res, which = "sims", test = 2) |> head()

# Only simulations for passing tests
as.data.frame(res, which = "sims", status = "pass") |> head()
```

`auxiliary` *Add or modify auxiliaries*

Description

Auxiliaries are dynamic variables used for intermediate calculations in the system. `auxiliary()` adds or changes an auxiliary variable. This is a convenience wrapper around `update()` with `type = "aux"`. See the **Auxiliaries** section of `update()` for more details.

Usage

```
auxiliary(object, name, eqn = 0, label = name, doc = "", non_negative = FALSE)

aux(object, name, eqn = 0, label = name, doc = "", non_negative = FALSE)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`name`	Variable name. Accepts a bare symbol (e.g., `population`), a string (`"population"`), or a vector via `c()` (e.g., `c(a, b)` or `c("a", "b")`). Use `!!` to inject from a variable.
`eqn`	Equation (or initial value in the case of stocks). Accepts a bare expression (e.g., `a * b + 1`), a string (`"a * b + 1"`), or a numeric value. Use `!!` to inject from a variable. Defaults to `0`.
`label`	Name of variable used for plotting. Defaults to the same as name.
`doc`	Description of variable. Defaults to `""` (no description).
`non_negative`	If TRUE, variable is enforced to be non-negative (i.e., strictly 0 or positive). Defaults to `FALSE`.

Value

A stock-and-flow model object of class `stockflow`

See Also

`update()`, `discard()`, `change_name()`

Examples

```
# Create an auxiliary for an intermediate calculation
sfm <- stockflow() |>
  stock(population, eqn = 100) |>
  constant(carrying_capacity, eqn = 1000) |>
  auxiliary(density, eqn = population / carrying_capacity, label = "Density")
```

---

change_name *Change name of variable*

Description

Change the name of a variable throughout the model. This updates the data frame and all references in equations, flow connections, and labels.

Usage

```
change_name(object, name, new_name)
```

Arguments

object	Stock-and-flow model, object of class stockflow.
name	Variable name. Accepts a bare symbol (e.g., population), a string ("population"), or a vector via c() (e.g., c(a, b) or c("a", "b")). Use !! to inject from a variable.
new_name	New name. Character vector of the same length as name. Must be unique across all existing variables.

Value

A stock-and-flow model object of class stockflow with the name changed throughout the model.

See Also

update(), discard()

Examples

```
sfm <- stockflow("sir")
sfm <- change_name(sfm, c(susceptible, infected, recovered),
  new_name = c(S, I, R)
)
print(sfm)

# References to old names are updated
as.data.frame(sfm, type = "flow", properties = c("name", "eqn", "to", "from"))
```

change_type *Change variable type*

Description

Change the type of a variable in a stock-and-flow model.

Usage

```
change_type(object, name, new_type)
```

Arguments

object	Stock-and-flow model, object of class stockflow.
name	Variable name. Accepts a bare symbol (e.g., population), a string ("population"), or a vector via c() (e.g., c(a, b) or c("a", "b")). Use !! to inject from a variable.
new_type	New variable type; one of 'stock', 'flow', 'constant', 'aux', 'lookup', or 'func'. Character vector of the same length as name.

Value

A stock-and-flow model object of class stockflow with the variable type changed throughout the model. Note that changing the type may result in changes to other properties (e.g., a flow must have "to" and/or "from" properties, so these will be added if not already present), and may require changes to the equations of connected variables.

See Also

update()

Examples

```
# Start with a simple predator-prey model
sfm <- stockflow("predator_prey")

# Change the birth rate of predators from a constant to an auxiliary
sfm <- change_type(sfm, delta, new_type = "aux")

# This allows us to introduce time-dependence in the birth rate
# (e.g., seasonality with a sine function)
sfm <- sfm |>
  update(delta, eqn = 0.025 + 0.01 * sin(2 * pi * t / 12))
sim <- simulate(sfm)
plot(sim)
```

compare_models *Compare two stock-and-flow models*

Description

Compares the structure, equations, and simulation settings of two `stockflow` models.

Usage

```
compare_models(sfm1, sfm2)
```

Arguments

`sfm1`	A stock-and-flow model of class `stockflow`.
`sfm2`	A stock-and-flow model of class `stockflow`.

Value

An object of class `compare_stockflow` (a list) containing:

`labels` Names of the two model objects (captured expressions).

`added` Variables present in `sfm2` but not `sfm1`.

`removed` Variables present in `sfm1` but not `sfm2`.

`type_changed` Variables with different types.

`eqn_changed` Variables with different equations.

`sim_settings_diff` Simulation settings that differ.

`properties` Per-model counts of variables by type.

See Also

`simulate()`, `summary()`

Examples

```
sfm1 <- stockflow("sir")
sfm2 <- stock(sfm1, "susceptible", eqn = 0.5)
compare_models(sfm1, sfm2)
```

`constant` *Add or modify constants*

Description

Constants are time-independent variables that do not change over the course of a simulation. `constant()` adds or changes a constant variable. This is a convenience wrapper around `update()` with `type = "constant"`. See the **Constants** section of `update()` for more details.

Usage

```
constant(object, name, eqn = 0, label = name, doc = "", non_negative = FALSE)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`name`	Variable name. Accepts a bare symbol (e.g., `population`), a string (`"population"`), or a vector via `c()` (e.g., `c(a, b)` or `c("a", "b")`). Use `!!` to inject from a variable.
`eqn`	Equation (or initial value in the case of stocks). Accepts a bare expression (e.g., `a * b + 1`), a string (`"a * b + 1"`), or a numeric value. Use `!!` to inject from a variable. Defaults to `0`.
`label`	Name of variable used for plotting. Defaults to the same as name.
`doc`	Description of variable. Defaults to `""` (no description).
`non_negative`	If TRUE, variable is enforced to be non-negative (i.e., strictly 0 or positive). Defaults to `FALSE`.

Value

A stock-and-flow model object of class `stockflow`

See Also

`update()`, `discard()`, `change_name()`

Examples

```
# Create constants for model parameters
sfm <- stockflow() |>
  constant(growth_rate, eqn = 0.1, label = "Growth Rate") |>
  constant(carrying_capacity, eqn = 1000, label = "Carrying Capacity")
```

contains_IM *Check whether value is in vector or string*

Description

Equivalent of .Contains() in Insight Maker.

Usage

```
contains_IM(haystack, needle)
```

Arguments

haystack	Vector or string to search through
needle	Value to search for

Value

Logical value

Examples

```
contains_IM(c("a", "b", "c"), "d") # FALSE
contains_IM(c("abcdef"), "bc") # TRUE
```

custom_func *Create or modify custom variables or functions*

Description

Custom functions are user-defined functions that can be used throughout a stock-and-flow model. `custom_func()` adds or changes a function. This is a convenience wrapper around `update()` with `type = "func"`.

Usage

```
custom_func(object, name, eqn = 0, label = name, doc = "")
```

Arguments

object	Stock-and-flow model, object of class `stockflow`.
name	Name of the function variable. The equation will be assigned to this name.
eqn	Equation of the function variable. A character vector. Defaults to `0`.
label	Name of variable used for plotting. Defaults to the same as name.
doc	Documentation. Defaults to "".

Value

A stock-and-flow model object of class stockflow

See Also

update(), discard(), change_name()

Examples

```
# Simple function
sfm <- stockflow() |>
  custom_func(double, eqn = "function(x) x * 2") |>
  constant(a, eqn = double(2))

# Function with defaults
sfm <- stockflow() |>
  custom_func(scale, eqn = "function(x, factor = 10) x * factor") |>
  constant(b, eqn = scale(2))

# If the logistic() function did not exist, you could create it yourself:
sfm <- stockflow() |>
  custom_func(my_logistic, eqn = "function(x, slope = 1, midpoint = .5){
   1 / (1 + exp(-slope*(x-midpoint)))
 }") |>
  constant(c_, eqn = my_logistic(2, slope = 50))
```

---

dependencies *Find dependencies*

Description

Find which other variables each variable is dependent on.

Usage

```
dependencies(object, name = NULL, type = NULL, reverse = FALSE)
```

Arguments

object	Stock-and-flow model, object of class stockflow.
name	Variable names to find dependencies for. Defaults to NULL to include all variables.
type	Variable types to find dependencies for. Must be one or more of 'stock', 'flow', 'constant', 'aux', 'lookup', or 'func'. Defaults to NULL to include all types.
reverse	If FALSE, list for each variable X which variables Y it depends on for its equation definition. If TRUE, don't show dependencies but dependents. This reverses the dependencies, such that for each variable X, it lists what other variables Y depend on X.

Value

List, with for each model variable what other variables it depends on, or if `reverse = TRUE`, which variables depend on it

Examples

```
sfm <- stockflow("sir")
dependencies(sfm)
```

`discard` *Remove variable(s)*

Description

Remove variable(s) from a stock-and-flow model. All references in flow connections and graphical function sources are also removed. A warning will be thrown if any lingering references to the removed name remain in the model.

Usage

```
discard(
  object,
  name,
  remove_references = c("to", "from", "source", "unit_test")
)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`name`	Name(s) to remove. Accepts bare symbols (e.g., `x`), strings, or vectors via `c()`. Must be variable names.
`remove_references`	Where to remove references to the discarded variables. By default, references to discarded variables in `"to"`, `"from"`, `"source"`, and `"unit_test"` are removed. Set to `NULL` to keep all references (not recommended). Note that any lingering references in equations will cause errors in simulation and should be removed or updated with `update()` after discarding the variable.

Value

A stock-and-flow model object of class `stockflow`

See Also

`update()`, `change_name()`

Examples

```
# Add stock
sfm <- stockflow() |> stock(x)
print(sfm)

# Remove stock
sfm <- discard(sfm, x)
print(sfm)
```

`discard_unit_test` *Remove a unit test from a stock-and-flow model*

Description

Remove one or more unit tests by `test` (integer position as shown by `unit_tests()`) or by `label` (character). Warns if a label or index is not found. Remaining tests are renumbered sequentially after removal.

Usage

```
discard_unit_test(object, label, test)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`label`	Character label(s) of the test(s) to remove. Supports NSE (bare symbol or string). For backward compatibility, integer values passed via `label` are also accepted.
`test`	Integer index/indices of the test(s) to remove. Corresponds to the order shown by `unit_tests()`.

Value

The model object with the specified test(s) removed.

See Also

`unit_test()`, `unit_tests()`

Examples

```
sfm <- stockflow("sir") |>
  unit_test(label = "susceptible is non-negative", expr = all(susceptible >= 0)) |>
  unit_test(label = "recovered increases", expr = all(diff(recovered) >= 0))

# Remove by test
sfm <- discard_unit_test(sfm, test = 1)
```

```
# Remove by label
sfm <- discard_unit_test(sfm, label = "recovered increases")
```

`ensemble` *Run ensemble simulation*

Description

Run large-scale (i.e., ensemble) simulations of stock-and-flow models, varying initial conditions and/or constants specified in `conditions`.

Usage

```
ensemble(
  object,
  n = 10,
  conditions = NULL,
  cross = TRUE,
  central,
  spread,
  quantiles,
  quiet = FALSE,
  ...
)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`n`	Number of simulations to run in the ensemble. When conditions is specified, n defines the number of simulations to run per condition. If each condition only needs to be run once, set `n = 1`. Defaults to `n = 10`.
`conditions`	A named list specifying **fixed** values for ensemble conditions. Names must correspond to stocks or constants in the model. Each list element should be a numeric vector of values to test. If cross = `TRUE` (default), all combinations of values are generated. For example, `list(param1 = c(1, 2), param2 = c(10, 20))` creates 4 conditions: `(1,10)`, `(1,20)`, `(2,10)`, `(2,20` If cross = `FALSE`, values are paired element-wise, requiring all vectors to have equal length. For example, `list(param1 = c(1, 2, 3), param2 = c(10, 20, 30))` creates 3 conditions: `(1,10)`, `(2,20)`, `(3,30)`. Defaults to `NULL` (no parameter variation).
`cross`	If `TRUE`, cross the parameters in the conditions list to generate all possible combinations of parameters. Defaults to `TRUE`.
`central`	Which central-tendency statistic(s) to compute across the simulations at each time point: any of `"mean"`, `"median"`, or `"none"`. Each one becomes a column of the same name in `summary`. Defaults to the model's `central` setting (see `sim_settings()`); set it here to override that for this run. Note that `central` and

	spread choose what is *computed*; plot.ensemble_stockflow() then chooses what to *show*.
spread	Which measures of spread to compute: "quantile" (columns quant1, quant2, ... at the probabilities in quantiles), "sd", "range" (a min and a max column), or "none". Several can be combined, e.g. c("quantile", "sd"). Defaults to the model's spread setting.
quantiles	Probabilities for the quantile columns, used when spread includes "quantile", e.g. c(0.025, 0.975). They become columns quant1, quant2, ... in the order given (so here quant1 is the 2.5% quantile and quant2 the 97.5%); the probabilities themselves are kept in sims$quantiles. Defaults to the model's quantiles setting.
quiet	If TRUE, suppress informational messages such as progress and status updates. Warnings and errors are always shown. Defaults to FALSE. R ensembles show a cli progress bar via progressr::handler_cli(), including when simulations run through a parallel future::plan().
...	Optional arguments passed to sim_settings(); these can be used to override the simulation specifications set in the model object.

Details

It is strongly recommended to reduce the size of the simulation output by saving fewer values with save_by, save_times, or save_length in sim_settings().

By default, only summary statistics across simulations are returned. To return individual simulations, set save_sims = TRUE in sim_settings() or pass save_sims = TRUE via ... in ensemble(). Note that returning individual simulations can consume a lot of memory for large ensembles.

For simulations in Julia, the ensemble can be run in parallel using multiple threads by setting nthreads in use_julia(). For simulations in R, use future::plan() to control parallel execution.

To create a reproducible ensemble simulation, set a seed using sim_settings().

If you do not see any variation within a condition of the ensemble (i.e., the confidence bands are virtually non-existent), there are likely no random elements in your model. Without these, there can be no variability in the model. Try specifying a random initial condition or adding randomness to other model elements (see examples).

