Package 'SimInf'

September 8, 2026

Title A Framework for Data-Driven Stochastic Disease Spread Simulations

Version 11.1.0

Description Provides an efficient and very flexible framework to conduct data-driven epidemiological modeling in realistic large scale disease spread simulations. The framework integrates infection dynamics in subpopulations as continuous-time Markov chains using the Gillespie stochastic simulation algorithm and incorporates available data such as births, deaths and movements as scheduled events at predefined time-points. Using C code for the numerical solvers and 'OpenMP' (if available) to divide work over multiple processors ensures high performance when simulating a sample outcome. One of our design goals was to make the package extendable and enable usage of the numerical solvers from other R extension packages in order to facilitate complex epidemiological research. The package contains template models and can be extended with user-defined models. For more details see the paper by Widgren, Bauer, Eriksson and Engblom (2019) <doi:10.18637/jss.v091.i12>. The package also provides functionality to fit models to time series data using the Approximate Bayesian Computation Sequential Monte Carlo ('ABC-SMC') algorithm of Toni and others (2009) <doi:10.1098/rsif.2008.0172> or the Particle Markov Chain Monte Carlo ('PMCMC') algorithm of 'Andrieu' and others (2010) <doi:10.1111/j.1467-9868.2009.00736.x>.

Acknowledgements This software has been made possible by support from the Swedish Research Council within the UPMARC Linnaeus center of Excellence (Pavol Bauer, Robin Eriksson, and Stefan Engblom), the Swedish Research Council Formas (Stefan Engblom and Stefan Widgren), the Swedish Board of Agriculture (Stefan Widgren), the Swedish strategic research program eSSENCE (Stefan Widgren), and in the framework of the Full Force project, supported by funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 773830: One Health European Joint Programme (Stefan Widgren).

License GPL-3

URL https://github.com/stewid/SimInf, https://stewid.github.io/SimInf/

BugReports https://github.com/stewid/SimInf/issues

Type Package

LazyData true

Biarch true

NeedsCompilation yes

SystemRequirements GNU Scientific Library (GSL)

Depends R (>= 4.0)

Imports digest, graphics, grDevices, MASS, methods, stats, utils, Matrix (>= 1.3-0)

Suggests knitr, rmarkdown

Collate 'C-generator.R' 'check_arguments.R' 'init.R' 'valid.R' 'classes.R' 'SimInf_model.R' 'SEIR.R' 'SIR.R' 'SIS.R' 'SISe.R' 'SISe3.R' 'SISe3_sp.R' 'SISe_sp.R' 'SimInf-package.R' 'SimInf.R' 'SimInf_events.R' 'SimInf_individual_events.R' 'run.R' 'density_ratio.R' 'abc.R' 'degree.R' 'distance.R' 'distributions.R' 'edge_properties.R' 'lambert.R' 'match_compartments.R' 'mparse.R' 'pmcmc.R' 'pfilter.R' 'n.R' 'openmp.R' 'package_skeleton.R' 'plot.R' 'prevalence.R' 'print.R' 'punchcard.R' 'trajectory.R' 'u0.R' 'v0.R'

Encoding UTF-8

VignetteBuilder utils, knitr

Config/roxygen2/version 8.1.0

Author Stefan Widgren [aut, cre] (ORCID: <https://orcid.org/0000-0001-5745-2284>), Robin Eriksson [aut] (ORCID: <https://orcid.org/0000-0002-4291-712X>), Stefan Engblom [aut] (ORCID: <https://orcid.org/0000-0002-3614-1732>), Pavol Bauer [aut] (ORCID: <https://orcid.org/0000-0003-4328-7171>), Thomas Rosendal [ctb] (ORCID: <https://orcid.org/0000-0002-6576-9668>), Ivana Rodriguez Ewerlöf [ctb] (ORCID: <https://orcid.org/0000-0002-9678-9813>), Attractive Chaos [cph] (Author of 'kvec.h'.)

Maintainer Stefan Widgren <stefan.widgren@gmail.com>

Repository CRAN

Date/Publication 2026-09-08 17:00:16 UTC

Contents

abc 5
add_spatial_coupling_to_ldata 8

as.data.frame.SimInf_abc . . . 10
as.data.frame.SimInf_events . . . 11
as.data.frame.SimInf_individual_events . . . 12
as.data.frame.SimInf_pmcmc . . . 13
boxplot,SimInf_model-method . . . 14
continue_abc . . . 15
continue_pmcmc . . . 16
C_code . . . 18
distance_matrix . . . 19
edge_properties_to_matrix . . . 20
events . . . 21
events_SEIR . . . 22
events_SIR . . . 24
events_SIS . . . 26
events_SISe3 . . . 28
gdata . . . 31
gdata<- . . . 32
get_individuals . . . 33
indegree . . . 34
individual_events . . . 35
lambertW0 . . . 36
ldata . . . 37
length,SimInf_pmcmc-method . . . 38
logLik,SimInf_pfilter-method . . . 38
mparse . . . 39
nodes . . . 42
node_events . . . 43
n_compartments . . . 44
n_generations . . . 45
n_nodes . . . 46
n_replicates . . . 47
outdegree . . . 48
package_skeleton . . . 49
pairs,SimInf_model-method . . . 50
pfilter . . . 51
plot,SimInf_abc-method . . . 53
plot,SimInf_events-method . . . 54
plot,SimInf_individual_events-method . . . 54
plot,SimInf_model-method . . . 55
plot,SimInf_pfilter-method . . . 57
plot,SimInf_pmcmc-method . . . 57
pmcmc . . . 58
prevalence . . . 61
prevalence,SimInf_model-method . . . 62
prevalence,SimInf_pfilter-method . . . 64
prevalence,SimInf_pmcmc-method . . . 65
punchcard<- . . . 66
run . . . 68

SEIR . . . 70
SEIR-class . . . 72
select_matrix . . . 73
select_matrix<- . . . 73
set_num_threads . . . 74
shift_matrix . . . 75
shift_matrix<- . . . 76
show,SimInf_abc-method . . . 77
show,SimInf_events-method . . . 78
show,SimInf_individual_events-method . . . 79
show,SimInf_model-method . . . 79
show,SimInf_pfilter-method . . . 80
show,SimInf_pmcmc-method . . . 81
SimInf . . . 82
SimInf_abc-class . . . 84
SimInf_events . . . 85
SimInf_events-class . . . 87
SimInf_individual_events-class . . . 89
SimInf_model . . . 90
SimInf_model-class . . . 92
SimInf_pfilter-class . . . 93
SimInf_pmcmc-class . . . 94
SIR . . . 95
SIR-class . . . 97
SIS . . . 97
SIS-class . . . 99
SISe . . . 100
SISe-class . . . 103
SISe3 . . . 104
SISe3-class . . . 108
SISe3_sp . . . 110
SISe3_sp-class . . . 113
SISe_sp . . . 115
SISe_sp-class . . . 118
summary,SimInf_abc-method . . . 120
summary,SimInf_events-method . . . 121
summary,SimInf_individual_events-method . . . 122
summary,SimInf_model-method . . . 123
summary,SimInf_pfilter-method . . . 124
summary,SimInf_pmcmc-method . . . 125
trajectory . . . 125
trajectory,SimInf_model-method . . . 126
trajectory,SimInf_pfilter-method . . . 128
u0 . . . 129
u0<- . . . 130
u0_from_individual_events . . . 132
u0_SEIR . . . 133
u0_SIR . . . 135

u0_SIS 136
u0_SISe3 138
v0<- 140

Index **143**

---

abc *Approximate Bayesian computation*

Description

Approximate Bayesian computation

Usage

```
abc(
  model,
  priors = NULL,
  n_particles = NULL,
  n_init = NULL,
  distance = NULL,
  tolerance = NULL,
  data = NULL,
  verbose = FALSE,
  post_gen = NULL,
  init_model = NULL
)

## S4 method for signature 'SimInf_model'
abc(
  model,
  priors = NULL,
  n_particles = NULL,
  n_init = NULL,
  distance = NULL,
  tolerance = NULL,
  data = NULL,
  verbose = FALSE,
  post_gen = NULL,
  init_model = NULL
)
```

Arguments

`model`	The `SimInf_model` object to generate data from.
`priors`	The priors for the parameters to fit. Each prior is specified with a formula notation, for example, `beta ~ uniform(0, 1)` specifies that beta is uniformly distributed between 0 and 1. Use `c()` to provide more than one prior, for example,

c(beta ~ uniform(0, 1), gamma ~ normal(10, 1)). The following distributions are supported: gamma, lognormal, normal and uniform. All parameters in priors must be only in either gdata or ldata.

n_particles — An integer (>1) specifying the number of particles to approximate the posterior with.

n_init — Specify a positive integer (>n_particles) to adaptively select a sequence of tolerances using the algorithm of Simola and others (2021). The initial tolerance is adaptively selected by sampling n_init draws from the prior and then retain the n_particles particles with the smallest distances. Note there must be enough initial particles to satisfactorily explore the parameter space, see Simola and others (2021). If the tolerance parameter is specified, then n_init must be NULL.

distance — A function for calculating the summary statistics for a simulated trajectory. For each particle, the function must determine the distance and return that information. The first argument, result, passed to the distance function is the result from a run of the model with one trajectory attached to it. The second argument, generation, to distance is an integer with the generation of the particle(s). Further arguments that can passed to the distance function comes from ... in the abc function. Depending on the underlying model structure, data for one or more particles have been generated in each call to distance. If the model only contains one node and all the parameters to fit are in ldata, then that node will be replicated and each of the replicated nodes represent one particle in the trajectory (see 'Examples'). On the other hand if the model contains multiple nodes or the parameters to fit are contained in gdata, then the trajectory in the result argument represents one particle. The function can return a numeric matrix (number of particles $\times$ number of summary statistics). Or, if the distance contains one summary statistic, a numeric vector with the length equal to the number of particles. Note that when using adaptive tolerance selection, only one summary statistic can be used, i.e., the function must return a matrix (number of particles $\times$ 1) or a numeric vector.

tolerance — A numeric matrix (number of summary statistics $\times$ number of generations) where each column contains the tolerances for a generation and each row contains a sequence of gradually decreasing tolerances. Can also be a numeric vector if there is only one summary statistic. The tolerance determines the number of generations of ABC-SMC to run. If the n_init parameter is specified, then tolerance must be NULL.

data — Optional data to be passed to the distance function. Default is NULL.

verbose — prints diagnostic messages when TRUE. Default is FALSE.

post_gen — An optional function that, if non-NULL, is applied after each completed generation. The function must accept one argument of type SimInf_abc with the current state of the fitting process. This function can be useful to, for example, save and inspect intermediate results.

init_model — An optional function that, if non-NULL, is applied before running each proposal. The function must accept one argument of type SimInf_model with the current model of the fitting process. This function can be useful to specify the initial state of u0 or v0 of the model before running a trajectory with proposed parameters.

Value

A SimInf_abc object.

References

T. Toni, D. Welch, N. Strelkowa, A. Ipsen, and M. P. H. Stumpf. Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface* **6**, 187–202, 2009. doi:10.1098/rsif.2008.0172

U. Simola, J. Cisewski-Kehe, M. U. Gutmann, J. Corander. Adaptive Approximate Bayesian Computation Tolerance Selection. *Bayesian Analysis*, **16**(2), 397–423, 2021. doi:10.1214/20BA1211

See Also

continue_abc.

Examples

```
## Not run:
## Let us consider an SIR model in a closed population with N = 100
## individuals of whom one is initially infectious and the rest are
## susceptible. First, generate one realisation (with a specified
## seed) from the model with known parameters \code{beta = 0.16} and
## \code{gamma = 0.077}. Then, use \code{abc} to infer the (known)
## parameters from the simulated data.
model <- SIR(u0 = data.frame(S = 99, I = 1, R = 0),
             tspan = 1:100,
             beta = 0.16,
             gamma = 0.077)

## Run the SIR model and plot the number of infectious.
set.seed(22)
infectious <- trajectory(run(model), "I")$I
plot(infectious, type = "s")

## The distance function to accept or reject a proposal. Each node
## in the simulated trajectory (contained in the 'result' object)
## represents one proposal.
distance <- function(result, ...) {
    ## Extract the time-series of infectious in each node as a
    ## data.frame.
    sim <- trajectory(result, "I")

    ## Split the 'sim' data.frame by node and calculate the sum of the
    ## squared distance at each time-point for each node.
    dist <- tapply(sim$I, sim$node, function(sim_infectious) {
        sum((infectious - sim_infectious)^2)
    })

    ## Return the distance for each node. Each proposal will be
    ## accepted or rejected depending on if the distance is less than
    ## the tolerance for the current generation.
```

```
    dist
}

## Fit the model parameters using ABC-SMC and adaptive tolerance
## selection. The priors for the parameters are specified using a
## formula notation. Here we use a uniform distribtion for each
## parameter with lower bound = 0 and upper bound = 1. Note that we
## use a low number particles here to keep the run-time of the example
## short. In practice you would want to use many more to ensure better
## approximations.
fit <- abc(model = model,
           priors = c(beta ~ uniform(0, 1), gamma ~ uniform(0, 1)),
           n_particles = 100,
           n_init = 1000,
           distance = distance,
           verbose = TRUE)

## Print a brief summary.
fit

## Display the ABC posterior distribution.
plot(fit)

## End(Not run)
```

`add_spatial_coupling_to_ldata` *Add spatial coupling information to local data*

Description

A utility function to augment local model parameters (`ldata`) with spatial coupling data from neighboring nodes.

Usage

```
add_spatial_coupling_to_ldata(
  x,
  y,
  cutoff,
  ldata = NULL,
  min_dist = NULL,
  na_fail = TRUE
)
```

Arguments

`x`	Numeric vector of projected x coordinates for each node.
`y`	Numeric vector of projected y coordinates for each node.

`cutoff`	Numeric scalar. The maximum distance for considering two nodes as neighbors. Pairs of nodes farther apart than this value are excluded from the neighbor data.
`ldata`	local data for the nodes. Can either be specified as a `data.frame` with one row per node. Or as a matrix where each column `ldata[, j]` contains the local data vector for the node `j`. The local data vector is passed as an argument to the transition rate functions and the post time step function.
`min_dist`	Numeric scalar. The value to use for the distance between two nodes if their coordinates are identical (distance = 0). This prevents division by zero errors in downstream calculations (e.g., inverse distance weighting). If `NULL` (default) and identical coordinates are found, an error is raised.
`na_fail`	Logical. If `TRUE` (default), missing values (`NA`) in `x` or `y` will raise an error. If `FALSE`, distances involving missing coordinates are set to zero.

Details

The function calculates distances between nodes based on projected coordinates (`x`, `y`) and appends neighbor data to the `ldata` matrix for each node.

Output Format: The returned matrix has the same number of rows as the input `ldata`. The columns are organized as follows:

- **Local Parameters**: The first n columns correspond to the original local parameters passed in `ldata`.
- **Neighbor Pairs**: Following the local parameters, the data is stored as pairs of columns: `(neighbor_index, distance)`. The `neighbor_index` is a zero-based index of the neighbor node. The `distance` is the Euclidean distance to that neighbor.
- **Stop Marker**: Each node's neighbor list is terminated by a pair `(-1, 0)` in the position where the next `neighbor_index` would appear. This marker appears exactly once per node, immediately after the last neighbor.

Value

A numeric matrix with the same number of rows as `ldata`, but with additional columns containing the neighbor indices and distances as described in the 'Output Format' section.

See Also

`distance_matrix` for computing pairwise distances between nodes, and `edge_properties_to_matrix` for a similar utility that converts edge properties to a matrix format.

Examples

```
## Generate a 5 x 5 grid of nodes separated by 1000m.
nodes <- expand.grid(
  x = seq(from = 0, to = 4000, by = 1000),
  y = seq(from = 0, to = 4000, by = 1000)
)

## Create local data with one parameter per node.
```

```
ldata <- matrix(0.1, nrow = 1, ncol = nrow(nodes))

## Add spatial coupling with a 2500m cutoff.
ldata_augmented <- add_spatial_coupling_to_ldata(
  x = nodes$x,
  y = nodes$y,
  cutoff = 2500,
  ldata = ldata
)

## Inspect the result for the first node.
ldata_augmented[, 1]
```

`as.data.frame.SimInf_abc` *Coerce a* `SimInf_abc` *object to a* `data.frame`

Description

Convert the results of an Approximate Bayesian Computation (ABC) analysis into a single `data.frame`. This function extracts the particle parameters, their acceptance weights, and the generation number for every particle across all generations.

Usage

```
## S3 method for class 'SimInf_abc'
as.data.frame(x, ...)
```

Arguments

`x`	A `SimInf_abc` object.
`...`	Additional arguments (currently ignored).

Details

The resulting `data.frame` has one row per particle. The columns include:

- `generation`: The generation number (integer).
- `weight`: The normalized weight of the particle (numeric).
- **Parameter columns**: One column for each parameter estimated in the ABC analysis (e.g., `beta`, `gamma`, `sigma`). The column names match the parameter names defined in the `priors`.

Value

A `data.frame` containing all particles from all generations.

See Also

`abc` for running the ABC analysis, `SimInf_abc` for the class definition, and `continue_abc` for continuing an existing ABC run.

`as.data.frame.SimInf_events` *Coerce a* `SimInf_events` *object to a* `data.frame`

Description

Convert the scheduled events stored in a `SimInf_events` object into a `data.frame`. This function extracts the event type, time, source node, destination node, number of individuals, proportion, and the specific columns from the select (`E`) and shift (`N`) matrices that define how each event modifies the compartment state. The resulting `data.frame` has one row per scheduled event.

Usage

```
## S3 method for class 'SimInf_events'
as.data.frame(x, ...)
```

Arguments

`x`	A `SimInf_events` object.
`...`	Additional arguments (currently ignored).

Value

A `data.frame` with columns:

- `event`: Event type (integer or character, depending on input).
- `time`: Time of the event (integer or `Date`, depending on input).
- `node`: Source node identifier.
- `dest`: Destination node identifier (may be `NA`).
- `n`: Number of individuals affected.
- `proportion`: Proportion of the population affected (if applicable).
- `select`: The column vector from the select matrix (`E`) that defines how the event modifies the compartment state.
- `shift`: The column vector from the shift matrix (`N`) that defines how the event modifies the compartment state.

See Also

`SimInf_events` for the class definition and `events` for extracting events from a model.

Examples

```
## Create an 'SIR' model with 1600 cattle herds (nodes) and
## initialize it to run over 4*365 days. Define 'tspan' to record
## the state of the system at daily time-points. Load scheduled
## events for the population of nodes with births, deaths and
## between-node movements of individuals.
model <- SIR(
  u0     = u0_SIR(),
  tspan  = seq(from = 1, to = 4*365, by = 1),
  events = events_SIR(),
  beta   = 0.16,
  gamma  = 0.01
)

## Extract the events from the model and convert to a data frame.
head(as.data.frame(events(model)))
```

---

as.data.frame.SimInf_individual_events

Coerce a `SimInf_individual_events` *object to a* `data.frame`

Description

Convert the cleaned individual-level events stored in a `SimInf_individual_events` object into a `data.frame`. This function extracts the individual identifier, event type, time, source node, and destination node. The resulting `data.frame` has one row per event.

Usage

```
## S3 method for class 'SimInf_individual_events'
as.data.frame(x, ...)
```

Arguments

`x`	A `SimInf_individual_events` object.
`...`	Additional arguments (currently ignored).

Value

A `data.frame` with columns:

- `id`: Identifier of the individual (integer or character, depending on input).
- `event`: Event type (integer or character, depending on input).
- `time`: Time of the event (numeric or `Date`, depending on input).
- `node`: Source node identifier (integer or character, depending on input).
- `dest`: Destination node identifier (integer or character, depending on input) (may be `NA`).

See Also

SimInf_individual_events for the class definition and individual_events for cleaning livestock event data and prepare it for usage in SimInf.