Value

Object of class ensemble_stockflow, which is a list containing:

success If TRUE, simulation was successful. If FALSE, simulation failed.

error_message If success is FALSE, contains the error message.

df data.frame with simulation results in long format, if save_sims is TRUE. The iteration number is indicated by column "sim". If conditions was specified, the condition is indicated by column "condition".

summary data.frame with summary statistics of the ensemble. Contains the statistics requested via central and spread (as columns named after each statistic, plus quant1, quant2, ... when spread includes "quantile"), as well as a missing_count column. If conditions was specified, summary statistics are calculated for each condition in the ensemble.

n Number of simulations run in the ensemble (per condition if conditions is specified).

n_total Total number of simulations run in the ensemble (across all conditions if conditions is specified).

n_conditions Total number of conditions.

conditions data.frame with the conditions used in the ensemble, if conditions was specified.

init List with df (if save_sims = TRUE) and summary, containing data.frame with the initial values of the stocks used in the ensemble.

constants List with df (if save_sims = TRUE) and summary, containing data.frame with the constant parameters used in the ensemble.

script Script used for the ensemble simulation.

duration Duration of the simulation in seconds.

... Other parameters passed to ensemble

See Also

update(), stockflow(), sim_settings(), use_julia(), future::plan()

Examples

```
# Ensemble simulation in R (no parallelization)
# Load example model
sfm <- stockflow("predator_prey")

# Ensemble simulations can only show variation
# if there is some randomness in the model.
# For example, we can initialize both stocks with
# random values between 20 and 80:
sfm <- update(sfm, c(predator, prey),
  eqn = runif(1, min = 20, max = 80)
)

# Saving all timepoints is computationally expensive,
# so we save only 20 values per simulation:
sfm <- sim_settings(sfm, save_length = 20)

# Run ensemble simulation
sims <- ensemble(sfm, n = 3)
plot(sims)


# To plot individual trajectories, rerun the ensemble with save_sims = TRUE.
# Note that this can consume a lot of memory for large simulations.
sims <- ensemble(sfm, n = 10, save_sims = TRUE)
plot(sims, which = "sims")

# Specify which trajectories to plot
plot(sims, which = "sims", sim = 1:2)

# Don't plot the central tendency
```

```
# with darker individual trajectories
plot(sims, central = "none", which = "sims", alpha = 0.7)

# We can speed up ensemble simulations using parallelization
future::plan(future::multisession, workers = 2)
sims <- ensemble(sfm, n = 25, save_sims = TRUE)

# Ensembles can also be run with exact values for the initial conditions
# and parameters. Below, we vary the initial values of the predator and the
# birth rate of the predators (delta). We generate a hundred samples per
# condition. By default, the parameters are crossed, meaning that all
# combinations of the parameters are run.
sims <- ensemble(sfm,
  n = 50,
  conditions = list(
    predator = c(10, 50),
    delta = c(.025, .05)
  )
)

plot(sims)

# By default, a maximum of nine conditions is plotted.
# Plot specific conditions:
plot(sims, condition = c(1, 3), nrows = 1)

# Generate a non-crossed design, where the length of each conditions vector
# needs to be equal:
sims <- ensemble(sfm,
  n = 10, cross = FALSE,
  conditions = list(
    predator = c(10, 20, 30),
    delta = c(.020, .025, .03)
  )
)
plot(sims, nrows = 3)

# Stop parallelization after use
future::plan(future::sequential)
```

`expit` *Expit function*

Description

Inverse of the logit function

Usage

```
expit(x)
```

Arguments

x	Numerical value

Value

Numerical value

Examples

```
expit(1)
```

`export_model` *Export a stock-and-flow model*

Description

Export a model of class `stockflow` to another format.

Usage

```
export_model(
  object,
  format = c("sdbuildR", "deSolve", "psychomodels"),
  file = NULL,
  title = object[["meta"]][["name"]],
  description = object[["meta"]][["caption"]],
  explanation = description,
  publication_doi = "",
  publication_citation = "",
  framework = "Ordinary Differential Equations",
  programming_language = "R",
  psychology_discipline = "",
  software_package = "",
  model_variable = "",
  code_repository_url = "",
  data_url = "",
  submission_remarks = "",
  created_by = "",
  updated_by = "",
  published_by = "",
  published_at = Sys.time(),
  published_pending_moderation_at = Sys.time(),
  publication_citation_fetched_at = Sys.time(),
  publication_csl_fetched_at = Sys.time(),
  publication_csl_json = "",
  id = NA,
  slug = NULL,
```

```
  include_latex = TRUE,
  pretty = TRUE
)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`format`	Export format. One of `"sdbuildR"`, `"deSolve"`, or `"psychomodels"`.
`file`	Output file path, or `NULL` to return the result directly.
`title`	[psychomodels] Model title. Defaults to `object[["meta"]][["name"]]`.
`description`	[psychomodels] Model description. Defaults to `object[["meta"]][["caption"]]`.
`explanation`	[psychomodels] Free-text explanation. Defaults to `description`.
`publication_doi`	[psychomodels] DOI for the associated publication.
`publication_citation`	[psychomodels] Citation text.
`framework`	[psychomodels] Modeling framework. Defaults to `"Ordinary Differential Equations"`.
`programming_language`	[psychomodels] Programming language.
`psychology_discipline`	[psychomodels] Discipline id(s), comma-separated.
`software_package`	[psychomodels] Package id(s), comma-separated.
`model_variable`	[psychomodels] Variable id(s), comma-separated.
`code_repository_url`	[psychomodels] URL to code repository.
`data_url`	[psychomodels] URL to model data.
`submission_remarks`	[psychomodels] Optional remarks.
`created_by`	[psychomodels] Identifier of creating user.
`updated_by`	[psychomodels] Identifier of last updating user.
`published_by`	[psychomodels] Identifier of publishing user.
`published_at`	[psychomodels] Publication timestamp. Defaults to current time.
`published_pending_moderation_at`	[psychomodels] Moderation timestamp.
`publication_citation_fetched_at`	[psychomodels] Citation fetch timestamp.
`publication_csl_fetched_at`	[psychomodels] CSL fetch timestamp.
`publication_csl_json`	[psychomodels] CSL JSON text.

id	[psychomodels] Optional record id.
slug	[psychomodels] Optional slug. Generated from title if NULL.
include_latex	[psychomodels] If TRUE, append LaTeX equations to explanation.
pretty	[psychomodels] If TRUE, pretty-print output JSON.

Details

sdbuildR format (format = "sdbuildR"):
Returns R code that reconstructs the model using sdbuildR functions. When file = NULL, returns a character string. When file is provided, writes an .R file and returns the path invisibly. If file has no .R extension, one is appended.

deSolve format (format = "deSolve"):
Returns a standalone R script using deSolve::ode() directly — no sdbuildR dependency required to run the output. When file = NULL, returns a character string. When file is provided, writes an .R file and returns the path invisibly. If file has no .R extension, one is appended. Requires sim_settings(language = "R") (the default).

Psychomodels format (format = "psychomodels"):
Generates a JSON record for upload to Psychomodels. When file = NULL, returns a JSON character string. When file is provided, writes a .json file and returns the path invisibly. If file has no .json extension, one is appended.

Value

For file = NULL: a character string containing the exported content. For file specified: invisibly returns the file path.

See Also

import_insightmaker(), import_desolve()

Examples

```
sfm <- stockflow("sir")

# Get sdbuildR reconstruction code
cat(export_model(sfm, format = "sdbuildR"))

# Get standalone deSolve script
cat(export_model(sfm, format = "deSolve"))

# Export to Psychomodels JSON
## Not run:
json <- export_model(sfm,
  format = "psychomodels",
  publication_doi = "10.0000/example"
)

## End(Not run)
```

`export_plot` *Save plot to a file*

Description

Save a plot of a stock-and-flow diagram or a simulation to a specified file path. Note that saving plots requires additional packages to be installed (see below).

Usage

```
export_plot(
  pl,
  file,
  width = 3,
  height = 4,
  units = "cm",
  dpi = 300,
  font_family = NULL,
  close_browser = TRUE
)
```

Arguments

`pl`	Plot object. Can be a `grViz` object from the DiagrammeR package (for stock-and-flow diagrams) or a `plotly` object from the plotly package (for (ensemble) simulation results).
`file`	File path to save plot to, including a file extension. For plotting a stock-and-flow model, the file extension can be one of png, pdf, svg, ps, eps, webp. For plotting a simulation, the file extension can be one of png, pdf, jpg, jpeg, webp. If no file extension is specified, it will default to png.
`width`	Width of image in units.
`height`	Height of image in units.
`units`	Units in which width and height are specified. Either "cm", "in", or "px".
`dpi`	Resolution of image. Only used if units is not "px".
`font_family`	Font family to render the plot in, overriding the font the plot was created with. Kebab-case names (e.g. `"eb-garamond"`) are loaded as webfonts (see Details); any other name must be installed on your system. Defaults to `NULL`, which keeps the plot's own font.
`close_browser`	If `TRUE` (default), browser-based exports run in a separate short-lived R process, so the headless browser is closed when the export finishes and leaves no state behind in your R session. Set to `FALSE` to render in a persistent browser session instead, which is faster when exporting many plots in a row; the browser then stays open in the background until your R session ends. Ignored for exports that do not use a browser.

Details

Exports that render in a headless browser (plotly plots, and diagrams using a webfont) require the suggested packages webshot2 and callr, plus a Chrome-based browser on your system. See `close_browser` for how the browser's lifetime is managed.

Value

Invisibly returns the file path of the exported plot, called for side effects.

Fonts

When `font_family` (either passed here or to the `plot()` call that created `pl`) is an all-lowercase kebab-case name, it is treated as a font identifier from Fontsource (https://fontsource.org/) – browse the catalogue there and use the id from the font page's URL, e.g. `"eb-garamond"` or `"source-serif-4"`. The font is then loaded as a *webfont*: a CSS rule pointing to the font's files on the jsDelivr content delivery network is attached to the plot, and the headless browser that renders the export fetches the font over the internet at render time, just like a webpage. No fonts are downloaded by R or installed on your system. Internet access is required during the export; without it, the export still succeeds in a fallback font. Note that webfonts are not supported for the ps/eps formats, which fall back to system fonts.

Examples

```
## Not run:
# Exporting stock-and-flow diagrams requires the DiagrammeRsvg and rsvg packages
if (!requireNamespace("DiagrammeRsvg", quietly = TRUE)) {
  install.packages("DiagrammeRsvg")
}
if (!requireNamespace("rsvg", quietly = TRUE)) {
  install.packages("rsvg")
}

# Load example model
sfm <- stockflow("sir")

# Temporary file path to save the plot
file <- tempfile(fileext = ".svg")

# Use system font (the default webfont requires webshot2 and a headless browser)
export_plot(plot(sfm, font_family = "serif"), file)

# Remove plot
file.remove(file)

# Export a simulation plot; requires Chrome
if (has_internet()) {
  # Run a simulation
  sim <- simulate(sfm)

  # Temporary file path to save the plot
  file <- tempfile(fileext = ".pdf")
```

```
  export_plot(plot(sim), file)

  # Remove plot
  file.remove(file)
}

## End(Not run)
```

`flow` *Add or modify flows*

Description

Flows move material and information through the system, increasing or decreasing stocks. `flow()` adds or changes a flow variable. This is a convenience wrapper around `update()` with `type = "flow"`. See the **Flows** section of `update()` for more details.

Usage

```
flow(
  object,
  name,
  eqn = 0,
  to = NULL,
  from = NULL,
  label = name,
  doc = "",
  non_negative = FALSE
)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`name`	Variable name. Accepts a bare symbol (e.g., `population`), a string (`"population"`), or a vector via `c()` (e.g., `c(a, b)` or `c("a", "b")`). Use `!!` to inject from a variable.
`eqn`	Equation (or initial value in the case of stocks). Accepts a bare expression (e.g., `a * b + 1`), a string (`"a * b + 1"`), or a numeric value. Use `!!` to inject from a variable. Defaults to `0`.
`to`	Target of flow. Accepts a bare symbol or string. Must be a stock in the model. Defaults to `NULL` to indicate no target.
`from`	Source of flow. Accepts a bare symbol or string. Must be a stock in the model. Defaults to `NULL` to indicate no source.
`label`	Name of variable used for plotting. Defaults to the same as name.
`doc`	Description of variable. Defaults to `""` (no description).
`non_negative`	If TRUE, variable is enforced to be non-negative (i.e., strictly 0 or positive). Defaults to `FALSE`.

Value

A stock-and-flow model object of class stockflow

See Also

update(), discard(), change_name()

Examples

```
# Create a flow into a stock
sfm <- stockflow() |>
  stock(population, eqn = 100) |>
  flow(births, eqn = population * 0.1, to = population) |>
  flow(deaths, eqn = population * 0.05, from = population)
```

---

head.simulate_stockflow

Print first rows of a simulation

Description

Print the first rows of a simulation data frame of a stock-and-flow model. This is a wrapper around head() that first converts the simulation results to a data frame using as.data.frame().

Usage

```
## S3 method for class 'simulate_stockflow'
head(x, n = 6L, ...)
```

Arguments

x	Output of simulate().
n	Number of rows to print. Defaults to 6.
...	Other arguments passed to as.data.frame.simulate_stockflow().

Value

A data.frame with the first rows of the simulation results.

Examples

```
sfm <- stockflow("sir")
sim <- simulate(sfm)
head(sim)
```

head.verify_stockflow *Print first rows of verify results*

Description

Wrapper around `head()` that first converts the results to a data frame using `as.data.frame.verify_stockflow()`.

Usage

```
## S3 method for class 'verify_stockflow'
head(x, n = 6L, ...)
```

Arguments

`x`	A `verify_stockflow` object.
`n`	Number of rows. Defaults to 6.
`...`	Other arguments passed to `as.data.frame.verify_stockflow()`.

Value

A `data.frame`.

Examples

```
sfm <- stockflow("sir") |>
  unit_test(expr = all(susceptible >= 0))
res <- verify(sfm)
head(res)
```

hill *Hill function*

Description

Computes the Hill function with configurable slope, midpoint, and upper asymptote.

Usage

```
hill(x, slope = 1, midpoint = 0.5, upper = 1)
```

Arguments

`x`	Value at which to evaluate the function
`slope`	Slope of Hill function at the midpoint. Defaults to 1.
`midpoint`	Midpoint of Hill function where the output is `upper/2`. Defaults to 0.5.
`upper`	Upper asymptote (maximal value) of the Hill function. Defaults to 1.

Details

The Hill function is a smooth S-shaped curve (when slope > 1) bounded between 0 and `upper`. It transitions from near 0 to near `upper` around the `midpoint`, with the steepness of this transition controlled by `slope`. See https://en.wikipedia.org/wiki/Hill_equation_%28biochemistry%29 for more details.

Value

Numeric value given by

$$f(x) = \frac{upper \cdot x^{slope}}{midpoint^{slope} + x^{slope}}$$

Examples

```
hill(0)

# Adjust parameters
hill(0, slope = 5, midpoint = 0.5, upper = 10)
```

`import_desolve` *Import a deSolve model*

Description

Convert a model written for deSolve into a stock-and-flow model of class `stockflow`.