`as.data.frame.SimInf_pmcmc`
Coerce a `SimInf_pmcmc` *object to a* `data.frame`

Description

Extract the posterior samples from the MCMC chain stored in a `SimInf_pmcmc` object and convert them into a `data.frame`.

Usage

```
## S3 method for class 'SimInf_pmcmc'
as.data.frame(x, ...)
```

Arguments

`x`	A `SimInf_pmcmc` object.
`...`	Additional arguments (currently ignored).

Details

The resulting `data.frame` contains one row per MCMC iteration and one column per parameter. These samples represent the joint posterior distribution of the parameters. This format is convenient for post-processing and visualization.

Value

A `data.frame` where rows represent MCMC iterations and columns represent the posterior samples of the parameters.

See Also

pmcmc for running the PMCMC analysis, SimInf_pmcmc for the class definition, and continue_pmcmc for continuing an existing PMCMC run.

`boxplot,SimInf_model-method`
Box plot of number of individuals in each compartment

Description

Produce box-and-whisker plots summarizing the distribution of the number of individuals in specified model compartments. The plots aggregate data across all time points in `tspan` and the selected nodes (specified by `index`).

Usage

```
## S4 method for signature 'SimInf_model'
boxplot(x, compartments = NULL, index = NULL, ...)
```

Arguments

`x`	The `SimInf_model` object containing the trajectory data.
`compartments`	Specify the names of the compartments to include. Can be a character vector (e.g., `c("S", "I","R")`) or a formula (e.g., `~S + I + R`). If `NULL` (default), all compartments in the model are included.
`index`	Indices of the nodes to include in the plot. If `NULL` (default), data from all nodes are pooled together. If a vector (e.g., `1:2`), only data from those specific nodes are used.
`...`	Additional graphical arguments passed to `boxplot()`.

Details

This function is useful for visualizing the variability and range of compartment counts across the entire simulation trajectory.

Examples

```
## For reproducibility, specify the number of threads.
set_num_threads(1)

## Create an 'SIR' model with 10 nodes.
model <- SIR(
  u0 = data.frame(
    S = rep(99, 10),
    I = rep(1, 10),
    R = rep(0, 10)
  ),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)
```

```
## Run the model with a fixed seed for reproducibility.
result <- run(model, seed = 22)

## Create a boxplot for all compartments across all nodes and time
## points.
boxplot(result)

## Create a boxplot for the S and I compartments, but only for
## nodes 1 and 2.
boxplot(result, compartments = c("S", "I"), index = 1:2)
```

`continue_abc` *Run more generations of ABC SMC*

Description

Run more generations of ABC SMC

Usage

```
continue_abc(
  object,
  tolerance = NULL,
  data = NULL,
  verbose = FALSE,
  post_gen = NULL
)

## S4 method for signature 'SimInf_abc'
continue_abc(
  object,
  tolerance = NULL,
  data = NULL,
  verbose = FALSE,
  post_gen = NULL
)
```

Arguments

`object`	The `SimInf_abc` object to continue from.
`tolerance`	A numeric matrix (number of summary statistics $\times$ number of generations) where each column contains the tolerances for a generation and each row contains a sequence of gradually decreasing tolerances. Can also be a numeric vector if there is only one summary statistic. The tolerance determines the number of generations of ABC-SMC to run.
`data`	Optional data to be passed to the `SimInf_abc@fn` function. Default is `NULL`.

verbose — prints diagnostic messages when TRUE. Default is FALSE.

post_gen — An optional function that, if non-NULL, is applied after each completed generation. The function must accept one argument of type SimInf_abc with the current state of the fitting process. This function can be useful to, for example, save and inspect intermediate results.

Value

A SimInf_abc object.

---

continue_pmcmc *Continue PMCMC from an Existing Chain*

Description

Extends a previously computed PMCMC chain by running additional iterations of the Metropolis-Hastings sampler, continuing from the final state of the previous chain.

Usage

```
continue_pmcmc(
  object,
  obs_process,
  n_iterations,
  post_proposal = NULL,
  init_model = NULL,
  post_particle = NULL,
  verbose = FALSE
)

## S4 method for signature 'SimInf_pmcmc'
continue_pmcmc(
  object,
  obs_process,
  n_iterations,
  post_proposal = NULL,
  init_model = NULL,
  post_particle = NULL,
  verbose = FALSE
)
```

Arguments

object — A SimInf_pmcmc object from a previous pmcmc or continue_pmcmc call.

obs_process
: Specification of the stochastic observation process. The obs_process can be specified as a formula if the model contains only one node and there is only one data point for each time in data. The left hand side of the formula must match a column name in the data data.frame and the right hand side of the formula is a character specifying the distribution of the observation process, for example, Iobs ~ poisson(I). The following distributions are supported: x ~ binomial(size, prob), x ~ poisson(rate) and x ~ uniform(min, max). The observation process can also be a function to evaluate the probability density of the observations given the simulated states. The first argument passed to the obs_process function is the result from a run of the model and it contains one trajectory with simulated data for a time-point, where the trajectory contains n_particles replicates, see trajectory,SimInf_model-method. The second argument to the obs_process function is a data.frame containing the rows for the specific time-point that the function is called for. Note that the function must return the log of the density.

n_iterations
: An integer specifying the number of iterations to run the PMCMC.

post_proposal
: An optional function that, if non-NULL, is applied on the model after the proposal has been set for the model, but before running the particle filter. The function must accept one argument of type SimInf_model with the current model of the fitting process. This function can be useful to update, for example, ldata of the model before running a trajectory with proposed parameters. The function must return the model object which is then used in the particle filter.

init_model
: Optional function applied before each particle filter run. Must accept a SimInf_model object and return the modified model. Useful for setting initial states (u0, v0) before running trajectories with proposed parameters.

post_particle
: An optional function that, if non-NULL, is applied after each completed particle. The function must accept three arguments: 1) an object of SimInf_pmcmc with the current state of the fitting process, 2) an object SimInf_pfilter with the last particle and one filtered trajectory attached, and 3) an integer with the iteration in the fitting process. This function can be useful to, for example, monitor, save and inspect intermediate results. Note that the second SimInf_pfilter argument, is non-NULL only for the first particle in the chain, and for accepted particles.

verbose
: prints diagnostic messages when TRUE. Default is FALSE. When verbose=TRUE, information is printed every 100 iterations. For pmcmc, it is possible to get information every nth information by specifying verbose=n, for example, verbose=1 or verbose=10.

Value

The updated SimInf_pmcmc object with the chain extended by n_iterations new rows.

See Also

pmcmc for initiating a new PMCMC chain.

C_code *Extract the C code from a* SimInf_model *object*

Description

This function extracts the C source code associated with the model. The code may have been automatically generated by mparse or manually provided by the user during model creation. It is useful for inspecting the implementation, debugging complex propensity functions, or verifying the code used in the simulation.

Usage

```
C_code(model)
```

Arguments

model	The SimInf_model object to extract the C code from.

Value

A character vector where each element corresponds to a line of the C source code stored in the model.

Examples

```
## Extract code from an mparse-generated SIR model.
## Using writeLines formats the character vector output for
## readability.
model <- mparse(
  transitions = c("S -> beta * S * I/(S + I + R) -> I",
                  "I -> gamma * I -> R"),
  compartments = c("S", "I", "R"),
  gdata = c(beta = 0.16, gamma = 0.077),
  u0 = data.frame(S = 99, I = 1, R = 0),
  tspan = 1:10,
  use_enum = TRUE
)

writeLines(C_code(model))
```

distance_matrix *Create a distance matrix between nodes for spatial models*

Description

Calculate the Euclidean distances between all pairs of nodes based on their projected coordinates (x, y). Distances greater than the specified cutoff are excluded from the result (stored as zeros in the sparse matrix).

Usage

```
distance_matrix(x, y, cutoff, min_dist = NULL, na_fail = TRUE)
```

Arguments

x	Numeric vector of projected x coordinates for each node.
y	Numeric vector of projected y coordinates for each node.
cutoff	Numeric scalar. The maximum distance to include in the matrix. Pairs of nodes farther apart than this value are excluded from the sparse structure (stored as zeros).
min_dist	Numeric scalar. The value to use for the distance between two nodes if their coordinates are identical (distance = 0). This prevents division by zero errors in downstream calculations (e.g., inverse distance weighting). If NULL (default) and identical coordinates are found, an error is raised.
na_fail	Logical. If TRUE (default), missing values (NA) in x or y will raise an error. If FALSE, distances involving missing coordinates are set to zero.

Details

The result is a symmetric sparse matrix (dgCMatrix) where the element d[i, j] contains the distance between node i and node j if it is less than or equal to cutoff, and 0 otherwise.

Value

A symmetric sparse matrix of class dgCMatrix. Non-zero entries represent distances $\leq$ cutoff; entries outside the cutoff are implicitly zero.

Examples

```
## Generate a 10 x 10 grid of nodes separated by 100m.
nodes <- expand.grid(
  x = seq(from = 0, to = 900, by = 100),
  y = seq(from = 0, to = 900, by = 100)
)
plot(nodes, main = "Node Grid", asp = 1)

## Calculate distances with a 300m cutoff.
```

```
## Only neighbors within 300m will have non-zero entries.
d <- distance_matrix(
  x = nodes$x,
  y = nodes$y,
  cutoff = 300
)

## Inspect the sparse matrix structure.
## Note: The matrix is symmetric and the diagonal is zero.
d[1:10, 1:10]

## Count the number of neighbors for the first node.
sum(d[1, ] > 0)
```

`edge_properties_to_matrix` *Convert an edge list with properties to a matrix*

Description

A utility function to convert a data frame of edge properties (e.g., movement rates, counts) into the specific matrix format required for `ldata` in spatial models. This format allows the C code to efficiently iterate over all incoming edges for a specific node.

Usage

```
edge_properties_to_matrix(edges, n_nodes)
```

Arguments

`edges`	A `data.frame` with properties assigned for each edge (`from` –> `to`). Must contain columns `from` and `to` (integer indices, 1-based) and any number of numeric property columns.
`n_nodes`	The total number of nodes in the model. The resulting matrix will have `n_nodes` columns.

Details

The function converts the `edges` data frame into a numeric matrix where:

- Each **column** corresponds to a `to` node.
- Each column contains a **single sequence** of data blocks, one for each incoming edge to that node.
- Each block starts with the **zero-based index** of the `from` node.
- The subsequent rows in the block contain the property values (e.g., rate, count).
- The sequence for a node ends with a **stop marker** (`-1`) in the **first row of the block** (the position where the `from` index is stored). This marker appears exactly once per column, immediately after the last incoming edge.

- Unused cells in the matrix (below the stop marker) are filled with NaN.

The edges data frame must contain columns from and to with valid node indices (1-based). All edges pointing to the same to node must be unique (no duplicate from -> to pairs).

Value

A numeric matrix with dimensions determined by the maximum number of incoming edges for any node. Columns correspond to to nodes. Entries are NaN where no data exists.

Examples

```
## Define edge properties: from, to, rate, and count.
edges <- data.frame(
  from  = c(2, 3, 4, 1, 4, 5, 1, 3, 1, 3),
  to    = c(1, 1, 1, 2, 3, 3, 4, 4, 5, 5),
  rate  = c(0.2, 0.01, 0.79, 1, 0.2, 0.05, 0.2, 0.8, 0.2, 0.8),
  count = c(5, 5, 5, 50, 10, 10, 5, 5, 5, 5)
)

## Convert to matrix for 6 nodes.
mat <- edge_properties_to_matrix(edges, 6)
print(mat)

## Interpretation of Column 1 (to node 1): The column contains a
## single sequence of 3 blocks (edges from # 2, 3, and 4).
## Row 1: Index of 'from' node 2 (2-1 = 1).
## Row 2: rate=0.2
## Row 3: count=5
## Row 4: Index of 'from' node 3 (3-1 = 2).
## Row 5: rate=0.01
## Row 6: count=5
## Row 7: Index of 'from' node 4 (4-1 = 3).
## Row 8: rate=0.79
## Row 9: count=5
## Row 10: Stop marker (-1) indicating the end of the list for node 1.
```

---

events *Extract the scheduled events from a* SimInf_model *object*

Description

Retrieve the SimInf_events object containing the schedule of discrete events (e.g., births, deaths, movements) associated with a SimInf_model. This object holds the timing, location, and type of each event, as well as the matrices defining how events affect the model state.

Usage

```
events(object, ...)

## S4 method for signature 'SimInf_model'
events(object, ...)
```

Arguments

object	A SimInf_model object.
...	Additional arguments (currently ignored).

Value

A SimInf_events object containing the event schedule and associated matrices (E and N).

See Also

SimInf_events for details on the structure of the returned event object (slots E (select matrix), N (shift matrix), event, etc.). events_SIR, events_SEIR, events_SISe3 for examples of pre-defined event datasets. mparse for defining custom models with event schedules. run for executing the simulation with the scheduled events. Vignette "Scheduled events" for a comprehensive tutorial on defining event data, using the **select** (E) and **shift** (N) matrices, and simulating complex demographic and movement processes.

Examples

```
## Create an SIR model with scheduled events.
model <- SIR(
  u0     = u0_SIR(),
  tspan  = 1:(4 * 365),
  events = events_SIR(),
  beta   = 0.16,
  gamma  = 0.077
)

## Extract the events and display a summary.
ev <- events(model)
summary(ev)

## Plot the event schedule over time.
plot(ev)
```

---

events_SEIR *Example event data for the SEIR model with cattle herds*

Description

Dataset containing 466,692 scheduled events for a population of 1,600 cattle herds over 1,460 days (4 years). Demonstrates how demographic and movement events affect SEIR dynamics in a cattle disease context.

Usage

```
events_SEIR()
```

Details

The event data contains three types of scheduled events that affect cattle herds (nodes):

Exit Deaths or removal of cattle from a herd (n = 182,535). These events decrease the population and affect all disease compartments proportionally.

Enter Births or introduction of cattle to a herd (n = 182,685). These events add susceptible cattle to herds, increasing overall herd size.

External transfer Movement of cattle between herds (n = 101,472). These events transfer cattle from one herd to another, potentially facilitating between-herd disease transmission.

The `select` column in the returned data frame is mapped to the columns of the internal select matrix:

- `select = 1` corresponds to **Enter** events, targeting the Susceptible (S) compartment.
- `select = 2` corresponds to **Exit** and **External Transfer** events, targeting all compartments.

Events are distributed across all 1,600 herds over the 4-year period. These are synthetic data generated to illustrate how to incorporate scheduled events (such as births, deaths, and movements) into a compartment model in the SimInf framework.

Value

A `data.frame` with columns:

event Event type: "exit", "enter", or "extTrans".

time Day when event occurs (1-1460).

node Affected herd identifier (1-1600).

dest Destination herd for external transfer events.

n Number of cattle affected.

proportion 0. Not used in this example.

select Model compartment to affect (see `SimInf_events`).

shift 0. Not used in this example.

See Also

`u0_SEIR` for the corresponding initial cattle population, `SEIR` for creating SEIR models with these events, and `SimInf_events` for event structure details

Examples

```
## For reproducibility, call the set.seed() function and specify the
## number of threads to use. To use all available threads, remove the
## set_num_threads() call.
set.seed(123)
set_num_threads(1)

## Create a 'SEIR' model with 1600 cattle herds (nodes) and initialize
## it to run over 4*365 days. Add ten exposed animals to the first
## herd. Define 'tspan' to record the state of the system at weekly
## time-points. Load scheduled events for the population of nodes with
## births, deaths and between-node movements of individuals.
u0 <- u0_SEIR()
u0$E[1] <- 10
model <- SEIR(
    u0      = u0,
    tspan   = seq(from = 1, to = 4*365, by = 7),
    events  = events_SEIR(),
    beta    = 0.16,
    epsilon = 0.25,
    gamma   = 0.01
)

## Display the number of cattle affected by each event type per day.
plot(events(model))

## Run the model to generate a single stochastic trajectory.
result <- run(model)

## Plot the median and interquartile range of the number of
## susceptible, exposed, infected and recovered individuals.
plot(result)

## Plot the trajectory for the first herd.
plot(result, index = 1)

## Summarize the trajectory. The summary includes the number of events
## by event type.
summary(result)
```

`events_SIR` *Example event data for the SIR model with cattle herds*

Description

Dataset containing 466,692 scheduled events for a population of 1,600 cattle herds over 1,460 days (4 years). Demonstrates how demographic and movement events affect SIR dynamics in a cattle disease context.

Usage

```
events_SIR()
```

Details

The event data contains three types of scheduled events that affect cattle herds (nodes):

Exit Deaths or removal of cattle from a herd (n = 182,535). These events decrease the population and remove cattle from the disease system.

Enter Births or introduction of cattle to a herd (n = 182,685). These events add susceptible cattle to herds, increasing potential targets for infection.

External transfer Movement of cattle between herds (n = 101,472). These events transfer cattle from one herd to another, potentially spreading disease across the herd network.

The `select` column in the returned data frame is mapped to the columns of the internal select matrix (`select_matrix_SIR`):

- `select` = 1 corresponds to **Enter** events, targeting the Susceptible (S) compartment.
- `select` = 4 corresponds to **Exit** and **External Transfer** events, targeting all compartments (S, I, and R).

Events are distributed across all 1,600 herds over the 4-year period. These are synthetic data generated to illustrate how to incorporate scheduled events (such as births, deaths, and movements) into a compartment model in the SimInf framework.

Value

A `data.frame` with columns:

event Event type: "exit", "enter", or "extTrans".

time Day when event occurs (1-1460).

node Affected herd identifier (1-1600).

dest Destination herd for external transfer events.

n Number of cattle affected.

proportion 0. Not used in this example.

select Model compartment to affect (see `SimInf_events`).

shift 0. Not used in this example.

See Also

`u0_SIR` for the corresponding initial cattle population, `SIR` for creating SIR models with these events, and `SimInf_events` for event structure details

Examples

```
## For reproducibility, call the set.seed() function and specify the
## number of threads to use. To use all available threads, remove the
## set_num_threads() call.
set.seed(123)
set_num_threads(1)

## Create an 'SIR' model with 1600 cattle herds (nodes) and initialize
## it to run over 4*365 days. Add one infected animal to the first
## herd to seed the outbreak. Define 'tspan' to record the state of
## the system at daily time-points. Load scheduled events for the
## population of nodes with births, deaths and between-node movements
## of individuals.
u0 <- u0_SIR()
u0$I[1] <- 1
model <- SIR(
    u0     = u0,
    tspan  = seq(from = 1, to = 4*365, by = 1),
    events = events_SIR(),
    beta   = 0.16,
    gamma  = 0.01
)

## Display the number of cattle affected by each event type per day.
plot(events(model))

## Run the model to generate a single stochastic trajectory.
result <- run(model)

## Plot the median and interquartile range of the number of
## susceptible, infected and recovered individuals.
plot(result)

## Plot the trajectory for the first herd.
plot(result, index = 1)

## Summarize the trajectory. The summary includes the number of events
## by event type.
summary(result)
```

---

events_SIS *Example event data for the SIS model with cattle herds*

Description

Dataset containing 466,692 scheduled events for a population of 1,600 cattle herds over 1,460 days (4 years). Demonstrates how demographic and movement events affect SIS dynamics in a cattle disease context.

Usage

```
events_SIS()
```

Details

The event data contains three types of scheduled events that affect cattle herds (nodes):

Exit Deaths or removal of cattle from a herd (n = 182,535). These events decrease the population in both susceptible and infected compartments.

Enter Births or introduction of cattle to a herd (n = 182,685). These events add susceptible cattle to herds.

External transfer Movement of cattle between herds (n = 101,472). These events transfer cattle from one herd to another, potentially spreading disease across the herd network. Either susceptible or infected animals may be transferred.