Usage

```
import_desolve(model, params, init, times, method = "lsoda", name = NULL)
```

Arguments

`model`	A deSolve-style ODE function with arguments `(t, state, parameters)`.
`params`	Named numeric vector of model parameters (constants).
`init`	Named numeric vector of initial state values (stocks).
`times`	Numeric vector of time points. Must be evenly spaced (e.g., from `seq(start, stop, by = dt)`).
`method`	Integration method. Defaults to `"lsoda"`. See `sim_methods()`.
`name`	Optional model name. Character scalar.

Details

The model function must follow the canonical deSolve convention:

```
model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dX <- <rate expression>   # d<VarName> for each state in init
    list(c(dX))
  })
}
```

State variable names are taken from `names(init)`, parameter names from `names(params)`. Each `d<VarName>` assignment inside the `with()` block is parsed as the net rate of change for stock `VarName` and becomes a flow in the sfm. Any other assignments in the `with()` block (intermediate calculations) are imported as auxiliary variables in the order they appear.

Value

A stock-and-flow model of class `stockflow`.

See Also

`import_insightmaker()`, `export_model()`, `update()`

Examples

```
logistic_model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dN <- r * N * (1 - N / K)
    list(c(dN))
  })
}
sfm <- import_desolve(
  model  = logistic_model,
  params = c(r = 0.3, K = 100),
  init   = c(N = 10),
  times  = seq(0, 50, by = 0.1),
  method = "lsoda",
  name   = "Logistic growth"
)

sim <- simulate(sfm)
plot(sim)
```

`import_insightmaker` *Import Insight Maker model*

Description

Import a stock-and-flow model from Insight Maker. Models may be your own or another user's. Importing causal loop diagrams or agent-based models is not supported.

Usage

```
import_insightmaker(
  url,
  file,
  keep_nonnegative_flow = TRUE,
  keep_nonnegative_stock = FALSE
)
```

Arguments

`url`	URL to Insight Maker model. Character.
`file`	File path to Insight Maker model. Only used if url is not specified. Needs to be a character with suffix .InsightMaker or .json.
`keep_nonnegative_flow`	If TRUE, keeps original non-negativity setting of flows. Defaults to TRUE.
`keep_nonnegative_stock`	If TRUE, keeps original non-negativity setting of stocks. Defaults to FALSE.

Details

Insight Maker models can be imported using a URL, Insight Maker file, or ModelJSON file. Ensure the URL refers to a public (not private) model. To download a model file from Insight Maker, first clone the model if it is not your own. Then, go to "Share" (top right), "Export", and "Download Insight Maker file" or "ModelJSON File".

Value

A stock-and-flow model object of class `stockflow`.

See Also

`update()`, `stockflow()`

Examples

```
# Load a model from Insight Maker
sfm <- import_insightmaker(
  url =
    "https://insightmaker.com/insight/43tz1nvUgbIiIOGSGtzIzj/Romeo-Juliet"
)
plot(sfm)



# Simulate the model
sim <- simulate(sfm)
plot(sim)
```

---

indexof *Find index of value in vector or string*

Description

Equivalent of .IndexOf() in Insight Maker.

Usage

```
indexof(haystack, needle)
```

Arguments

haystack	Vector or string to search through
needle	Value to search for

Value

Index, integer

Examples

```
indexof(c("a", "b", "c"), "b") # 2
indexof("haystack", "hay") # 1
indexof("haystack", "m") # 0
```

`insightmaker_to_json` *Convert .InsightMaker file to .json file*

Description

Import and convert a stock-and-flow model from Insight Maker to a .json file. Models may be your own or another user's. Importing causal loop diagrams or agent-based models is not supported.

Usage

```
insightmaker_to_json(url, file, destfile = NULL)
```

Arguments

`url`	URL to Insight Maker model. Character.
`file`	File path to Insight Maker model. Only used if url is not specified. Needs to be a character with suffix .InsightMaker.
`destfile`	Output file path. Must have extension .json or no extension. Overwrites file if it already exists. If not provided, return model in json format.

Details

Insight Maker models can be imported using a URL or Insight Maker file. Ensure the URL refers to a public (not private) model. To download a model file from Insight Maker, first clone the model if it is not your own. Then, go to "Share" (top right), "Export", and "Download Insight Maker file".

Value

If destfile is not provided; object of class "json". If destfile provided, invisibly returns destfile (character string).

Examples

```
# Convert a model from Insight Maker to json
destfile <- tempfile(fileext = ".json")
json <- insightmaker_to_json(
  url =
    "https://insightmaker.com/insight/43tz1nvUgbIiIOGSGtzIzj/Romeo-Juliet",
  destfile = destfile
)
file.remove(destfile)
```

`install_julia_env` *Install, update, or remove Julia environment*

Description

Instantiate the Julia environment for sdbuildR to run stock-and-flow models using Julia. For more guidance, see this vignette.

Usage

```
install_julia_env(remove = FALSE, force = FALSE, quiet = FALSE)
```

Arguments

`remove`	If `TRUE`, remove the Julia environment for sdbuildR by deleting its environment directory (containing Project.toml and Manifest.toml). All other Julia packages and environments remain untouched. The packages themselves stay in your Julia depot, so reinstalling is quick; run `Pkg.gc()` in Julia if you want to reclaim that disk space.
`force`	If `TRUE`, rebuild the environment from scratch even when it is already up to date. By default an environment that is already installed, built from the same dependencies, and resolved by a compatible version of Julia is left alone, since rebuilding re-resolves every package and forces Julia to precompile them all again. Defaults to `FALSE`.
`quiet`	If `TRUE`, suppress informational messages such as progress and status updates. Warnings and errors are always shown. Defaults to `FALSE`.

Details

`install_julia_env()` will:

- Start a Julia session
- Activate a Julia environment using sdbuildR's Project.toml
- Install SystemDynamicsBuildR.jl from GitHub (https://github.com/kcevers/SystemDynamicsBuildR.jl)
- Install all other required Julia packages
- Create Manifest.toml
- Precompile packages for faster subsequent loading
- Stop the Julia session

Note that this may take 10-25 minutes the first time as Julia downloads and compiles packages.

Value

Invisibly returns `NULL` after instantiating the Julia environment.

See Also

use_julia()

Examples

```
## Not run:
install_julia_env()

# Remove Julia environment
install_julia_env(remove = TRUE)

## End(Not run)
```

`length_IM` *Length of vector or string*

Description

Equivalent of .Length() in Insight Maker, which returns the number of elements when performed on a vector, but returns the number of characters when performed on a string

Usage

```
length_IM(x)
```

Arguments

x	A vector or a string

Value

The number of elements in x if x is a vector; the number of characters in x if x is a string

Examples

```
length_IM(c("a", "b", "c")) # 3
length_IM("abcdef") # 6
```

`logistic` *Logistic function*

Description

Computes the logistic (i.e., sigmoid) function with configurable slope, midpoint, and upper asymptote.

Usage

```
logistic(x, slope = 1, midpoint = 0, upper = 1)

sigmoid(x, slope = 1, midpoint = 0, upper = 1)
```

Arguments

`x`	Value at which to evaluate the function
`slope`	Slope of logistic function at the midpoint. Defaults to 1.
`midpoint`	Midpoint of logistic function where the output is `upper/2`. Defaults to 0.
`upper`	Upper asymptote (maximal value) of the logistic function. Defaults to 1.

Details

The logistic function is a smooth S-shaped curve bounded between 0 and `upper`. It transitions from near 0 to near `upper` around the `midpoint`, with the steepness of this transition controlled by `slope`.

Value

Numeric value given by

$$f(x) = \frac{upper}{1 + e^{-slope \cdot (x - midpoint)}}$$

Examples

```
logistic(0)
# equivalent:
sigmoid(0)

# Adjust parameters
logistic(0, slope = 5, midpoint = 0.5, upper = 10)

# Visualize different slopes
curve(logistic(x, slope = 1), from = -5, to = 5, ylab = "f(x)", ylim = c(0, 1))
curve(logistic(x, slope = 5), add = TRUE, col = "blue")
curve(logistic(x, slope = 50), add = TRUE, col = "red")
legend("topleft",
  legend = c("slope = 1", "slope = 5", "slope = 50"),
  col = c("black", "blue", "red"), lty = 1
)
```

logit *Logit function*

Description

Logit function

Usage

```
logit(p)
```

Arguments

p	Probability, numerical value between 0 and 1

Value

Numerical value

Examples

```
logit(.1)
```

lookup *Add or modify lookup variables (graphical functions)*

Description

Lookup variables define piecewise relationships using specified (x, y) points. `lookup()` adds or changes a lookup variable. This is a convenience wrapper around `update()` with `type = "lookup"`. See the **Lookup Variables** section of `update()` for more details.

Usage

```
lookup(
  object,
  name,
  xpts,
  ypts,
  source = NULL,
  interpolation = "linear",
  extrapolation = "nearest",
  label = name,
  doc = "",
  non_negative = FALSE
)
```

Arguments

object	Stock-and-flow model, object of class stockflow.
name	Variable name. Accepts a bare symbol (e.g., population), a string ("population"), or a vector via c() (e.g., c(a, b) or c("a", "b")). Use !! to inject from a variable.
xpts	Only for graphical functions: vector of x-domain points. Must be of the same length as ypts.
ypts	Only for graphical functions: vector of y-domain points. Must be of the same length as xpts.
source	Only for graphical functions: name of the variable which will serve as the input to the graphical function. Accepts a bare symbol or string. Defaults to NULL.
interpolation	Only for graphical functions: interpolation method. Must be either "constant" or "linear". Defaults to "linear".
extrapolation	Only for graphical functions: extrapolation method. Must be either "nearest" or "NA". Defaults to "nearest".
label	Name of variable used for plotting. Defaults to the same as name.
doc	Description of variable. Defaults to "" (no description).
non_negative	If TRUE, variable is enforced to be non-negative (i.e., strictly 0 or positive). Defaults to FALSE.

Value

A stock-and-flow model object of class stockflow

See Also

update(), discard(), change_name()

Examples

```
# Create a lookup variable for a non-linear relationship
sfm <- stockflow() |>
  lookup(output,
    source = t,
    xpts = c(0, 5, 10),
    ypts = c(0, 10, 15),
    interpolation = "linear"
  ) |>
  stock(x) |>
  flow(x_in, eqn = output(t), to = x)

sim <- simulate(sfm)
plot(sim)
```

`meta` *Modify meta of stock-and-flow model*

Description

The meta of a stock-and-flow model contains metadata about the model, such as the name, author, and version. Modify the meta of an existing model with standard or custom properties.

Usage

```
meta(
  object,
  name = "My Model",
  caption = "My Model Description",
  created = Sys.time(),
  author = "Me",
  version = "0.0.0.9000",
  URL = "",
  doi = "",
  ...
)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`name`	Model name. Defaults to "My Model".
`caption`	Model description. Defaults to "My Model Description".
`created`	Date the model was created. Defaults to Sys.time().
`author`	Creator of the model. Defaults to "Me".
`version`	Model version. Defaults to "0.0.0.9000", a development version.
`URL`	URL associated with model. Defaults to "".
`doi`	DOI associated with the model. Defaults to "".
`...`	Optional other entries to add to the meta.

Value

A stock-and-flow model object of class `stockflow`

Examples

```
sfm <- stockflow() |>
  meta(
    name = "My first model",
    caption = "This is my first model",
    author = "Kyra Evers",
    version = "1.1"
  )
```

`plot.ensemble_stockflow` *Plot timeseries of ensemble simulation*

Description

Visualize ensemble simulation results of a stock-and-flow model. Either summary statistics or individual trajectories can be plotted. When multiple conditions j are specified, a grid of subplots is plotted. See `ensemble()` for examples.

Usage