The `select` column in the returned data frame is mapped to the columns of the internal select matrix:

- `select` = 1 corresponds to **Enter** events, targeting the Susceptible (S) compartment.
- `select` = 2 corresponds to **Exit** and **External Transfer** events, targeting all compartments (S and I).

Events are distributed across all 1,600 herds over the 4-year period. These are synthetic data generated to illustrate how to incorporate scheduled events (such as births, deaths, and movements) into a compartment model in the SimInf framework.

Value

A `data.frame` with columns:

event Event type: "exit", "enter", or "extTrans".

time Day when event occurs (1-1460).

node Affected herd identifier (1-1600).

dest Destination herd for external transfer events.

n Number of cattle affected.

proportion 0. Not used in this example.

select Model compartment to affect (see `SimInf_events`).

shift 0. Not used in this example.

See Also

`u0_SIS` for the corresponding initial cattle population, `SIS` for creating SIS models with these events, and `SimInf_events` for event structure details

Examples

```
## For reproducibility, call the set.seed() function and specify the
## number of threads to use. To use all available threads, remove the
## set_num_threads() call.
set.seed(123)
set_num_threads(1)

## Create an 'SIS' model with 1600 cattle herds (nodes) and initialize
## it to run over 4*365 days. Add one infected animal to the first
## herd to seed the outbreak. Define 'tspan' to record the state of
## the system at daily time-points. Load scheduled events for the
## population of nodes with births, deaths and between-node movements
## of individuals.
u0 <- u0_SIS()
u0$I[1] <- 1
model <- SIS(
    u0     = u0,
    tspan  = seq(from = 1, to = 4*365, by = 1),
    events = events_SIS(),
    beta   = 0.16,
    gamma  = 0.01
)

## Display the number of cattle affected by each event type per day.
plot(events(model))

## Run the model to generate a single stochastic trajectory.
result <- run(model)

## Plot the median and interquartile range of the number of
## susceptible and infected individuals.
plot(result)

## Plot the trajectory for the first herd.
plot(result, index = 1)

## Summarize the trajectory. The summary includes the number of events
## by event type.
summary(result)
```

`events_SISe3` *Example event data for the SISe3 model with cattle herds*

Description

Dataset containing 783,773 scheduled events for a population of 1,600 cattle herds stratified by age over 1,460 days (4 years). Demonstrates how demographic, movement, and age-transition events affect SISe3 dynamics in a cattle disease context.

Usage

```
data(events_SISe3)
```

Format

A `data.frame`

Details

This dataset contains four types of scheduled events that affect cattle herds (nodes) with age structure:

Exit Deaths or removal of cattle from a herd (n = 182,535). These events remove cattle from susceptible or infected compartments across age categories.

Enter Births or introduction of cattle to a herd (n = 182,685). These events add susceptible cattle, typically to the youngest age category.

Internal transfer Age transitions or within-herd movements (n = 317,081). These events move cattle between age categories within a herd, reflecting maturation and changing infection risk with age.

External transfer Movement of cattle between herds (n = 101,472). These events transfer cattle from one herd to another across age categories, potentially introducing infected animals.

The `select` column in the returned data frame is mapped to the columns of the internal select matrix as follows:

- `select` = `1`: Targets **S_1** (Susceptible, age 1).
- `select` = `2`: Targets **S_2** (Susceptible, age 2).
- `select` = `3`: Targets **S_3** (Susceptible, age 3).
- `select` = `4`: Targets **S_1** and **I_1** (Susceptible and Infected, age 1).
- `select` = `5`: Targets **S_2** and **I_2** (Susceptible and Infected, age 2).
- `select` = `6`: Targets **S_3** and **I_3** (Susceptible and Infected, age 3).

The `shift` column is used for **Internal transfer** events to define the destination compartment. It corresponds to the column index in the internal `N` matrix that specifies the transition (e.g., moving from age 1 to age 2).

Events are distributed across all 1,600 herds over the 4-year period. These are synthetic data generated to illustrate how to incorporate scheduled events (including births, deaths, movements, and age transitions) into an age-structured compartment model in the SimInf framework. The higher event count compared to non-age-structured models reflects the addition of internal transfer events required for age category transitions.

The data contains:

event Event type: "exit", "enter", "intTrans", or "extTrans".

time Day when event occurs (1-1460).

node Affected herd identifier (1-1600).

dest Destination herd for external transfer events, else 0.

n Number of cattle affected.

select Model compartment to affect (see SimInf_events).

proportion 0. Not used in this example.

shift Determines how individuals in internal transfer events are shifted to enter another compartment.

See Also

u0_SISe3 for the corresponding initial cattle population with age structure, SISe3 for creating SISe3 models with these events and SimInf_events for event structure details

Examples

```
## For reproducibility, call the set.seed() function and specify the
## number of threads to use. To use all available threads, remove the
## set_num_threads() call.
set.seed(123)
set_num_threads(1)

## Create an 'SISe3' model with 1600 cattle herds (nodes) stratified
## by age, initialize it to run over 4*365 days and record data at
## weekly time-points. Add ten infected animals to age category 1 in
## the first herd to seed the outbreak.  Define 'tspan' to record the
## state of the system at weekly time-points. Load scheduled events
## events for the population of nodes with births, deaths and
## between-node movements of individuals.
u0 <- u0_SISe3
u0$I_1[1] <- 10
model <- SISe3(
    u0 = u0,
    tspan = seq(from = 1, to = 4*365, by = 7),
    events = events_SISe3,
    phi = rep(0, nrow(u0)),
    upsilon_1 = 1.8e-2,
    upsilon_2 = 1.8e-2,
    upsilon_3 = 1.8e-2,
    gamma_1 = 0.1,
    gamma_2 = 0.1,
    gamma_3 = 0.1,
    alpha = 1,
    beta_t1 = 1.0e-1,
    beta_t2 = 1.0e-1,
    beta_t3 = 1.25e-1,
    beta_t4 = 1.25e-1,
    end_t1 = 91,
    end_t2 = 182,
    end_t3 = 273,
    end_t4 = 365,
    epsilon = 0
)
```

```
## Display the number of cattle affected by each event type per day.
plot(events(model))

## Run the model to generate a single stochastic trajectory.
result <- run(model)

## Plot the median and interquartile range of the number of
## susceptible and infected individuals.
plot(result)

## Plot the proportion of nodes with at least one infected individual.
plot(result, I_1 + I_2 + I_3 ~ ., level = 2, type = "l")

## Plot the trajectory for the first herd.
plot(result, index = 1)

## Summarize the trajectory. The summary includes the number of events
## by event type.
summary(result)
```

---

gdata *Extract global data from a* SimInf_model *object*

Description

The global data is a numeric vector that is common to all nodes. The global data vector is passed as an argument to the transition rate functions and the post time step function.

Usage

```
gdata(model)

## S4 method for signature 'SimInf_model'
gdata(model)
```

Arguments

model	The model to get global data from.

Value

a numeric vector

See Also

ldata for retrieving local data (node-specific), and SimInf_model for the class definition and overview of model structure.

Examples

```
## Create an SIR model
model <- SIR(
  u0 = data.frame(S = 99, I = 1, R = 0),
  tspan = 1:5,
  beta = 0.16,
  gamma = 0.077
)

## Set 'beta' to a new value
gdata(model, "beta") <- 2

## Extract the global data vector that is common to all nodes
gdata(model)
```

---

gdata<- *Set a global data parameter for a* SimInf_model *object*

Description

The global data is a numeric vector that is common to all nodes. The global data vector is passed as an argument to the transition rate functions and the post time step function.

Usage

```
gdata(model, parameter) <- value

## S4 replacement method for signature 'SimInf_model'
gdata(model, parameter) <- value
```

Arguments

model	The model to set a global model parameter for.
parameter	The name of the parameter to set.
value	A numeric value.

Value

a SimInf_model object

See Also

ldata for retrieving local data (node-specific), and SimInf_model for the class definition and overview of model structure.

Examples

```
## Create an SIR model
model <- SIR(
  u0 = data.frame(S = 99, I = 1, R = 0),
  tspan = 1:5,
  beta = 0.16,
  gamma = 0.077
)

## Set 'beta' to a new value
gdata(model, "beta") <- 2

## Extract the global data vector that is common to all nodes
gdata(model)
```

`get_individuals` *Extract individuals from* `SimInf_individual_events`

Description

Lookup individuals, in which node they are located and their age at a specified time-point.

Usage

```
get_individuals(x, time = NULL)

## S4 method for signature 'SimInf_individual_events'
get_individuals(x, time = NULL)
```

Arguments

`x`	an individual events object of class `SimInf_individual_events`.
`time`	the time-point for the lookup of individuals. Default is `NULL` which means to extract the individuals at the minimum time-point in the events object `x`.

Value

a `data.frame` with the columns `id`, `node`, and `age`.

indegree *Determine in-degree for each node in a model*

Description

Calculate the in-degree of each node based on **external transfer** events (`"extTrans"`) in the model's schedules events.

Usage

```
indegree(model)
```

Arguments

`model`	A `SimInf_model` object containing the event schedule.

Details

The in-degree is defined as the number of **unique source nodes** that have sent individuals to the target node at least once. This metric measures the connectivity of the network, indicating how many different neighbors directly supply individuals to a specific node.

Value

An integer vector where each element corresponds to a node, containing the count of unique source nodes sending individuals to it.

See Also

`outdegree` for calculating the number of unique destination nodes a node sends individuals to. `events_SIR` for example event data used in network analysis. `SimInf_events` for details on the structure of scheduled events.

Examples

```
## Create an 'SIR' model with example data.
model <- SIR(
  u0 = u0_SIR(),
  tspan = 1:1460,
  events = events_SIR(),
  beta = 0.16,
  gamma = 0.077
)

## Calculate the in-degree for each node.
deg <- indegree(model)

## View the in-degree for the first 6 nodes.
head(deg)
```

```
## Plot the distribution of in-degrees across all nodes.  This
## shows how many source nodes typically send individuals to a given
## node.
hist(
  deg,
  main = "Distribution of In-Degree (Unique Sources)",
  xlab = "Number of Unique Source Nodes"
)
```

`individual_events` *Individual events*

Description

In many countries, individual-based livestock data are collected to enable contact tracing during disease outbreaks. However, the livestock databases are not always structured in such a way that relevant information for disease spread simulations is easily retrieved. The aim of this function is to facilitate cleaning livestock event data and prepare it for usage in SimInf.