```
## S3 method for class 'ensemble_stockflow'
plot(
  x,
  which = c("summary", "sims")[1],
  sim = seq(1, min(c(x[["n"]], 100))),
  condition = seq(1, min(c(x[["n_conditions"]], 9))),
  vars = NULL,
  order = vars,
  show_constants = FALSE,
  nrows = ceiling(sqrt(max(condition))),
  margin = 0.05,
  shareX = TRUE,
  shareY = TRUE,
  palette = "Dark 2",
  alpha = list(central = 1, spread = 0.3, sims = 0.3),
  colors = NULL,
  line_width = list(central = 3, spread = 0, sims = 1),
 line_type = list(stock = "solid", flow = "solid", aux = "solid", constant = "dash",
    lookup = "dash"),
  font_family = getOption("sdbuildR.font_family", default = "stix-two-text"),
  font_size = 16,
  wrap_width = 25,
  show_legend = TRUE,
  label_subplots = TRUE,
  central = c("mean", "median", "none"),
  spread = c("quantile", "sd", "range"),
  format_label = TRUE,
  condition_display = c("subplots", "slider", "dropdown"),
  control_options = list(),
  animation = c("none", "time"),
  webgl = getOption("sdbuildR.webgl", default = TRUE),
  ...
)
```

Arguments

x	Output of ensemble().
which	Type of plot. Either "summary" for a summary plot with mean or median lines and confidence intervals, or "sims" for individual simulation trajectories with mean or median lines. Defaults to "summary".
sim	Indices of the individual trajectories to plot if which = "sims". Defaults to the first min(n, 100) trajectories. Including a high number of trajectories will slow down plotting considerably.
condition	Indices of the condition(s) to plot. Defaults to 1:9.
vars	Variables to plot. Defaults to NULL to plot all variables.
order	Character vector of model variable names giving the trace and legend order. Listed variables appear first; any plotted variables not listed keep their default relative order. Defaults to vars, so the order in which variables are listed in vars is followed unless order is set explicitly.
show_constants	If TRUE, include constants in plot. Defaults to FALSE.
nrows	Number of rows in the plot grid. Defaults to ceiling(sqrt(max(condition))).
margin	Margin between subplots. Either a single numeric or a vector of length four(left, right, top, bottom). See ?plotly::subplot() for more details. Defaults to 0.05.
shareX	If TRUE, share the x-axis across subplots. Defaults to TRUE.
shareY	If TRUE, share the y-axis across subplots. Defaults to TRUE.
palette	Colour palette. Must be one of hcl.pals().
alpha	Opacity, with the same grammar as line_width: a single value, a named per-variable vector, or a list keyed by layer (central/spread/ sims). Defaults to list(central = 1, spread = 0.3, sims = 0.3).
colors	Colours for the plotted variables. A named vector (names are variable names) sets the colours of those variables and the palette fills the rest, so you can re-colour only a few variables. An unnamed vector assigns colours in plot order. NULL uses palette. Defaults to NULL.
line_width	Line width(s). The plot draws three layers: the central tendency line (central), the uncertainty band's border (spread), and the individual trajectories (sims). Supply a single value (used for every layer and variable), a named per-variable vector (names are variable names; unnamed values fill in plot order), or a list keyed by layer (list(central = , spread = , sims =)) whose elements are themselves a single value or per-variable vector. Unspecified layers/variables fall back to the defaults. Defaults to list(central = 3, spread = 0, sims = 1) (a spread width of 0 draws no band border).
line_type	Dash style of the plotted lines. Each value is one of "solid", "dot", "dash", "longdash", "dashdot", "longdashdot", or a pixel pattern such as "5px,10px". Supply a single style for every variable (line_type = "dash"), a named vector to style variables individually (line_type = c(S = "solid", I = "dot")), an unnamed vector in plot order, or a list keyed by variable type to style each type at once (line_type = list(stock = "solid", flow = "dash"); the types are stock, flow, aux, constant, and lookup). By default stocks, flows, and auxiliaries are solid and constants and lookups are dashed.

font_family
: Font family. Kebab-case names (e.g. "eb-garamond") are Fontsource ids (browse them at https://fontsource.org/), loaded as webfonts: no installation is needed, but internet access is required to display them. Other fonts must be installed on the system viewing the plot. Defaults to the sdbuildR.font_family option, or the "stix-two-text" webfont when the option is unset; use options(sdbuildR.font_family =) to change the default for all plots, e.g. in your .Rprofile.

font_size
: Font size. Defaults to 16.

wrap_width
: Width of text wrapping for labels. Must be an integer. Defaults to 25.

show_legend
: Whether to show legend. Must be TRUE or FALSE. Defaults to TRUE.

label_subplots
: Whether to plot labels indicating the condition of the subplot.

central
: Which central-tendency line to draw, given as preferences in order: the first one that ensemble() computed is used. For example, c("mean", "median") draws the mean if it is available, otherwise the median, and "none" draws no line. Defaults to c("mean", "median", "none").

spread
: Which uncertainty band to draw, again as ordered preferences: "quantile" (between the lowest and highest quantile), "sd" (the central line plus/minus one standard deviation), "range" (between min and max), or "none". The first band the computed statistics can support is used; "sd" also needs a central line. Defaults to c("quantile", "sd", "range").

format_label
: If TRUE, apply default formatting (replacing periods and underscores with spaces) to variable labels that are the same as the variable name. Applies to the legend and any condition controls. Defaults to TRUE.

condition_display
: How to display multiple conditions. Use "subplots" to show conditions as panels, "slider" to select one condition with a slider, or "dropdown" to select one condition with a dropdown. Defaults to "subplots". If only a single condition is available (e.g. no conditions were varied), "slider" and "dropdown" fall back to "subplots" with a message. To guarantee the control geometry, plots with a slider/dropdown have a fixed height (sized to the number of controls) instead of a responsive one.

control_options
: Named list fine-tuning the "slider"/"dropdown" condition control and the animation = "time" animation. For the condition control it supports max_labels: the maximum number of slider tick labels to keep visible when many conditions are varied (the slider always keeps one step per condition; intermediate labels are thinned above this count); and spacing: the vertical gap (in pixels) between the tops of stacked controls when several condition parameters are varied. By default the spacing and the reserved bottom margin are sized automatically so the controls never overlap each other or the x-axis title; pass a number to widen or tighten the gap. For the animation it supports frame_ms: the duration of each frame in milliseconds (default 100); duration: the total animation length in seconds, as an alternative to frame_ms (supplying both is an error); transition_ms: the transition time between frames in milliseconds (default 0); and max_frames: the maximum number of animation frames (default 50). Defaults to list(), i.e. all defaults.

animation	Animation mode. Use "none" for a static plot or "time" to cumulatively reveal trajectories over time. Defaults to "none". Time animation requires which = "sims" (confidence ribbons cannot be animated) and a single condition (one panel); combining it with condition_display controls or multiple conditions is not supported.
webgl	If TRUE, render trajectories with WebGL (plotly scattergl) for performance with many lines; if FALSE, use SVG (scatter). Defaults to getOption("sdbuildR.webgl", default = TRUE). Set options(sdbuildR.webgl = FALSE) (e.g. in vignettes or dashboards, or when a plot renders blank) to disable WebGL globally. WebGL is not supported for filled flow traces (fill_flows = TRUE) or time animations; these always render with SVG scatter regardless of webgl.
...	Optional parameters

Value

Plotly object

Styling variables and layers

Names in colors, alpha, and line_width refer to the model variable names, not the labels shown in the legend. For example, use infected, not Infected, even when format_label = TRUE changes the legend text.

A single value applies everywhere: plot(sims, line_width = 3, alpha = 0.7).

A named vector styles selected variables and leaves the rest at their defaults: plot(sims, colors = c(infected = "firebrick"), line_width = c(infected = 4)).

Ensemble plots also have layers. Use a list to style the central tendency line, the uncertainty band border, and individual trajectories separately: plot(sims, which = "sims", line_width = list(central = 3, sims = 1, spread = 0)).

List elements can be named vectors too, which is useful when only one layer needs variable-specific styling: plot(sims, alpha = list(central = 1, sims = c(infected = 0.15), spread = 0.25)).

The spread line width controls the border of the uncertainty band. The default is 0, so the band is filled but not outlined.

See Also

ensemble()

---

plot.simulate_stockflow

Plot timeseries of simulation

Description

Visualize simulation results of a stock-and-flow model. Plot the evolution of stocks over time, with the option of also showing other model variables.

Usage

```
## S3 method for class 'simulate_stockflow'
plot(
  x,
  show_constants = FALSE,
  vars = NULL,
  order = vars,
  palette = "Dark 2",
  colors = NULL,
  line_width = 2,
 line_type = list(stock = "solid", flow = "solid", aux = "solid", constant = "dash",
    lookup = "dash"),
  vars_display = c("vstack", "hstack", "joint"),
  fill_flows = TRUE,
  font_family = getOption("sdbuildR.font_family", default = "stix-two-text"),
  font_size = 16,
  wrap_width = 25,
  show_legend = TRUE,
  format_label = TRUE,
  animation = c("none", "time"),
  control_options = list(),
  webgl = getOption("sdbuildR.webgl", default = TRUE),
  ...
)
```

Arguments

x	Output of simulate().
show_constants	If TRUE, include constants in plot. Defaults to FALSE.
vars	Variables to plot. Defaults to NULL to plot all variables.
order	Character vector of model variable names giving the trace and legend order. Listed variables appear first; any plotted variables not listed keep their default relative order. Defaults to vars, so the order in which variables are listed in vars is followed unless order is set explicitly.
palette	Colour palette. Must be one of hcl.pals().
colors	Colours for the plotted variables. A named vector (names are variable names) sets the colours of those variables and the palette fills the rest, so you can re-colour only a few variables. An unnamed vector assigns colours in plot order. NULL uses palette. Defaults to NULL.
line_width	Line width of the plotted trajectories. Either a single value applied to all variables, a named per-variable vector (names are variable names), or an unnamed vector with one value per variable in plot order. Defaults to 2.
line_type	Dash style of the plotted lines. Each value is one of "solid", "dot", "dash", "longdash", "dashdot", "longdashdot", or a pixel pattern such as "5px,10px". Supply a single style for every variable (line_type = "dash"), a named vector to style variables individually (line_type = c(S = "solid", I = "dot")),

an unnamed vector in plot order, or a list keyed by variable type to style each type at once (`line_type = list(stock = "solid", flow = "dash")`; the types are `stock`, `flow`, `aux`, `constant`, and `lookup`). By default stocks, flows, and auxiliaries are solid and constants and lookups are dashed.

`vars_display` How to arrange saved variables. Use `"vstack"` to stack stocks above flows, auxiliaries, constants, and lookups; `"hstack"` to put stocks beside non-stock variables; or `"joint"` to draw all variables in one panel. If only one variable group is available, a single panel is drawn. Defaults to `"vstack"`.

`fill_flows` If `TRUE`, flow traces are filled down to zero. Auxiliaries, constants, and lookups are never filled. Defaults to `TRUE`. Filled flows always render with SVG `scatter`, as WebGL does not support fills (see `webgl`).

`font_family` Font family. Kebab-case names (e.g. `"eb-garamond"`) are Fontsource ids (browse them at https://fontsource.org/), loaded as webfonts: no installation is needed, but internet access is required to display them. Other fonts must be installed on the system viewing the plot. Defaults to the `sdbuildR.font_family` option, or the `"stix-two-text"` webfont when the option is unset; use `options(sdbuildR.font_family = )` to change the default for all plots, e.g. in your .Rprofile.

`font_size` Font size. Defaults to 16.

`wrap_width` Width of text wrapping for labels. Must be an integer. Defaults to 25.

`show_legend` Whether to show legend. Must be `TRUE` or `FALSE`. Defaults to `TRUE`.

`format_label` If `TRUE`, apply default formatting (replacing periods and underscores with spaces) to variable labels that are the same as the variable name. Applies to the legend and any condition controls. Defaults to `TRUE`.

`animation` Animation mode. Use `"none"` for a static plot or `"time"` to cumulatively reveal trajectories over time. Defaults to `"none"`.

`control_options`

Named list fine-tuning the speed and smoothness of the `animation = "time"` animation. Supports `frame_ms`: the duration of each frame in milliseconds (default `100`); `duration`: the total animation length in seconds, as an alternative to `frame_ms` (supplying both is an error); `transition_ms`: the transition time between frames in milliseconds (default `0`); and `max_frames`: the maximum number of animation frames (default `50`; lower it for a chunkier reveal that also shortens the animation at a fixed `frame_ms`). Defaults to `list()`, i.e. all defaults.

`webgl` If `TRUE`, render trajectories with WebGL (plotly `scattergl`) for performance with many lines; if `FALSE`, use SVG (`scatter`). Defaults to `getOption("sdbuildR.webgl", default = TRUE)`. Set `options(sdbuildR.webgl = FALSE)` (e.g. in vignettes or dashboards, or when a plot renders blank) to disable WebGL globally. WebGL is not supported for filled flow traces (`fill_flows = TRUE`) or time animations; these always render with SVG `scatter` regardless of `webgl`.

`...` Optional parameters

Value

Plotly object

Styling variables

Names in `colors`, `line_width`, and `line_type` refer to the model variable names, not the labels shown in the legend.

Use one value to style every trajectory: `plot(sim, line_width = 3)`.

Use a named vector to style selected variables and leave the rest at their defaults or palette colours: `plot(sim, colors = c(susceptible = "#377EB8"), line_width = c(infected = 4))`.

See Also

`simulate()`, `as.data.frame.simulate_stockflow()`, `plot.simulate_stockflow()`

Examples

```
sfm <- stockflow("sir")
sim <- simulate(sfm)
plot(sim)

# When all variables are saved in the output, the stocks are plotted
# separately from the other variables by default:
sim_all <- simulate(sfm, only_stocks = FALSE)
plot(sim_all)


# Plot all variables in one panel:
plot(sim_all, vars_display = "joint")

# Animate the simulation over time:
plot(sim_all, animation = "time")

# Slow the animation down to ~10 seconds in total, or speed up the
# individual frames
plot(sim, animation = "time", control_options = list(duration = 10))
plot(sim, animation = "time", control_options = list(frame_ms = 40))

## Styling options
# The default plot title and axis labels can be changed like so:
plot(sim, main = "Simulated trajectory", xlab = "Time", ylab = "Value")

# Add constants to the plot
plot(sim, show_constants = TRUE)

# Plot selected variables:
plot(sim, vars = c("susceptible", "infected"))
```

---

plot.stockflow *Plot stock-and-flow diagram*

Description

Visualize a stock-and-flow diagram using the R package DiagrammeR. Stocks are represented as boxes. Flows are represented as arrows between stocks and/or double circles, where the latter represent what it outside of the model boundary. Thin grey edges indicate dependencies between variables. By default, constants (indicated by italic labels) are not shown. Hover over the variables to see their equations.

Usage

```
## S3 method for class 'stockflow'
plot(
  x,
  vars = NULL,
  format_label = TRUE,
  wrap_width = 20,
  font_size = 18,
  font_family = getOption("sdbuildR.font_family", default = "stix-two-text"),
  colors = NULL,
  color_dependency = "#999999",
  font_color = "black",
  show_eqn = TRUE,
  show_tooltip = TRUE,
  show_dependencies = TRUE,
  show_constants = FALSE,
  show_aux = TRUE,
  flow_length = 1,
  margin = 0.1,
  spacing = 0.5,
  direction = "LR",
  align = NULL,
  order = NULL,
  arrowhead_dependency = "open",
  ...
)
```

Arguments

x	A stock-and-flow model object of class stockflow.
vars	Variables to plot. Defaults to NULL to plot all variables.
format_label	If TRUE, apply default formatting (removing periods and underscores) to labels if labels are the same as variable names.
wrap_width	Width of text wrapping for labels. Must be an integer. Defaults to 20.
font_size	Font size. Defaults to 18.
font_family	Font name. Kebab-case names (e.g. "eb-garamond") are Fontsource ids (browse them at https://fontsource.org/), loaded as webfonts: no installation is needed, but internet access is required to display them. Other fonts must be installed on the system viewing the plot. Defaults to the sdbuildR.font_family

option, or the "stix-two-text" webfont when the option is unset; use options(sdbuildR.font_family =) to change the default for all plots, e.g. in your .Rprofile.

colors — Colours of the diagram variables, by variable type or variable name. A single colour applies to every variable. A named list keyed by type (list(stock = , flow = , constant = , aux =)), each entry a single colour, overrides the colour of those types and keeps the defaults for the rest (as with utils::modifyList()); the defaults are list(stock = "#83d3d4", flow = "#f48153", constant = "grey90", aux = "grey90"). A named vector (names are variable names, as in the other plot methods) recolours only those variables and leaves the rest at their type default. Stocks, constants, and auxiliaries are coloured by their node fill; flows by the coloured band of their flow arrows. Defaults to NULL (the default type colours).

color_dependency — Colour of dependency arrows. Defaults to "#999999".

font_color — Colour of variable labels (and of the equation text when show_eqn = TRUE). Defaults to "black".

show_eqn — If TRUE, show each variable's equation on a new line beneath its label, in a smaller font and the same colour as the label (font_color). The equation is prefixed by what it defines for that variable type: Initial value = for stocks, Rate = for flows, Value = for constants, and Equation = for auxiliaries. Defaults to TRUE.

show_tooltip — If TRUE, show each variable's equation as a tooltip when hovering over it. Defaults to TRUE.

show_dependencies — If TRUE, show dependencies between variables. Defaults to TRUE.

show_constants — If TRUE, show constants. Defaults to FALSE.

show_aux — If TRUE, show auxiliary variables. Defaults to TRUE.

flow_length — Minimum length of flow arrows; controls how far apart connected variables sit along the flow direction. Must be an integer. Defaults to 1.

margin — Padding around the diagram. Defaults to 0.1.

spacing — Minimum space between adjacent variables placed side by side (across the flow direction). Defaults to 0.5.

direction — Overall flow direction of the layout, passed to Graphviz's rankdir. One of "LR" (left-to-right, the default), "TB" (top-to-bottom), "RL" (right-to-left), or "BT" (bottom-to-top).

align — Optional alignment of variables *across* the flow direction. A character vector of variable names, or a list of such vectors. Each group is placed on the same Graphviz rank ({rank=same; ...}), so its members line up (vertically when direction = "LR", horizontally when direction = "TB"). Works for any variable (stocks, flows, auxiliaries, constants), not only stocks. Names that are not currently drawn (hidden by vars, show_constants, or show_aux) are ignored with a warning; unknown names raise an error. Defaults to NULL.

order — Optional ordering of variables *along* the flow direction. A character vector of variable names, or a list of such vectors, giving the desired sequence. Implemented as invisible edges between consecutive names, so it acts as a soft hint

that Graphviz balances against the real flows rather than a hard constraint. On its own it sequences variables into successive ranks (e.g. separate columns when `direction = "LR"`); to instead line variables up in a single rank and control their order *within* it, combine `order` with `align` (the `align` group sets the rank, `order` sets the position within it). Same validation as `align`. Defaults to `NULL`.

`arrowhead_dependency`
: Shape of the arrowhead on dependency arrows, passed to Graphviz's `arrowhead`. One of `"open"` (the default, an open V-shape), `"normal"`, `"vee"`, `"diamond"`, `"dot"`, `"box"`, `"crow"`, `"curve"`, `"inv"`, `"tee"`, or `"none"`. See https://graphviz.org/docs/attr-types/arrowType/ for illustrations.

`...`
: Optional arguments

Value

Stock-and-flow diagram

See Also

`import_insightmaker()`, `stockflow()`, `plot.simulate_stockflow()`

Examples