Usage

```
individual_events(events)
```

Arguments

`events`	a `data.frame` with the columns `id`, `event`, `time`, `node`, and `dest` to define the events, see `details`.

Details

The argument `events` in `individual_events` must be a `data.frame` with the following columns:

- **id:** an integer or character identifier of the individual.
- **event:** four event types are supported: *exit*, *enter*, *internal transfer*, and *external transfer*. When assigning the events, they can either be coded as a numerical value or a character string: *exit;* `0` or `'exit'`, *enter;* `1` or `'enter'`, *internal transfer;* `2` or `'intTrans'`, and *external transfer;* `3` or `'extTrans'`.
- **time:** an integer, character, or date (of class `Date`) for when the event occured. If it's a character it must be able to coerce to `Date`.
- **node:** an integer or character identifier of the source node.
- **dest:** an integer or character identifier of the destination node for movement events, and 'dest' will be replaced with `NA` for non-movement events.

Value

SimInf_individual_events

References

I. R. Ewerl<c3><b6>f, J. Fr<c3><b6>ssling, M. Tr<c3><a5>v<c3><a9>n, S. Gunnarsson, L. Steng<c3><a4>rde, E. Hurri, and S. Widgren. Exploring structural changes in the Swedish cattle population and between-holding movements. *Preventive Veterinary Medicine*, **243**, 106608, 2025. doi:10.1016/j.prevetmed.2025.106608

See Also

node_events.

`lambertW0` *Lambert W0 function*

Description

The principal branch of the Lambert W function, which solves $We^W = x$. Defined for $x >= -1/e$, with output in the range $[-1, \infty)$. The value is calculated using the GNU Scientific Library (GSL).

Usage

```
lambertW0(x)
```

Arguments

x	A numeric vector of values.

Value

A numeric vector of the same length as x containing the principal branch of the Lambert W function evaluated at each element.

References

GNU Scientific Library <https://www.gnu.org/software/gsl/>

Examples

```
## Should equal 1, as 1 * exp(1) = e.
lambertW0(exp(1))

## Should equal 0, as 0 * exp(0) = 0.
lambertW0(0)

## Should equal -1.
lambertW0(-exp(-1))
```

ldata *Extract local data from a node*

Description

The local data is a numeric vector that is specific to a node. The local data vector is passed as an argument to the transition rate functions and the post time step function.

Usage

```
ldata(model, node)

## S4 method for signature 'SimInf_model'
ldata(model, node)
```

Arguments

model	The model to get local data from.
node	index to node to extract local data from.

Value

a numeric vector

See Also

gdata for retrieving global data (common to all nodes), and SimInf_model for the class definition and overview of model structure.

Examples

```
## Create an 'SISe' model with 1600 nodes.
model <- SISe(
  u0 = u0_SIS(),
  tspan = 1:100,
  events = events_SIS(),
  phi = 0,
  upsilon = 1.8e-2,
  gamma = 0.1,
  alpha = 1,
  beta_t1 = 1.0e-1,
  beta_t2 = 1.0e-1,
  beta_t3 = 1.25e-1,
  beta_t4 = 1.25e-1,
  end_t1 = c(91, 101),
  end_t2 = c(182, 185),
  end_t3 = c(273, 275),
  end_t4 = c(365, 360),
  epsilon = 0
```

```
)

## Display local data from the first two nodes.
ldata(model, node = 1)
ldata(model, node = 2)
```

---

length,SimInf_pmcmc-method

Length of the MCMC chain

Description

Get the number of iterations (samples) in the Markov Chain Monte Carlo (MCMC) chain stored in a `SimInf_pmcmc` object.

Usage

```
## S4 method for signature 'SimInf_pmcmc'
length(x)
```

Arguments

`x`	A `SimInf_pmcmc` object containing the MCMC results.

Value

An integer scalar representing the number of rows in the `chain` slot (i.e., the total number of samples in the chain).

---

logLik,SimInf_pfilter-method

Extract Log-Likelihood

Description

Extract the estimated log-likelihood from a `SimInf_pfilter` object.

Usage

```
## S4 method for signature 'SimInf_pfilter'
logLik(object)
```

Arguments

`object`	The `SimInf_pfilter` object.

Value

The estimated log-likelihood, returned as a numeric scalar.

See Also

pfilter for running a particle filter and creating SimInf_pfilter objects, SimInf_pfilter for the class definition.

---

mparse *Model parser to define new models for* SimInf

Description

Describe your model using a simple string syntax in R. mparse parses this description, generates model-specific C code, and returns a SimInf_model object ready for simulation.

Usage

```
mparse(
  transitions = NULL,
  compartments = NULL,
  ldata = NULL,
  gdata = NULL,
  u0 = NULL,
  v0 = NULL,
  tspan = NULL,
  events = NULL,
  E = NULL,
  N = NULL,
  pts_fun = NULL,
  use_enum = FALSE,
  pre_code = NULL,
  replicates = NULL
)
```

Arguments

transitions — Character vector defining the state transitions. Each element follows the format "Source -> Propensity -> Target".

- **Syntax**: "X -> rate_expr -> Y" moves individuals from compartment X to Y with rate rate_expr.
- **Empty Set**: Use @ for the empty set (e.g., "I -> mu*I -> @" for death, or "@ -> lambda -> S" for birth).
- **Variables**: Define intermediate variables using <-. Example: "N <- S + I + R". Variables can be used in propensity expressions. Order does not matter.

- **Types**: By default, variables are `double`. Use `(int)` to force integer type (e.g., `"(int)N <- S+I+R"`).

Example: `c("S -> beta*S*I/N -> I", "I -> gamma*I -> R","N <- S+I+R")`.

`compartments` Character vector of compartment names (e.g., `c("S", "I", "R")`).

`ldata` Optional local data (node-specific parameters). Can be:

- A `data.frame` with one row per node (columns = parameters).
- A numeric matrix where columns are nodes and row names are parameters.
- A named vector (for single-node models).

`gdata` Optional global data (common to all nodes). Can be:

- A named numeric vector (names identify parameters).
- A one-row `data.frame`.

Unnamed parameters in a vector must be referenced by index in the transitions.

`u0` Initial state vector. Can be a `data.frame`, matrix, or named vector. See `SimInf_model` for details.

`v0` optional data with the initial continuous state in each node. `v0` can be specified as a `data.frame` with one row per node, as a numeric matrix where column `v0[,j]` contains the initial state vector for the node `j`, or as a named vector when the model only contains one node. If `v0` is specified as a `data.frame`, each column is one parameter. If `v0` is specified as a matrix, the row names identify the parameters. If `v0` is specified as a named vector, the names identify the parameters. The 'v' vector is passed as an argument to the transition rate functions and the post time step function. The continuous state can be updated in the post time step function.

`tspan` A vector (length >= 1) of increasing time points where the state of each node is to be returned. Can be either an `integer` or a `Date` vector.

- If `integer`: Represents the specific time points (e.g., days, hours) at which to record the state.
- If `Date`: Coerced to a numeric vector representing the **day of the year** (1–366) relative to the first date in the vector. The original `Date` objects are preserved as names for the numeric vector, facilitating time-series plotting.

`events` A `data.frame` with the scheduled events. Default is `NULL` i.e. no scheduled events in the model.

`E` Optional select matrix for events. Can be a `matrix` or `data.frame`. Defines which compartments are affected by events and their sampling weights. See `SimInf_events` for detailed logic on `data.frame` columns (`compartment`, `select`, `value`). Default is `NULL` (no events).

`N` Optional shift matrix for events. Can be a `matrix` or `data.frame`. Defines how individuals are moved between compartments during events. See `SimInf_events` for detailed logic on `data.frame` columns (`compartment`, `shift`, `value`). Default is `NULL` (no events).

`pts_fun` Optional character vector with C code for the post-time-step function. Should contain only the function body (code between curly brackets).

use_enum — Logical. If TRUE, generates C enumeration constants for parameters found in the u, v, gdata, and ldata vectors to facilitate manual C code modification or use in pts_fun. The name of each enumeration constant is the upper-case version of the corresponding parameter name (e.g., beta becomes BETA). Additionally, the following constants are automatically generated:

- N_COMPARTMENTS_U: The number of discrete compartments.
- N_COMPARTMENTS_V: The number of continuous state variables.

These constants facilitate indexing of the u and v vectors in C code. Note that N_COMPARTMENTS_U and N_COMPARTMENTS_V cannot be used as compartment or variable names. Default is FALSE (parameters accessed by integer indices).

pre_code — Optional character vector with C code to be inserted into the generated model code, after the enumeration constants and before the transition rate functions. This allows users to define helper functions or include statements that can be referenced from transition propensity expressions. Include statements, if needed, should be placed at the beginning of the vector. Default is NULL, i.e., no additional code is inserted.

replicates — Number of model replicates to simulate (default NULL, treated as 1L). When replicates > 1L, each replicate is simulated independently using its own initial state (from u0), but shares the same parameters (gdata, ldata), scheduled events, and structure (transitions, compartments).

The u0 argument must contain initial states for all replicates. Each replicate requires n nodes, where n is the number of nodes in the model. Thus, u0 must have replicates * n nodes total. Nodes are grouped by replicate: the first n nodes belong to replicate 1, the next n to replicate 2, and so on. This allows different starting conditions per replicate if desired.

ldata remains unchanged: its data per node is shared across all replicates. Scheduled events are also shared—the same event schedule applies to each replicate.

Use this when you need multiple independent stochastic trajectories from the same model in a single simulation run. For identical starting conditions across replicates, simply repeat the same node pattern in u0.

Value

a SimInf_model object

See Also

SimInf_model for the class definition of the returned model object. SIR, SEIR, SIS, SISe for high-level model constructors that use predefined structures. Vignette "Getting started with mparse" for a comprehensive tutorial on defining custom models, including syntax, variables, and event handling. package_skeleton for creating an installable R package from a mparse model.