```
sfm <- stockflow("sir")
plot(sfm)

# Don't show constants or auxiliaries
plot(sfm, show_constants = FALSE, show_aux = FALSE)

# Only show specific variables
plot(sfm, vars = "susceptible")

# Hide the equations shown beneath each label
plot(sfm, show_eqn = FALSE)

# Hide the equation tooltips shown on hover
plot(sfm, show_tooltip = FALSE)

# Custom label colour
plot(sfm, font_color = "#333333")

# Colour every variable the same
plot(sfm, colors = "lightblue")

# Change only the stock colour; other types keep their defaults
plot(sfm, colors = list(stock = "gold"))

# Recolour a single variable
plot(sfm, colors = c(susceptible = "gold"))

# Lay the model out top-to-bottom instead of left-to-right
plot(sfm, direction = "TB")
```

```
# Align variables across the flow direction (same Graphviz rank)
plot(sfm, align = c("susceptible", "recovered"))

# Order variables along the flow direction (soft hint via invisible edges)
plot(sfm, order = c("susceptible", "infected", "recovered"))
```

`plot.verify_stockflow` *Plot verify results*

Description

Visualize the simulation(s) used during `verify()`. Each *condition* is one simulation run – the model with one set of parameter overrides (from `unit_test(conditions = )`, possibly none) – shared by every unit test declaring those overrides. Each condition is displayed as a subplot (or a slider/dropdown step, see `condition_display`), labelled by its overrides and the test(s) evaluated on it: e.g. `"rate = 0 (test 2)"`, with the unmodified run labelled `"Baseline (tests 1, 3)"` (or just `"Tests 1, 3"` when no test varies anything). Simulations are always available since `verify()` unconditionally retains them.

Usage

```
## S3 method for class 'verify_stockflow'
plot(
  x,
  test = NULL,
  vars = NULL,
  order = vars,
  show_constants = FALSE,
  label = NULL,
  ignore_case = TRUE,
  status = c("pass", "fail", "error", "skip"),
  condition = seq(1, min(c(x[["n_conditions"]], 9))),
  nrows = ceiling(sqrt(max(condition))),
  shareX = TRUE,
  shareY = TRUE,
  palette = "Dark 2",
  colors = NULL,
  line_width = 2,
 line_type = list(stock = "solid", flow = "solid", aux = "solid", constant = "dash",
    lookup = "dash"),
  font_family = getOption("sdbuildR.font_family", default = "stix-two-text"),
  font_size = 16,
  wrap_width = 25,
  show_legend = TRUE,
  label_subplots = TRUE,
```

```
  alpha = 1,
  margin = 0.05,
  format_label = TRUE,
  condition_display = c("subplots", "slider", "dropdown"),
  control_options = list(),
  animation = c("none", "time"),
  webgl = getOption("sdbuildR.webgl", default = TRUE),
  ...
)
```

Arguments

x	Output of verify().
test	Integer vector of test numbers to plot. Combines with label and status as AND intersection.
vars	Variable names to retain in the data frame. Only applies when which = "sims". Defaults to NULL to include all variables.
order	Character vector of model variable names giving the trace and legend order. Listed variables appear first; any plotted variables not listed keep their default relative order. Defaults to vars, so the order in which variables are listed in vars is followed unless order is set explicitly.
show_constants	If TRUE, include constants in plot. Defaults to FALSE.
label	Character vector of regex patterns for partial, case-insensitive label matching. A test is included if its label matches any pattern.
ignore_case	Logical; whether label matching is case-insensitive. Default TRUE.
status	Optional character vector of statuses to include (e.g., c("fail", "error")). Defaults to all statuses. • which = "tests": filters rows by test status. • which = "sims": filters to conditions that have at least one test with a matching status.
condition	Integer vector of condition numbers to plot. Defaults to 1:9. If only one condition is specified, the plot will not be a grid of subplots.
nrows	Number of subplot rows. Defaults to ceiling(sqrt(max(condition))).
shareX	Share the x-axis across subplots. Defaults to TRUE.
shareY	Share the y-axis across subplots. Defaults to TRUE.
palette	Colour palette (see hcl.pals()). Defaults to "Dark 2".
colors	Colours for the plotted variables. A named vector (names are variable names) recolours only those variables, the palette fills the rest; an unnamed vector assigns colours in plot order. Defaults to NULL.
line_width	Line width of the trajectories. Either a single value, a named per-variable vector (names are variable names), or an unnamed vector with one value per variable in plot order. Defaults to 2.

`line_type` Dash style of the plotted lines. Each value is one of `"solid"`, `"dot"`, `"dash"`, `"longdash"`, `"dashdot"`, `"longdashdot"`, or a pixel pattern such as `"5px,10px"`. Supply a single style for every variable (`line_type = "dash"`), a named vector to style variables individually (`line_type = c(S = "solid", I = "dot")`), an unnamed vector in plot order, or a list keyed by variable type to style each type at once (`line_type = list(stock = "solid", flow = "dash")`; the types are `stock`, `flow`, `aux`, `constant`, and `lookup`). By default stocks, flows, and auxiliaries are solid and constants and lookups are dashed.

`font_family` Font family. Kebab-case names (e.g. `"eb-garamond"`) are Fontsource ids (browse them at https://fontsource.org/), loaded as webfonts: no installation is needed, but internet access is required to display them. Other fonts must be installed on the system viewing the plot. Defaults to the `sdbuildR.font_family` option, or the `"stix-two-text"` webfont when the option is unset; use `options(sdbuildR.font_family = )` to change the default for all plots, e.g. in your .Rprofile.

`font_size` Font size. Defaults to `16`.

`wrap_width` Label wrap width. Defaults to `25`.

`show_legend` Whether to show the legend. Defaults to `TRUE`.

`label_subplots` Whether to title each subplot with its condition's parameter overrides (or `"Baseline"`) and test number(s), e.g. `"rate = 0 (test 2)"`. Defaults to `TRUE`.

`alpha` Trajectory opacity, between 0 and 1. A single value or a named per-variable vector. Defaults to `1`.

`margin` Margin between subplots. Either a single numeric or a vector of length four(left, right, top, bottom). See `?plotly::subplot()` for more details. Defaults to 0.05.

`format_label` If `TRUE`, apply default formatting (replacing periods and underscores with spaces) to variable labels that are the same as the variable name. Applies to the legend and any condition controls. Defaults to `TRUE`.

`condition_display` How to display multiple conditions. Use `"subplots"` to show conditions as panels, `"slider"` to select one condition with a slider, or `"dropdown"` to select one condition with a dropdown. Defaults to `"subplots"`. If only a single condition is available (e.g. no conditions were varied), `"slider"` and `"dropdown"` fall back to `"subplots"` with a message. To guarantee the control geometry, plots with a slider/dropdown have a fixed height (sized to the number of controls) instead of a responsive one.

`control_options` Named list fine-tuning the `"slider"`/`"dropdown"` condition control and the `animation = "time"` animation; see `plot.ensemble_stockflow()` for the supported options (`max_labels`, `spacing`, `frame_ms`, `duration`, `transition_ms`, `max_frames`). Defaults to `list()`, i.e. all defaults.

`animation` Animation mode. Use `"none"` for a static plot or `"time"` to cumulatively reveal trajectories over time. Defaults to `"none"`.

`webgl` If `TRUE`, render trajectories with WebGL (plotly `scattergl`) for performance with many lines; if `FALSE`, use SVG (`scatter`). Defaults to `getOption("sdbuildR.webgl", default = TRUE)`. Set `options(sdbuildR.webgl = FALSE)` (e.g. in vignettes or

dashboards, or when a plot renders blank) to disable WebGL globally. WebGL is not supported for filled flow traces (`fill_flows = TRUE`) or time animations; these always render with SVG `scatter` regardless of `webgl`.

`...` Additional arguments passed to `plot.simulate_stockflow()`.

Value

A plotly object.

Styling variables

Names in `colors`, `line_width`, and `alpha` refer to the model variable names, not the labels shown in the legend. This keeps the styling stable if labels are formatted or wrapped for display.

Use one value to style every trajectory: `plot(res, line_width = 3, alpha = 0.6)`.

Use named vectors to style selected variables: `plot(res, colors = c(susceptible = "#377EB8"), alpha = c(infected = 0.4))`.

Unnamed vectors are applied in plot order. Named vectors are usually clearer, especially when `vars`, `test`, `label`, or `status` filters change what is drawn.

See Also

`verify()`, `plot.simulate_stockflow()`, `plot.ensemble_stockflow()`

Examples

```
sfm <- stockflow("sir") |>
  unit_test(expr = all(susceptible >= 0)) |>
  unit_test(expr = all(infected >= 0), conditions = list(infected = 100))
result <- verify(sfm)
plot(result)


# Select one condition at a time with a slider or dropdown
plot(result, condition_display = "slider")

# Animate the simulation over time (one condition at a time)
plot(result, animation = "time", condition = 1)
```

---

print.compare_stockflow

Print comparison of two stock-and-flow models

Description

Print comparison of two stock-and-flow models

Usage

```
## S3 method for class 'compare_stockflow'
print(x, ...)
```

Arguments

x	An object of class compare_stockflow
...	Additional arguments (unused)

Value

Invisibly returns x.

---

print.simulate_stockflow

Print simulation of a stock-and-flow model

Description

Prints the first rows of the simulation results in wide format. For a statistical summary per variable use summary().

Usage

```
## S3 method for class 'simulate_stockflow'
print(x, ...)
```

Arguments

x	A simulation result of class simulate_stockflow
...	Additional arguments (unused)

Value

Invisibly returns x

See Also

simulate.stockflow(), summary.simulate_stockflow(), plot.simulate_stockflow(), as.data.frame.simulate_s

Examples

```
sfm <- stockflow("sir")
sim <- simulate(sfm)
print(sim)
```

print.stockflow *Print overview of stock-and-flow model*

Description

Prints a descriptive overview of the model structure, including stock-flow topology, variable names, and simulation settings. For model diagnostics, use `summary()`.

Usage

```
## S3 method for class 'stockflow'
print(x, ...)
```

Arguments

`x`	A stock-and-flow model object of class `stockflow`
`...`	Additional arguments (unused)

Value

Invisibly returns `x`

See Also

`summary.stockflow()`, `dependencies()`

Examples

```
sfm <- stockflow("sir")
print(sfm)
```

print.summary_stockflow *Print method for summary_stockflow*

Description

Print method for summary_stockflow

Usage

```
## S3 method for class 'summary_stockflow'
print(x, ...)
```

Arguments

x	Object of class summary_stockflow
...	Ignored

Value

x invisibly

---

pulse *Create pulse function*

Description

Create a pulse function that jumps from zero to a specified height at a specified time, and returns to zero after a specified width. The pulse can be repeated at regular intervals.

Usage

```
pulse(times, start, height = 1, width = 1, repeat_interval = NULL)
```

Arguments

times	Vector of simulation times
start	Start time of pulse in simulation time units.
height	Height of pulse. Defaults to 1.
width	Width of pulse in simulation time units. This cannot be equal to or less than 0. To indicate an instantaneous pulse, specify the simulation step size.
repeat_interval	Interval at which to repeat pulse. Defaults to NULL to indicate no repetition.

Details

Equivalent of Pulse() in Insight Maker

Value

Pulse interpolation function

See Also

step(), ramp(), seasonal()

Examples

```
# Create a simple model with a pulse function
# that starts at time 5, jumps to a height of 2
# with a width of 1, and does not repeat
sfm <- stockflow() |>
  update("a", "stock") |>
  # Specify the global variable "times" as simulation times
  update("input", "constant", eqn = "pulse(times, 5, 2, 1)") |>
  update("inflow", "flow", eqn = "input(t)", to = "a")



sim <- simulate(sfm, only_stocks = FALSE)
plot(sim)

# Create a pulse that repeats every 5 time units
sfm <- update(sfm, "input", eqn = "pulse(times, 5, 2, 1, 5)")

sim <- simulate(sfm, only_stocks = FALSE)
plot(sim)
```

---

ramp *Create ramp function*

Description

Create a ramp function that increases linearly from 0 to a specified height at a specified start time, and stays at this height after the specified end time.

Usage

```
ramp(times, start, finish, height = 1)
```

Arguments

`times`	Vector of simulation times
`start`	Start time of ramp
`finish`	End time of ramp
`height`	End height of ramp, defaults to 1

Details

Equivalent of Ramp() in Insight Maker

Value

Ramp interpolation function

See Also

step(), pulse(), seasonal()

Examples