Examples

```
## Create an SIR model with a defined population size variable.
model <- mparse(
  transitions = c(
```

```
    "S -> beta * S * I / N -> I",
    "I -> gamma * I -> R",
    "N <- S + I + R"
  ),
  compartments = c("S", "I", "R"),
  gdata = c(beta = 0.16, gamma = 0.077),
  u0 = data.frame(S = 100, I = 1, R = 0),
  tspan = 1:100
)

## Running a model defined with 'mparse' compiles model-specific C
## code and requires a C compiler on the system. See the vignette
## "Getting started with mparse" for a complete walkthrough
## including running and plotting.
```

---

nodes *Example data with spatial distribution of nodes*

Description

Example data containing spatial coordinates for 1600 nodes, used to initialize spatially distributed population models and visualize simulation results.

Usage

```
data(nodes)
```

Format

A `data.frame` with 1600 rows and 2 columns:

x Numeric vector of x-coordinates.

y Numeric vector of y-coordinates.

Examples

```
## For reproducibility, specify the number of threads.
set_num_threads(1)

## Create an 'SIR' model with 1600 nodes and initialize
## it to run over 4*365 days. Add one infected individual
## to the first node.
u0 <- u0_SIR()
u0$I[1] <- 1
tspan <- seq(from = 1, to = 4*365, by = 1)

model <- SIR(
    u0     = u0,
    tspan  = tspan,
    events = events_SIR(),
```

```
    beta   = 0.16,
    gamma  = 0.077
)

## Run the model to generate a single stochastic trajectory.
## For reproducibility, specify the seed.
result <- run(model, seed = 22)

## Determine nodes with one or more infected individuals in the
## trajectory. Extract the 'I' compartment and check for any
## infected individuals in each node.
infected <- colSums(trajectory(result, ~ I, format = "matrix")) > 0

## Display infected nodes in 'blue' and non-infected nodes in 'yellow'.
data("nodes", package = "SimInf")
plot(
    y ~ x,
    nodes,
    col = ifelse(infected, "blue", "yellow"),
    pch = 20,
    asp = 1
)
```

`node_events` *Transform individual events to node events for a model*

Description

In many countries, individual-based livestock data are collected to enable contact tracing during disease outbreaks. However, the livestock databases are not always structured in such a way that relevant information for disease spread simulations is easily retrieved. The aim of this function is to facilitate cleaning livestock event data and prepare it for usage in SimInf.

Usage

```
node_events(x, time = NULL, target = NULL, age = NULL)

## S4 method for signature 'SimInf_individual_events'
node_events(x, time = NULL, target = NULL, age = NULL)
```

Arguments

`x`	an individual events object of class `SimInf_individual_events`.
`time`	All events that occur after 'time' are included. Default is `NULL` which means to extract the events after the minimum time-point in the `SimInf_individual_events` object.
`target`	The SimInf model ('SEIR', 'SIR', 'SIS', 'SISe3', 'SISe3_sp', 'SISe', or 'SISe_sp') to target the events and u0 for. The default, `NULL`, creates events but they might have to be post-processed to fit the specific use case.
`age`	Integer vector with break points in days for the ageing events.

Details

The individual-based events will be aggregated on node-level. The `select` value is determined by the event type and age category. If there is only one age category, i.e., `age=NULL`, then `select=1` for the enter events, and `select=2` for all other events. If there are two age categories, then `select=1` for the enter events in the first age category, and `select=2` for the enter events in the second age category. Similarly, `select=3` for all other events in the first age category, and `select=4` for all other events in the first second category. With three age categories, it works similarly with `select=1,2,3` for the enter events in each age category, respectively. And `select=4,5,6` for all other events.

Value

a `data.frame` with the columns `event`, `time`, `node`, `dest`, `n`, `proportion`, `select`, and `shift`.

See Also

`individual_events`.

---

n_compartments *Determine the number of compartments in a model*

Description

Extract the number of compartments from a `SimInf_model` object. Compartments represent the distinct states an individual can occupy within a node (e.g., Susceptible, Infected, Recovered). This count is equivalent to the number of columns in the initial state vector `u0`.

Usage

```
n_compartments(model)

## S4 method for signature 'SimInf_model'
n_compartments(model)
```

Arguments

`model`	A `SimInf_model` object.

Value

An integer scalar representing the total number of compartments in the model.

Examples

```
## Create an 'SIR' model with 3 compartments (S, I, R).
u0 <- data.frame(
  S = rep(99, 100),
  I = rep(1, 100),
  R = rep(0, 100)
)

model <- SIR(
  u0 = u0,
  tspan = 1:10,
  beta = 0.16,
  gamma = 0.077
)

## Get the number of compartments.
n_compartments(model)
```

---

n_generations *Determine the number of generations in an ABC analysis*

Description

Extract the number of generations performed in an Approximate Bayesian Computation (ABC) analysis from a SimInf_abc object. Each generation represents a sequential round of the algorithm, involving the simulation of particles, acceptance/rejection based on the distance threshold, and parameter perturbation for the next round.

Usage

```
n_generations(object)

## S4 method for signature 'SimInf_abc'
n_generations(object)
```

Arguments

object	A SimInf_abc object containing the results of an ABC analysis.

Value

An integer scalar representing the total number of generations executed in the analysis.

n_nodes *Determine the number of nodes in a model*

Description

Extract the number of nodes from a SimInf_model object. A node represents a distinct subpopulation in the model, but its definition is determined by the modeller. For example, a node can represent a cattle herd, a pen within a herd, or even a single individual, depending on the research question and the scale of the study. This count is equivalent to the number of rows in the initial state vector u0.

Usage

```
n_nodes(model)

## S4 method for signature 'SimInf_model'
n_nodes(model)

## S4 method for signature 'SimInf_pfilter'
n_nodes(model)

## S4 method for signature 'SimInf_pmcmc'
n_nodes(model)
```

Arguments

model	A SimInf_model object.

Value

An integer scalar representing the total number of nodes in the model.

Examples

```
## Create an 'SIR' model with 100 nodes.
u0 <- data.frame(
  S = rep(99, 100),
  I = rep(1, 100),
  R = rep(0, 100)
)

model <- SIR(
  u0 = u0,
  tspan = 1:10,
  beta = 0.16,
  gamma = 0.077
)

## Get the number of nodes.
```

```
n_nodes(model)
```

`n_replicates` *Determine the number of replicates in a model*

Description

Extract the number of replicates from a `SimInf_model` object. Replicates are independent copies of the model state used by the **particle filter** (`pfilter`). This value is set **internally** by `pfilter` and is not specified when creating a standard model. If the model has not been processed by a filter, the value will be 1.

Usage

```
n_replicates(model)

## S4 method for signature 'SimInf_model'
n_replicates(model)
```

Arguments

`model`	A `SimInf_model` object.

Value

An integer scalar representing the number of replicates in the model.

Examples

```
## Create a standard 'SIR' model.
u0 <- data.frame(
  S = rep(99, 100),
  I = rep(1, 100),
  R = rep(0, 100)
)

model <- SIR(
  u0 = u0,
  tspan = 1:10,
  beta = 0.16,
  gamma = 0.077
)

## Get the number of replicates (default is 1 for standard
## models).
n_replicates(model)
```

outdegree *Determine out-degree for each node in a model*

Description

Calculate the out-degree of each node based on **external transfer** events (`"extTrans"`) in the model's scheduled events.

Usage

```
outdegree(model)
```

Arguments

`model`	A `SimInf_model` object containing the scheduled events.

Details

The out-degree is defined as the number of **unique destination nodes** that receive individuals from the source node at least once. This metric measures the connectivity of the network, indicating how many different neighbors a specific node directly sends individuals to.

Value

An integer vector where each element corresponds to a node, containing the count of unique destination nodes receiving individuals from it.

See Also

`indegree` for calculating the number of unique source nodes that send individuals to a node. `events_SIR` for example event data used in network analysis. `SimInf_events` for details on the structure of scheduled events.

Examples

```
## Create an 'SIR' model with example data.
model <- SIR(
  u0 = u0_SIR(),
  tspan = 1:1460,
  events = events_SIR(),
  beta = 0.16,
  gamma = 0.077
)

## Calculate the out-degree for each node.
deg <- outdegree(model)

## View the out-degree for the first 6 nodes.
head(deg)
```

```
## Plot the distribution of out-degrees across all nodes.
## This shows how many destination nodes typically receive
## individuals from a given node.
hist(
  deg,
  main = "Distribution of Out-Degree (Unique Destinations)",
  xlab = "Number of Unique Destination Nodes"
)
```

---

package_skeleton *Create a package skeleton from a* SimInf_model

---

Description

Generate a source package directory from a SimInf_model object, allowing the model to be installed as a standalone add-on R package. This is useful for distributing custom models or for improving startup performance by compiling the C code once during package installation rather than on the first call to run().

Usage

```
package_skeleton(
  model,
  name = NULL,
  path = ".",
  author = NULL,
  email = NULL,
  maintainer = NULL,
  license = "GPL-3"
)
```

Arguments

model	A SimInf_model object to create the package skeleton from.
name	Character string with the package name. It must contain only ASCII letters, numbers, and dots; have at least two characters; start with a letter; and not end in a dot. The package name is also used for the class name of the model and the directory name of the package.
path	Path to put the package directory in. Default is "." (the current directory).
author	Author of the package.
email	Email address of the package maintainer.
maintainer	Maintainer of the package.
license	License of the package. Default is "GPL-3".

Details

The typical workflow is:

1. Define the model using `mparse` or `SimInf_model`.
2. Call `package_skeleton` to generate the package
3. Install the package using `install.packages` or `R CMD INSTALL`.

Value

`invisible(NULL)`.

References

Read the *Writing R Extensions* manual for more details on building R packages.

See Also

`mparse` for creating a `SimInf_model` object from a model specification. `INSTALL` and `install.packages` for installing the generated package.