```
# Create a simple model with a ramp function
sfm <- stockflow() |>
  update("a", "stock") |>
  # Specify the global variable "times" as simulation times
  update("input", "constant", eqn = "ramp(times, 20, 30, 3)") |>
  update("inflow", "flow", eqn = "input(t)", to = "a")



sim <- simulate(sfm, only_stocks = FALSE)
plot(sim)

# To create a decreasing ramp, set the height to a negative value
sfm <- update(sfm, "input", eqn = "ramp(times, 20, 30, -3)")

sim <- simulate(sfm, only_stocks = FALSE)
plot(sim)
```

---

rbool *Generate random logical value*

Description

Equivalent of RandBoolean() in Insight Maker

Usage

```
rbool(p)
```

Arguments

p	Probability of TRUE, numerical value between 0 and 1

Value

Logical value

Examples

```
rbool(.5)
```

rdist *Generate random number from custom distribution*

Description

Equivalent of RandDist() in Insight Maker

Usage

```
rdist(a, b)
```

Arguments

a	Vector to draw sample from
b	Vector of probabilities

Value

One sample from custom distribution

Examples

```
rdist(c(1, 2, 3), c(.5, .25, .25))
```

rem *Remainder and modulus*

Description

Remainder and modulus operators. The modulus and remainder are not the same in case either a or b is negative. If you work with negative numbers, modulus is always non-negative (it matches the sign of the divisor).

Usage

```
rem(a, b)

mod(a, b)

a %REM% b
```

Arguments

a	Dividend
b	Divisor

Value

Remainder

Examples

```
# Modulus and remainder are the same when a and b are positive
a <- 7
b <- 3
rem(a, b)
mod(a, b)
# Modulus and remainder are NOT when either a or b is negative
a <- -7
b <- 3
rem(a, b)
mod(a, b)
a <- 7
b <- -3
rem(a, b)
mod(a, b)
# Modulus and remainder are the same when both a and b are negative
a <- -7
b <- -3
rem(a, b)
mod(a, b)

# Alternative way of computing the remainder:
a %REM% b
```

---

ricker *Generalized Ricker function*

Description

Computes the generalized Ricker function, a smooth hump-shaped curve that rises from zero, peaks, and then decays back towards zero. It is commonly used to describe humped (non-monotonic) dependencies, such as stock-recruitment relationships in ecology or size-dependent predation.

Usage

```
ricker(x, location = 1, upper = 1, shape = 1, a = NULL, b = NULL)
```

Arguments

x	Value at which to evaluate the function. Because the curve involves a fractional power of x, x is expected to be non-negative.
location	Value of x at which the function reaches its peak. Defaults to 1.
upper	Maximal value (height) of the function, attained at the peak. Defaults to 1.

shape — Exponent controlling the width of the peak: values above 1 narrow the peak, values below 1 broaden it. shape = 1 gives the standard Ricker function. Defaults to 1.

a, b — Coefficients of the equivalent expanded form $f(x) = a \cdot x^{shape} \cdot e^{-b \cdot x}$. Optional alternative to location and upper: when both a and b are supplied, they take precedence and set location = shape / b and upper = a \cdot (location / e)^{shape}. With shape = 1 this is the standard Ricker parameterization $f(x) = a \cdot x \cdot e^{-b \cdot x}$. Supplying only one of them, or combining them with an explicit location or upper, is an error. Default to NULL.

Details

The generalized Ricker function (Persson et al., 1998) is defined as:

$$f(x) = upper \cdot \left(\frac{x}{location} \cdot e^{1 - x/location} \right)^{shape}$$

with a power parameter (α, or shape) that broadens or narrows the peak. The function peaks at x = location, where it attains the value upper, for any shape.

Expanding the expression shows that it is equivalent to:

$$f(x) = a \cdot x^{shape} \cdot e^{-b \cdot x}$$

with coefficients

$$a = upper \cdot (e/location)^{shape}$$

$$b = shape/location$$

or equivalently

$$location = shape/b$$

$$upper = a \cdot (location/e)^{shape}.$$

Note that e is the base of the natural logarithm (i.e., exp(1)). When shape = 1, the power on x is 1 and this reduces to the standard Ricker function $f(x) = a \cdot x \cdot e^{-b \cdot x}$, with $a = upper \cdot e/location$ and $b = 1/location$.

See Bolker, B. M. (2008). *Ecological Models and Data in R*. Princeton University Press, Section 8.1.

Value

Numeric value given by

$$f(x) = upper \cdot \left(\frac{x}{location} \cdot e^{1 - x/location} \right)^{shape}$$

Examples

```
ricker(1)

# Adjust parameters
ricker(2, location = 2, upper = 10, shape = 1)

# Use the expanded form f(x) = a * x^shape * exp(-b * x) instead.
# With shape = 1 this is the standard Ricker f(x) = a * x * exp(-b * x).
ricker(3, a = 2.5, b = 0.4)
# equivalent to:
ricker(3, location = 1 / 0.4, upper = 2.5 * (1 / 0.4 / exp(1)))

# The mapping holds for any shape, e.g. f(x) = a * x^2 * exp(-b * x)
ricker(3, a = 2.5, b = 0.4, shape = 2)

# Visualize different peak widths
curve(ricker(x, location = 2), from = 0, to = 10, ylab = "f(x)", ylim = c(0, 1.5))
curve(ricker(x, location = 2, shape = 0.5), add = TRUE, col = "blue")
curve(ricker(x, location = 2, shape = 3), add = TRUE, col = "red")
legend("topright",
  legend = c("shape = 1", "shape = 0.5", "shape = 3"),
  col = c("black", "blue", "red"), lty = 1
)
```

---

round_IM *Round values half-up (as in Insight Maker)*

Description

R rounds .5 to 0, whereas Insight Maker rounds .5 to 1. This function is the equivalent of Insight Maker's Round() function.

Usage

```
round_IM(x, digits = 0)
```

Arguments

x	Value
digits	Number of digits; optional, defaults to 0

Value

Rounded value

Examples

```
round_IM(.5) # 1
round(.5) # 0
round_IM(-0.5) # 0
round(-0.5) # 0
round_IM(1.5) # 2
round(1.5) # 2
```

---

sdbuildR-as.data.frame

Data frame methods for sdbuildR objects

Description

`as.data.frame()` extracts the contents of an sdbuildR object as a data frame:

Details

`as.data.frame.stockflow()` Structure of a model: its stocks, flows, constants and auxiliaries.

`as.data.frame.simulate_stockflow()` Results of a simulation.

`as.data.frame.ensemble_stockflow()` Results of an ensemble simulation.

`as.data.frame.verify_stockflow()` Results of `verify()`.

See Also

head() and tail() methods

---

sdbuildR-head-tail

First and last rows of sdbuildR results

Description

`head()` and `tail()` return the first or last rows of the data frame produced by `as.data.frame()`:

Details

`head.simulate_stockflow()`, `tail.simulate_stockflow()` Rows of a simulation.

`head.verify_stockflow()`, `tail.verify_stockflow()` Rows of `verify()` results.

`head()` and `tail()` methods are also defined for `ensemble_stockflow` objects.

See Also

as.data.frame() methods

sdbuildR-plot *Plot methods for sdbuildR objects*

Description

sdbuildR provides a `plot()` method for each of its object classes:

Details

`plot.stockflow()` Stock-and-flow diagram of a model.

`plot.simulate_stockflow()` Timeseries of a simulation.

`plot.ensemble_stockflow()` Timeseries of an ensemble simulation.

`plot.verify_stockflow()` Results of `verify()`.

See Also

`stockflow()`, `simulate()`, `ensemble()`, `verify()`

sdbuildR-print *Print methods for sdbuildR objects*

Description

sdbuildR provides a `print()` method for each of its object classes, called automatically when an object is shown at the console:

Details

`print.stockflow()` Overview of a stock-and-flow model.

`print.simulate_stockflow()` Overview of a simulation.

`print.summary_stockflow()` Model diagnostics from `summary()`.

`print.compare_stockflow()` Comparison of two models from `compare_models()`.

`print()` methods are also defined for `ensemble_stockflow`, `verify_stockflow` and `unit_tests_stockflow` objects.

sdbuildR-summary *Summary methods for sdbuildR objects*

Description

sdbuildR provides a `summary()` method for each of its object classes:

Details

`summary.stockflow()` Run diagnostics on a stock-and-flow model.

`summary.simulate_stockflow()` Statistical summary per variable of a simulation.

A `summary()` method is also defined for `ensemble_stockflow` objects, returning the summary statistics computed across runs.

seasonal *Create a seasonal wave function*

Description

Create a seasonal wave function that oscillates between -1 and 1, with a specified period and shift. The wave peaks at the specified shift time.

Usage

```
seasonal(times, period = 1, shift = 0)
```

Arguments

`times`	Vector of simulation times
`period`	Duration of wave in simulation time units. Defaults to 1.
`shift`	Timing of wave peak in simulation time units. Defaults to 0.

Details

Equivalent of Seasonal() in Insight Maker

Value

Seasonal interpolation function

See Also

`step()`, `pulse()`, `ramp()`

Examples

```
# Create a simple model with a seasonal wave
sfm <- stockflow() |>
  update("a", "stock") |>
  # Specify the global variable "times" as simulation times
  update("input", "constant", eqn = "seasonal(times, 10, 0)") |>
  update("inflow", "flow", eqn = "input(t)", to = "a")

sim <- simulate(sfm, only_stocks = FALSE)
plot(sim)
```

`simulate.stockflow` *Simulate stock-and-flow model*

Description

Simulate a stock-and-flow model with simulation specifications defined by `sim_settings()`. If `sim_settings(language = "julia")`, the Julia environment will first be set up with `use_julia()`. If any problems are detected by `summary()`, the model cannot be simulated.

Usage

```
## S3 method for class 'stockflow'
simulate(object, nsim = 1, seed = NULL, quiet = FALSE, ...)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`nsim`	Number of simulations to run (unused; see `ensemble()` for running multiple simulations).
`seed`	Seed number to ensure reproducibility across runs in case of random elements. Must be an integer. Defaults to `NULL` (no seed).
`quiet`	If `TRUE`, suppress informational messages such as progress and status updates. Warnings and errors are always shown. Defaults to `FALSE`.
`...`	Optional arguments passed to `sim_settings()`; these can be used to override the simulation specifications set in the model object.

Value

Object of class `simulate_stockflow`, a list containing:

object Stock-and-flow model object of class `stockflow`

df Data frame: simulation results (time, variable, value)

init Named vector: initial stock values

constants Named vector: constant parameters

script Character: generated simulation code (R or Julia)

duration Numeric: simulation time in seconds

success Logical: TRUE if completed without errors

error_message NULL if completed without errors

Use as.data.frame() to extract results, plot() to visualize.

See Also

update(), stockflow(), summary(), sim_settings(), use_julia()

Examples

```
sfm <- stockflow("sir")
sim <- simulate(sfm)
plot(sim)

# Obtain all model variables
sim <- simulate(sim_settings(sfm, only_stocks = FALSE))
plot(sim, show_constants = TRUE)
```

---

sim_methods *Translate between deSolve and DifferentialEquations.jl solver names*

Description

Translate between deSolve and DifferentialEquations.jl solver names, or validate that a given solver name is recognized in either language. This is used internally to allow users to specify familiar R solvers when using Julia for simulation, and to provide warnings when an exact equivalent is not available.

Usage

```
sim_methods(method, from = NULL, to = NULL)
```

Arguments

method	Solver name to validate or translate.
from	Source solver family, either "R" or "Julia".
to	Target solver family when translating, either "R" or "Julia".

Value

A character scalar (validated or translated solver name), a character vector of solver names when method is omitted, or a named list of solver names for both languages when called with no arguments.

Examples

```
# List supported solvers
sim_methods()

# List supported R solvers
sim_methods(from = "R")

# List supported Julia solvers
sim_methods(from = "Julia")

# Validate or translate specific solvers
sim_methods("rk4", from = "R", to = "Julia")
```

`sim_settings` *Modify simulation specifications*

Description

Simulation specifications are the settings that determine how the model is simulated, such as the integration method (i.e., solver), start and stop time, and timestep. Modify these specifications for an existing stock-and-flow model.

Usage

```
sim_settings(
  object,
  method = "euler",
  start = 0,
  stop = 100,
  dt = 0.01,
  save_by = NULL,
  save_times = NULL,
  save_length = NULL,
  seed = NULL,
  time_units = "seconds",
  language = "R",
  only_stocks = TRUE,
  vars = NULL,
  keep_nonnegative_stock = FALSE,
  keep_nonnegative_flow = TRUE,
  save_sims = FALSE,
  central = c("mean", "median"),
  spread = "quantile",
  quantiles = c(0.025, 0.975),
  ...
)
```

Arguments

object	Stock-and-flow model, object of class stockflow.
method	Integration method. Defaults to "euler".
start	Start time of simulation. Defaults to 0.
stop	End time of simulation. Defaults to 100.
dt	Timestep of solver; controls simulation accuracy. Smaller = more accurate but slower. Defaults to 0.01.
save_by	Save output at a regular interval, mirroring the by argument of seq(): output times are seq(start, stop, by = save_by). Must be a single number >= dt. Use a value larger than dt to reduce output size without sacrificing accuracy. Example: dt = 0.01, save_by = 1 saves every 100th computed point. Pass NA, NULL, or "" to reset to saving all dt steps. Mutually exclusive with save_times and save_length. Defaults to NULL (save all).
save_times	Numeric vector of explicit output times to save. Values must lie within [start, stop]. Example: save_times = c(0, 10) saves exactly those two times. Pass NA, NULL, or "" to reset to saving all dt steps. Mutually exclusive with save_by and save_length. Defaults to NULL (save all).
save_length	Number of output times to save, mirroring the length.out argument of seq(): output times are seq(start, stop, length.out = save_length). Must be a single whole positive number. save_length = 1 saves only the final time point (stop). Pass NA, NULL, or "" to reset to saving all dt steps. Mutually exclusive with save_by and save_times. Defaults to NULL (save all).
seed	Seed number to ensure reproducibility across runs in case of random elements. Must be an integer. Defaults to NULL (no seed).
time_units	Simulation time unit. Defaults to "seconds".
language	Coding language in which to simulate model. Either "R" or "Julia". Defaults to "R".
only_stocks	If TRUE, only return stocks in output, discarding flows and auxiliaries. If FALSE, flows and auxiliaries are saved, which slows down the simulation. Defaults to TRUE.
vars	Character vector of variable names to save in simulation output. Use this to keep selected flows or auxiliaries without saving every model variable. If specified, this overrides only_stocks. Pass NULL, "", or an empty character vector to clear a previous vars setting.
keep_nonnegative_stock	If TRUE, keeps original non-negativity setting of stocks. Defaults to FALSE.
keep_nonnegative_flow	If TRUE, keeps original non-negativity setting of flows. Defaults to TRUE.
save_sims	If TRUE, individual simulations are retained in ensemble() output. Defaults to FALSE.
central	Central-tendency statistics to compute across simulations in ensemble(). One or more of "mean", "median", or "none" (compute no central tendency). Each requested statistic becomes a column in the ensemble summary. Defaults to c("mean", "median"). Can be overridden per call via the central argument of ensemble().

spread	Spread (uncertainty) statistics to compute across simulations in ensemble(). One or more of "quantile" (quantile columns at the probabilities given by quantiles), "sd" (standard deviation), "range" (minimum and maximum, returned as min/max columns), or "none" (compute no spread). Defaults to "quantile". Can be overridden per call via the spread argument of ensemble().
quantiles	Quantile probabilities to compute when spread includes "quantile", e.g. c(0.025, 0.975). Returned as columns quant1, quant2, ... in the order given. At least two unique values are required. Defaults to c(0.025, 0.975). Can be overridden per call via the quantiles argument of ensemble().
...	Reserved for future use. Passing unknown arguments (including the removed save_at and save_n) raises an error.

Value

A stock-and-flow model object of class stockflow

See Also

sim_methods()

Examples

```
sfm <- stockflow("predator_prey") |>
  sim_settings(start = 0, stop = 50, dt = 0.1)
sim <- simulate(sfm)
plot(sim)

# Change the simulation method to "rk4"
sfm <- sim_settings(sfm, method = "rk4")

# Change the time units to "years", such that one time unit is one year
sfm <- sim_settings(sfm, time_units = "years")

# Save at a regular interval to reduce output size without affecting accuracy
sfm <- sim_settings(sfm, save_by = 1)
sim <- simulate(sfm)
head(as.data.frame(sim))

# Save exactly 11 evenly-spaced time points (t=0, 5, 10, ..., 50)
sfm <- sim_settings(sfm, save_length = 11)
sim <- simulate(sfm)
head(as.data.frame(sim))

# Save only specific time points
sfm <- sim_settings(sfm, save_times = c(0, 25, 50))
sim <- simulate(sfm)
as.data.frame(sim)

# Add stochastic initial condition but specify seed to obtain same result
sfm <- sim_settings(sfm, seed = 1) |>
  update(c(predator, prey), eqn = runif(1, 20, 50))
```

step *Create step function*

Description

Create a step function that jumps from zero to a specified height at a specified time, and remains at that height until the end of the simulation time.

Usage

```
step(times, start, height = 1)
```

Arguments

times	Vector of simulation times
start	Start time of step
height	Height of step, defaults to 1

Details

Equivalent of Step() in Insight Maker

Value

Step interpolation function

See Also

ramp(), pulse(), seasonal()

Examples

```
# Create a simple model with a step function
# that jumps at time 50 to a height of 5
sfm <- stockflow() |>
  update("a", "stock") |>
  # Specify the global variable "times" as simulation times
  update("input", "constant", eqn = "step(times, 50, 5)") |>
  update("inflow", "flow", eqn = "input(t)", to = "a")



sim <- simulate(sfm, only_stocks = FALSE)
plot(sim)

# Negative heights are also possible
sfm <- update(sfm, "input", eqn = "step(times, 50, -10)")
```

```
sim <- simulate(sfm, only_stocks = FALSE)
plot(sim)
```

`stock` *Add or modify stocks*

Description

Stocks accumulate material or information over time, defining the state of the system. `stock()` adds or changes a stock variable. This is a convenience wrapper around `update()` with `type = "stock"`. See the **Stocks** section of `update()` for more details.

Usage

```
stock(object, name, eqn = 0, label = name, doc = "", non_negative = FALSE)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`name`	Variable name. Accepts a bare symbol (e.g., `population`), a string (`"population"`), or a vector via `c()` (e.g., `c(a, b)` or `c("a", "b")`). Use `!!` to inject from a variable.
`eqn`	Equation (or initial value in the case of stocks). Accepts a bare expression (e.g., `a * b + 1`), a string (`"a * b + 1"`), or a numeric value. Use `!!` to inject from a variable. Defaults to `0`.
`label`	Name of variable used for plotting. Defaults to the same as name.
`doc`	Description of variable. Defaults to `""` (no description).
`non_negative`	If TRUE, variable is enforced to be non-negative (i.e., strictly 0 or positive). Defaults to `FALSE`.

Value

A stock-and-flow model object of class `stockflow`

See Also

`update()`, `discard()`, `change_name()`

Examples

```
# Create a stock with an initial value
sfm <- stockflow() |>
  stock(population, eqn = 100, label = "Population")

# Multiple stocks
sfm <- stockflow() |>
```

```
stock(susceptible, eqn = 999, label = "susceptible") |>
stock(infected, eqn = 1, label = "infected") |>
stock(recovered, eqn = 0, label = "recovered")
```

`stockflow` *Create a new stock-and-flow model*

Description

Initialize a stock-and-flow model of class `stockflow`. You can either create an empty stock-and-flow model or load a template from the model library.

Usage

```
stockflow(template = NULL, version = NULL)
```

Arguments

`template` Name of the template to load (case-insensitive). If `NULL`, an empty stock-and-flow model is created with default simulation parameters and a default meta. If specified, `template` should be one of:

- **logistic_model**: Population growth with carrying capacity (v1.0.0)
- **sir**: Epidemic model (Susceptible-Infected-Recovered) (v1.0.0)
- **predator_prey**: Lotka-Volterra dynamics (v1.0.0)
- **cusp**: Cusp catastrophe (v1.0.0)
- **crielaard2022**: Eating behavior (doi: 10.1037/met0000484) (v1.0.0)
- **coffee_cup**: Temperature equilibration (Meadows) (v1.0.0)
- **bank_account**: Compound interest (Meadows) (v1.0.0)
- **lorenz**: Lorenz attractor (chaotic) (v1.0.0)
- **rossler**: Rossler attractor (chaotic) (v1.0.0)
- **vanderpol**: Van der Pol oscillator (v1.0.0)
- **duffing**: Forced Duffing oscillator (v1.0.0)
- **chua**: Chua's circuit (chaotic) (v1.0.0)
- **burnout**: Toy model of burnout (v1.0.0)
- **fitzhugh_nagumo**: Two-variable excitable neuron model (v1.0.0)
- **queue**: Simple demonstration model of people waiting in a queue (v1.0.0)
- **jdr**: Job Demands-Resources Theory (JD-R, version 1.0.0) as formalized in Evers et al. (under review) (v1.0.0)
- **jdr**: Job Demands-Resources Theory (JD-R, version 2.0.0) as formalized in Evers et al. (under review) (v2.0.0)
- **jdr**: Job Demands-Resources Theory (JD-R, version 3.0.0) as formalized in Evers et al. (under review) (v3.0.0)

`version` Optional semantic version of the template to load. Flexible in what it accepts: a number (`1`), a string (`"1.2.1"`), an optional `"v"` prefix (`"v1"`), a `package_version()` value (injected with `!!`), or a bare symbol (`v1.0.0`). Matching is prefix-based: `version = 1` returns the highest `1.x.y`, `version = "1.2"` the highest `1.2.y`, and `version = "1.2.1"` is exact. Defaults to the latest available version.

Details

Do not edit the object manually; this will likely lead to errors downstream. Rather, use `meta()`, `sim_settings()`, `update()`, and `custom_func()` for safe manipulation.

Value

A stock-and-flow model object of class `stockflow`. Its structure is based on XML Interchange Language for System Dynamics (XMILE). It is a nested list, containing:

meta Meta-information about model. A list containing arguments listed in `meta()`.

sim_settings Simulation specifications. A list containing arguments listed in `sim_settings()`.

model Model variables, grouped under the variable types stock, flow, aux (auxiliaries), constant, gf (graphical functions), and func (custom functions). Each variable contains arguments as listed in `update()`.

Use `summary()` to run model diagnostics, `as.data.frame()` to convert to a data.frame, `plot()` to visualize.

See Also

`update()`, `meta()`, `custom_func()`, `sim_settings()`

Examples

```
sfm <- stockflow()
summary(sfm)



# Load a template
sfm <- stockflow("lorenz")
sim <- simulate(sfm)
plot(sim)
```

---

summary.simulate_stockflow

Summarize simulation results

Description

Returns a data frame with per-variable summary statistics (min, mean, max, and final value) over the simulated time range.

Usage

```
## S3 method for class 'simulate_stockflow'
summary(object, ...)
```

Arguments

object	A simulation result of class simulate_stockflow
...	Additional arguments (unused)

Value

A data.frame with columns variable, min, mean, max, final.

See Also

print.simulate_stockflow(), simulate.stockflow()

Examples

```
sfm <- stockflow("sir")
sim <- simulate(sfm)
summary(sim)
```

---

summary.stockflow *Run model diagnostics*

Description

Check for common formulation problems in a stock-and-flow model.

Usage

```
## S3 method for class 'stockflow'
summary(object, ...)
```

Arguments

object	Stock-and-flow model, object of class stockflow.
...	Additional arguments (currently unused).

Details

The following problems are detected:

- An absence of stocks
- Flows without a source (`from`) or target (`to`)
- Flows connected to a stock that does not exist
- Undefined variable references in equations
- Circularity in equations

The following potential problems are detected:

- Absence of flows
- Stocks without inflows or outflows
- Equations with a value of 0

Value

Object of class `summary_stockflow`. A flat named list with one entry per check. Each entry contains a `problem` field (`"none"`, `"warning"`, or `"error"`) and type-specific data fields.

Examples

```
# No issues
sfm <- stockflow("sir")
summary(sfm)

# Detect absence of stocks or flows
sfm <- stockflow()
summary(sfm)

# Detect stocks without inflows or outflows
sfm <- stockflow() |> update("Prey", "stock")
summary(sfm)

# Detect circularity in equation definitions
sfm <- stockflow() |>
  update("Prey", "stock", eqn = "Predator") |>
  update("Predator", "stock", eqn = "Prey")
summary(sfm)
```

tail.simulate_stockflow *Print last rows of a simulation*

Description

Print the last rows of a simulation data frame of a stock-and-flow model. This is a wrapper around `tail()` that first converts the simulation results to a data frame using as.data.frame().

Usage

```
## S3 method for class 'simulate_stockflow'
tail(x, n = 6L, ...)
```

Arguments

x	Output of `simulate()`.
n	Number of rows to print. Defaults to 6.
...	Other arguments passed to `as.data.frame.simulate_stockflow()`.

Value

A data.frame with the last rows of the simulation results.

Examples

```
sfm <- stockflow("sir")
sim <- simulate(sfm)
tail(sim)
```

tail.verify_stockflow *Print last rows of verify results*

Description

Wrapper around `tail()` that first converts the results to a data frame using `as.data.frame.verify_stockflow()`.

Usage

```
## S3 method for class 'verify_stockflow'
tail(x, n = 6L, ...)
```

Arguments

x	A verify_stockflow object.
n	Number of rows. Defaults to 6.
...	Other arguments passed to as.data.frame.verify_stockflow().

Value

A data.frame.

Examples

```
sfm <- stockflow("sir") |>
  unit_test(expr = all(susceptible >= 0))
res <- verify(sfm)
tail(res)
```

---

unit_test *Add or modify unit tests*

Description

Unit tests are assertions about model behavior that can be evaluated against simulation results. For example, you might assert that a stock remains non-negative, or that a certain variable reaches a threshold by the end of the simulation. Unit tests can be added to a model such that they can be evaluated with verify(). All unit tests can be displayed with unit_tests().

Usage

```
unit_test(object, test, expr, label, conditions = list(), active = TRUE)
```

Arguments

object	Stock-and-flow model, object of class stockflow.
test	Integer number of the test to modify. Must be a positive integer (a warning is issued and the value rounded when a non-integer is supplied). When test exceeds the current number of tests a warning is issued and a new test is appended instead. Can be omitted when adding a new test.
expr	An expression to evaluate against simulation results. Variable names in the expression refer to model variables; each resolves to a numeric vector of time-series values. Required when adding a new test; optional when modifying (keeps the current expression if omitted).
label	A descriptive label for the test. If omitted when adding, auto-generated from expr. If omitted when modifying, the current label is kept. Labels must be unique.
conditions	A named list of constant or initial stock overrides used when evaluating this test. If non-empty, verify.stockflow() will re-simulate the model with these parameter values before evaluating expr.
active	If FALSE, the test is defined but skipped during verify(). Defaults to TRUE.

Details

The `expr` argument accepts a plain logical expression:

- **Logical**: `all(S >= 0)`, `cor(D, C) < -.5`.

When `label` is omitted, a human-readable label is generated automatically by parsing the expression (e.g., `all(S >= 0)` → `"S is at least 0 (for all values)"`).

Value

The model object with the unit test added or modified, invisibly.

Adding vs. modifying

- **Add** a new test: omit `test` (and provide a `label` that does not match any existing test, or omit `label` to auto-generate one).
- **Modify** an existing test by number: supply `test` (integer).
- **Modify** an existing test by label: supply a `label` that matches an existing test (without specifying `test`).

When modifying, only the arguments you explicitly supply are changed; all other fields keep their current value.

Uniqueness

Labels must be unique across all unit tests. An error is thrown if a new or modified label would create a duplicate. Expressions must also be unique; an error is thrown if an identical `expr` already exists on another test.

See Also

`verify()`, `unit_tests()`, `discard_unit_test()`

Examples

```
sfm <- stockflow("sir") |>
  unit_test(expr = all(susceptible >= 0))

# Run unit tests
verify(sfm)

# Add test with label
sfm <- unit_test(sfm,
  label = "recovered increases",
  expr = all(diff(recovered) >= 0)
)
verify(sfm)

# Add test with conditions
sfm <- unit_test(sfm,
  expr = all(infected == infected[1]),
```

```
  label = "When infection_rate is zero, no one gets infected",
  conditions = list(infection_rate = 0)
)
verify(sfm)

# View all tests
unit_tests(sfm)

# Deactivate test test 1
sfm <- unit_test(sfm, test = 1, active = FALSE)
verify(sfm)

# Modify test by label, e.g., to change the expression
sfm <- unit_test(sfm,
  label = "recovered increases over time",
  expr = all(diff(recovered) > -1)
)
verify(sfm)
```

`unit_tests` *Display unit tests defined on a stock-and-flow model*

Description

Returns an overview of all unit tests attached to the model. The result has a `print()` method.

Usage

```
unit_tests(object, test = NULL, label = NULL, ignore_case = TRUE)
```

Arguments

`object`	Stock-and-flow model, object of class `stockflow`.
`test`	Integer vector of test number(s) to display (1-based). Defaults to `NULL` (show all tests). Can be combined with `label` (intersection).
`label`	Character vector of regex patterns for partial, case-insensitive label matching. A test is included if its label matches *any* pattern. E.g., `c("non-neg", "beta")` returns tests matching either fragment. Can be combined with `test` (intersection).
`ignore_case`	Logical; whether `label` matching is case-insensitive. Default `TRUE`.

Value

An object of class `unit_tests_stockflow`, printed automatically.

See Also

`unit_test()`, `verify()`

Examples

```
sfm <- stockflow("sir") |>
  unit_test(expr = all(susceptible >= 0)) |>
  unit_test(
    label = "recovered increases over time",
    expr = all(diff(recovered) >= 0)
  )

unit_tests(sfm)
unit_tests(sfm, test = 1L)
unit_tests(sfm, label = "increases")
```

`update.stockflow` *Create or modify variables*

Description

Add or change variables in a stock-and-flow model. Variables may be stocks, flows, constants, auxiliaries, or graphical functions. When creating new variables, only "name", "type", and "eqn" (initial value for stocks) are required. When modifying existing variables, only "name" is required to identify the variable to modify, and any other properties can be updated by including the corresponding arguments.

Usage

```
## S3 method for class 'stockflow'
update(
  object,
  name,
  type = NULL,
  eqn = 0,
  label = name,
  doc = "",
  to = NULL,
  from = NULL,
  non_negative = FALSE,
  xpts = NULL,
  ypts = NULL,
  source = NULL,
  interpolation = "linear",
  extrapolation = "nearest",
  df = NULL,
  ...
)
```

Arguments

object	Stock-and-flow model, object of class stockflow.
name	Variable name. Accepts a bare symbol (e.g., population), a string ("population"), or a vector via c() (e.g., c(a, b) or c("a", "b")). Use !! to inject from a variable.
type	Type of building block(s); accepts a bare symbol or string. One of stock, flow, constant, aux, lookup, or func. Does not need to be specified to modify an existing variable.
eqn	Equation (or initial value in the case of stocks). Accepts a bare expression (e.g., a * b + 1), a string ("a * b + 1"), or a numeric value. Use !! to inject from a variable. Defaults to 0.
label	Name of variable used for plotting. Defaults to the same as name.
doc	Description of variable. Defaults to "" (no description).
to	Target of flow. Accepts a bare symbol or string. Must be a stock in the model. Defaults to NULL to indicate no target.
from	Source of flow. Accepts a bare symbol or string. Must be a stock in the model. Defaults to NULL to indicate no source.
non_negative	If TRUE, variable is enforced to be non-negative (i.e., strictly 0 or positive). Defaults to FALSE.
xpts	Only for graphical functions: vector of x-domain points. Must be of the same length as ypts.
ypts	Only for graphical functions: vector of y-domain points. Must be of the same length as xpts.
source	Only for graphical functions: name of the variable which will serve as the input to the graphical function. Accepts a bare symbol or string. Defaults to NULL.
interpolation	Only for graphical functions: interpolation method. Must be either "constant" or "linear". Defaults to "linear".
extrapolation	Only for graphical functions: extrapolation method. Must be either "nearest" or "NA". Defaults to "nearest".
df	A data.frame with variable properties to add and/or modify. Each row represents one variable to update. Required columns depend on the variable type being created: • All types require: 'type', 'name' • Stocks require: 'eqn' (initial value) • Flows require: 'eqn', and at least one of 'from' or 'to' • Constants require: 'eqn' • Auxiliaries require: 'eqn' • Graphical functions require: 'xpts', 'ypts' Optional columns for all types: 'label', 'doc', 'non_negative' Optional columns for graphical functions: 'source', 'interpolation', 'extrapolation' Columns not applicable to a variable type should be set to NA. See Examples for a complete demonstration.
...	Additional arguments (currently unused).

Value

A stock-and-flow model object of class `stockflow`

Stocks

Stocks define the state of the system. They accumulate material or information over time, such as people, products, or beliefs, which creates memory and inertia in the system. As such, stocks need not be tangible. Stocks are variables that can increase and decrease, and can be measured at a single moment in time. The value of a stock is increased or decreased by flows. A stock may have multiple inflows and multiple outflows. The net change in a stock is the sum of its inflows minus the sum of its outflows.

The obligatory properties of a stock are "name", "type", and "eqn". Optional additional properties are "label", "doc", "non_negative".

Flows

Flows move material and information through the system. Stocks can only decrease or increase through flows. A flow must flow from and/or flow to a stock. If a flow is not flowing from a stock, the source of the flow is outside of the model boundary. Similarly, if a flow is not flowing to a stock, the destination of the flow is outside the model boundary. Flows are defined in units of material or information moved over time, such as birth rates, revenue, and sales.

The obligatory properties of a flow are "name", "type", "eqn", and either "from", "to", or both. Optional additional properties are "label", "doc", "non_negative".

Constants

Constants are variables that do not change over the course of the simulation - they are time-independent. These may be numbers, but also functions. They can depend only on other constants.

The obligatory properties of a constant are "name", "type", and "eqn". Optional additional properties are "label", "doc", "non_negative".

Auxiliaries

Auxiliaries are dynamic variables that change over time. They are used for intermediate calculations in the system, and can depend on other flows, auxiliaries, constants, and stocks.

The obligatory properties of an auxiliary are "name", "type", and "eqn". Optional additional properties are "label", "doc", "non_negative".

Graphical functions

Graphical functions, also known as table or lookup functions, are interpolation functions used to define the desired output (y) for a specified input (x). They are defined by a set of x- and y-domain points, which are used to create a piecewise linear function. The interpolation method defines the behavior of the graphical function between x-points ("constant" to return the value of the previous x-point, "linear" to linearly interpolate between defined x-points), and the extrapolation method defines the behavior outside of the x-points ("NA" to return NA values outside of defined x-points, "nearest" to return the value of the closest x-point).

The obligatory properties of a graphical function are "name", "type", "xpts", and "ypts". "xpts" and "ypts" must be of the same length. Optional additional properties are "label", "doc", "source", "interpolation", "extrapolation".

Non-standard evaluation (NSE)

The `name`, `type`, `eqn`, `to`, `from`, and `source` arguments support non-standard evaluation. This means you can pass bare symbols and expressions instead of quoted strings:

```
# These are equivalent:
stock(sfm, "population", eqn = "birth_rate * 0.1")
stock(sfm, population, eqn = birth_rate * 0.1)
```

To inject the value of a variable (rather than its name), use the `!!` (bang-bang) operator from rlang:

```
my_name <- "population"
stock(sfm, !!my_name, eqn = 100)
```

The `label`, `doc`, `non_negative`, `xpts`, `ypts`, `interpolation`, and `extrapolation` arguments are not affected by NSE and are evaluated normally.

See Also

`stockflow()` to initialize a model, `simulate()` to simulate a model, and `summary()` to run model diagnostics. Variable-specific helper functions `stock()`, `flow()`, `constant()`, `aux()`, and `lookup()` are also available as wrappers around update() that set the "type" argument for convenience. Further helper functions for modifying models are `change_name()` to rename a variable, `change_type()` to change a variable's type, and `discard()` to remove a variable.

Examples

```
# First initialize an empty model
sfm <- stockflow()
print(sfm)


# Add two stocks. Specify their initial values in the "eqn" property
# and their plotting label.
sfm <- stock(sfm, predator, eqn = 10, label = "Predator") |>
  stock(prey, eqn = 50, label = "Prey")


# Add four flows: the births and deaths of both the predators and prey. The
# "eqn" property of flows represents the rate of the flow. In addition, we
# specify which stock the flow is coming from ("from") or flowing to ("to").
sfm <- flow(sfm, predator_births,
  eqn = delta * prey * predator,
  label = "Predator Births", to = predator
) |>
  flow(predator_deaths,
```

```
    eqn = gamma * predator,
    label = "Predator Deaths", from = predator
  ) |>
  flow(prey_births,
    eqn = alpha * prey,
    label = "Prey Births", to = prey
  ) |>
  flow(prey_deaths,
    eqn = beta * prey * predator,
    label = "Prey Deaths", from = prey
  )
plot(sfm)

# The flows make use of four other variables: "delta", "gamma", "alpha", and
# "beta". Define these as constants in a vectorized manner for efficiency.
sfm <- constant(sfm, c(delta, gamma, alpha, beta),
  eqn = c(.025, .5, .5, .05),
  label = c("Delta", "Gamma", "Alpha", "Beta"),
  doc = c(
    "Birth rate of predators", "Death rate of predators",
    "Birth rate of prey", "Death rate of prey by predators"
  )
)

# We now have a complete predator-prey model which is ready to be simulated.
sim <- simulate(sfm)
plot(sim)

# Modify a variable - note that we no longer need to specify type
sfm <- update(sfm, delta, eqn = .03, label = "DELTA")

# To add and/or modify variables more quickly, pass a data.frame.
# The data.frame is processed per row.
# For instance, to create a logistic population growth model:
df <- data.frame(
  type = c("stock", "flow", "flow", "constant", "constant"),
  name = c("X", "inflow", "outflow", "r", "K"),
  eqn = c(.01, "r * X", "r * X^2 / K", 0.1, 1),
  label = c(
    "Population size", "Births", "Deaths", "Growth rate",
    "Carrying capacity"
  ),
  to = c(NA, "X", NA, NA, NA),
  from = c(NA, NA, "X", NA, NA)
)
sfm <- update(stockflow(), df = df)

# Run model diagnostics
summary(sfm)

# --- Programmatic usage ---

# To inject the value of an R variable, use !! (bang-bang)
```

```
my_name <- "growth"
sfm <- constant(sfm, !!my_name, eqn = 0.1)

# Strings also work
sfm <- constant(sfm, "growth", eqn = 0.2)
```

`url_to_insightmaker` *Extract Insight Maker model from URL*

Description

Create XML string from Insight Maker URL. This function is for internal use; use `import_insightmaker()` to import an Insight Maker model.

Usage

```
url_to_insightmaker(url, file = NULL)
```

Arguments

`url`	String with URL to an Insight Maker model
`file`	If specified, file path to save Insight Maker model to. If NULL, do not save model.

Value

XML string with Insight Maker model

See Also

`import_insightmaker()`

Examples

```
url <- "https://insightmaker.com/insight/43tz1nvUgbIiIOGSGtzIzj/Romeo-Juliet"
xml <- url_to_insightmaker(url)

# Save model to file
file <- tempfile(fileext = ".InsightMaker")
xml <- url_to_insightmaker(url, file = file)
file.remove(file)
```

use_julia *Start Julia and activate environment*

Description

Start Julia session and activate Julia environment to simulate stock-and-flow models. To do so, Julia needs to be installed (see https://julialang.org/install/) and findable from within R. See this vignette for guidance. In addition, the Julia environment specifically for sdbuildR needs to have been instantiated. This can be set up with install_julia_env().

Usage

```
use_julia(stop = FALSE, restart = FALSE, nthreads = NULL, quiet = FALSE)
```

Arguments

stop	If TRUE, stop active Julia session. Defaults to FALSE.
restart	If TRUE, force Julia session to restart.
nthreads	If not NULL, set the number of threads for Julia to use. This will temporarily set the environment variable JULIA_NUM_THREADS and restart Julia if it is already running to apply the new thread setting. See this page for more details on threading in Julia.
quiet	If TRUE, suppress informational messages such as progress and status updates. Warnings and errors are always shown. Defaults to FALSE.

Details

In every R session, use_julia() needs to be run once (which is done automatically in simulate()), which takes under 10 seconds while Julia starts and loads the packages.

Value

Returns TRUE invisibly after setting up the Julia environment, and NULL invisibly if Julia was already set up or when stopping Julia with stop = TRUE. Used for side effects.

See Also

install_julia_env()

Examples

```
# Start a Julia session and activate the Julia environment for sdbuildR
use_julia()

# Start Julia with 2 threads (only works if threading is supported)
use_julia(nthreads = 2)
```

```
# Restart Julia session (in case of issues)
use_julia(restart = TRUE)

# Stop Julia session
use_julia(stop = TRUE)
```

`verify` *Verify model behavior with unit tests*

Description

Verify model behavior with unit tests

Usage

```
verify(object, ...)
```

Arguments

`object`	A model object to verify.
`...`	Additional arguments passed to specific methods.

`verify.stockflow` *Verify unit tests against simulation results*

Description

Run all active unit tests defined on a stock-and-flow model. Use `unit_test()` to define tests; use `unit_tests()` to display them.

Usage

```
## S3 method for class 'stockflow'
verify(object, test = NULL, ...)
```

Arguments

`object`	An `stockflow` object.
`test`	Integer vector of test number(s) to run (numbers-based, as shown by `unit_tests()`). Defaults to `NULL` (run all tests).
`...`	Additional arguments passed to `sim_settings()` (e.g., `seed`, `dt`).

Details

Calling `verify()` on a `stockflow` model will first simulate the model, then run all tests — including those that require re-simulation under alternative conditions. Simulations are always retained in the returned object so that `plot.verify_stockflow()` works without any extra arguments.

For repeated-run robustness testing use `ensemble()` instead.

Value

An object of class `verify_stockflow`, returned invisibly. Use `as.data.frame()` to extract results as a data frame and `plot()` to visualize the simulations used. The object contains:

results List of test result entries, one per test (including inactive tests, which appear with `status = "skip"`). Each entry has `label`, `expr_str`, `conditions`, `status`, `error_type`, `message`, and `outcome`.

object The `stockflow` model the tests were run against.

sims Nested list of `simulate_stockflow` objects used internally by `plot.verify_stockflow()`. Always present (never `NULL`).

condition Named integer vector mapping each test label to its condition index. Used internally by `plot.verify_stockflow()`.

n_conditions Number of unique simulation conditions.

test_indices Integer vector of the original 1-based test numbers that were run (as shown by `unit_tests()`). Equal to `seq_along(results)` when `test = NULL` (all tests run).

See Also

`unit_test()`, `unit_tests()`, `simulate.stockflow()`, `as.data.frame.verify_stockflow()`, `plot.verify_stockflow()`

Examples

```
sfm <- stockflow("sir") |>
  unit_test(expr = all(susceptible >= 0)) |>
  unit_test(
    label = "recovered increases over time",
    expr = all(diff(recovered) >= 0)
  )

verify(sfm)
```

Index