---

pairs,SimInf_model-method

Scatterplot matrix of number of individuals in each compartment

---

Description

Produce a matrix of scatterplots showing the relationship between the number of individuals in different model compartments. The `ij`th panel displays the counts of compartment `j` plotted against compartment `i`.

Usage

```
## S4 method for signature 'SimInf_model'
pairs(x, compartments = NULL, index = NULL, ...)
```

Arguments

`x`	The `SimInf_model` object containing the trajectory data.
`compartments`	Specify the names of the compartments to include. Can be a character vector (e.g., `c("S", "I","R")`) or a formula (e.g., `~S + I + R`). If `NULL` (default), all compartments in the model are included.
`index`	Indices of the nodes to include in the plot. If `NULL` (default), data from all nodes are pooled together. If a vector (e.g., `1:2`), only data from those specific nodes are used.
`...`	Additional graphical arguments passed to `pairs()`.

Details

Data is aggregated across all time points in `tspan` and the selected nodes (specified by `index`). This allows for visualizing the correlation structure between compartments throughout the simulation.

Examples

```
## For reproducibility, specify the number of threads.
set_num_threads(1)

## Create an 'SIR' model with 10 nodes.
model <- SIR(
  u0 = data.frame(
    S = rep(99, 10),
    I = rep(1, 10),
    R = rep(0, 10)
  ),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Run the model with a fixed seed for reproducibility.
result <- run(model, seed = 22)

## Create a scatterplot matrix for all compartments across all
## nodes.
pairs(result)

## Create a scatterplot matrix for the S and I compartments,
## using only data from nodes 1 and 2.
pairs(result, compartments = c("S", "I"), index = 1:2)
```

`pfilter` *Bootstrap particle filter*

Description

The bootstrap filtering algorithm. Systematic resampling is performed at each observation.

Usage

```
pfilter(model, obs_process, data, n_particles, init_model = NULL)

## S4 method for signature 'SimInf_model'
pfilter(model, obs_process, data, n_particles, init_model = NULL)
```

Arguments

model	The SimInf_model object to simulate data from.
obs_process	Specification of the stochastic observation process. The obs_process can be specified as a formula if the model contains only one node and there is only one data point for each time in data. The left hand side of the formula must match a column name in the data data.frame and the right hand side of the formula is a character specifying the distribution of the observation process, for example, Iobs ~ poisson(I). The following distributions are supported: x ~ binomial(size, prob), x ~ poisson(rate) and x ~ uniform(min, max). The observation process can also be a function to evaluate the probability density of the observations given the simulated states. The first argument passed to the obs_process function is the result from a run of the model and it contains one trajectory with simulated data for a time-point, where the trajectory contains n_particles replicates, see trajectory,SimInf_model-method. The second argument to the obs_process function is a data.frame containing the rows for the specific time-point that the function is called for. Note that the function must return the log of the density.
data	A data.frame holding the time series data.
n_particles	An integer with the number of particles (> 1) to use at each timestep.
init_model	An optional function that, if non-NULL, is applied before running each proposal. The function must accept one argument of type SimInf_model with the current model of the fitting process. This function can be useful to specify the initial state of u0 or v0 of the model before running a trajectory with proposed parameters.

Value

A SimInf_pfilter object.

References

N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation. *Radar and Signal Processing, IEE Proceedings F*, **140**(2) 107–113, 1993. doi:10.1049/ipf2.1993.0015

Examples

```
## Not run:
## Let us consider an SIR model in a closed population with N = 100
## individuals of whom one is initially infectious and the rest are
## susceptible. First, generate one realisation (with a specified
## seed) from the model with known parameters 'beta = 0.16' and
## 'gamma = 0.077'. Then, use 'pfilter' to apply the bootstrap
## particle algorithm on the simulated data.
model <- SIR(u0 = data.frame(S = 99, I = 1, R = 0),
             tspan = seq(1, 100, by = 3),
             beta = 0.16,
             gamma = 0.077)
```

```
## Run the SIR model to generate simulated observed data for the
## number of infected individuals.
set.seed(22)
infected <- trajectory(run(model), "I")[, c("time", "I")]
colnames(infected) <- c("time", "Iobs")

## Use a Poison observation process for the infected individuals, such
## that 'Iobs ~ poison(I + 1e-6)'. A small constant '1e-6' is added to
## prevent numerical errors, since the simulated counts 'I' could be
## zero, which would result in the Poisson rate parameter being zero,
## which violates the conditions of the Poisson distribution. Use 1000
## particles.
pf <- pfilter(model,
              obs_process = Iobs ~ poisson(I + 1e-6),
              data = infected,
              n_particles = 1000)

## Print a brief summary.
pf

## Compare the number infected 'I' in the filtered trajectory with the
## infected 'Iobs' in the observed data.
plot(pf, ~I)
lines(Iobs ~ time, infected, col = "blue", lwd = 2, type = "s")

## End(Not run)
```

`plot,SimInf_abc-method`

Display the ABC posterior distribution

Description

Produce diagnostic plots of the Approximate Bayesian Computation (ABC) posterior distribution stored in a `SimInf_abc` object.

Usage

```
## S4 method for signature 'SimInf_abc'
plot(x, y, ...)
```

Arguments

`x`	The `SimInf_abc` object containing the ABC results.
`y`	The generation number to plot. The default is `NULL`, which displays the **last** generation (the final posterior). Specify an integer to view intermediate generations for convergence diagnostics.
`...`	Additional graphical arguments passed to the underlying plotting functions (e.g., `col` for contour colors, `lwd`).

Details

The function generates a scatterplot matrix of the parameter values for the specified generation:

- **Diagonal panels**: Display a normalized density estimate with a rug plot showing individual samples.
- **Upper triangular panels**: Display scatterplots of the raw parameter samples for each pair of variables.
- **Lower triangular panels**: Display contour lines representing the 2D kernel density estimate of the joint distribution between parameter pairs.

If only a single parameter is selected, a single density plot with a rug is produced.

---

plot,SimInf_events-method

Display the distribution of scheduled events over time

Description

Display the distribution of scheduled events over time

Usage

```
## S4 method for signature 'SimInf_events'
plot(x, frame.plot = FALSE, ...)
```

Arguments

x	The events data to plot.
frame.plot	Draw a frame around each plot. Default is FALSE.
...	Additional arguments affecting the plot

---

plot,SimInf_individual_events-method

Display the distribution of individual events over time

Description

Display the distribution of individual events over time

Usage

```
## S4 method for signature 'SimInf_individual_events'
plot(x, frame.plot = FALSE, ...)
```

Arguments

x	The individual events data to plot.
frame.plot	a logical indicating whether a box should be drawn around the plot.
...	Other graphical parameters that are passed on to the plot function.

---

plot,SimInf_model-method

Display the outcome from a simulated trajectory

Description

Plot the median and quantile range of the counts in all nodes, the counts in specified nodes, or the prevalence of a disease. The function supports formula notation for specifying compartments and prevalence calculations.

Usage

```
## S4 method for signature 'SimInf_model'
plot(
  x,
  y,
  level = 1,
  index = NULL,
  range = 0.5,
  type = "s",
  lwd = 2,
  frame.plot = FALSE,
  legend = TRUE,
  log = "",
  ...
)
```

Arguments

x	The `model` to plot.
y	Character vector or formula with the compartments in the model to include in the plot. Default includes all compartments in the model. Can also be a formula that specifies the compartments that define the cases with a disease or that have a specific characteristic (numerator), and the compartments that define the entire population of interest (denominator). The left-hand-side of the formula defines the cases, and the right-hand-side defines the population, for example, `I~S+I+R` in a 'SIR' model (see 'Examples'). The `.` (dot) is expanded to all compartments, for example, `I~.` is expanded to `I~S+I+R` in a 'SIR' model (see 'Examples').

level	The level at which the prevalence is calculated at each time point in tspan. 1 (population prevalence): calculates the proportion of the individuals (cases) in the population. 2 (node prevalence): calculates the proportion of nodes with at least one case. 3 (within-node prevalence): calculates the proportion of cases within each node. Default is 1.
index	Indices specifying the nodes to include when plotting data. Plot one line for each node. Default (index = NULL) is to extract data from all nodes and plot the median count for the specified compartments.
range	Show the quantile range of the count in each compartment. Default is to show the interquartile range i.e. the middle 50% of the count in transparent color. The median value is shown in the same color. Use range = 0.95 to show the middle 95% of the count. To display individual lines for each node, specify range = FALSE.
type	The type of plot to draw. The default type = "s" draws stair steps. See base plot for other values.
lwd	The line width. Default is 2.
frame.plot	a logical indicating whether a box should be drawn around the plot.
legend	a logical indicating whether a legend for the compartments should be added to the plot. A legend is not drawn for a prevalence plot.
log	A character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.
...	Other graphical parameters (e.g. xlab, ylab, main) that are passed on to the plot function.

Details

For a comprehensive tutorial with detailed explanations and more examples, see the vignette("Post-process data in a trajectory", package = "SimInf").

Examples

```
## For reproducibility, specify the number of threads.
set_num_threads(1)

## Create an 'SIR' model with 100 nodes.
model <- SIR(u0 = data.frame(S = rep(990, 100),
                             I = rep(10, 100),
                             R = rep(0, 100)),
             tspan = 1:100,
             beta = 0.16,
             gamma = 0.077)

## Run the model with a fixed seed for reproducibility.
result <- run(model, seed = 22)

## Plot counts (median and IQR)
plot(result)
```

```
## Plot individual trajectories for specific nodes
plot(result, index = 1:3, range = FALSE)

## Plot prevalence (proportion of infected)
plot(result, I ~ S + I + R)

## Customize labels
plot(result, "I", xlab = "Time", ylab = "Count", main = "Infections")
```

`plot,SimInf_pfilter-method`

Diagnostic plot of a particle filter object

Description

Diagnostic plot of a particle filter object

Usage

```
## S4 method for signature 'SimInf_pfilter'
plot(x, y, ...)
```

Arguments

`x`	The `SimInf_pfilter` object to plot.
`y`	If `y` is `NULL` or missing (default), the filtered trajectory (top) and the effective sample size (bottom) are displayed. If `y` is a character vector or a formula, the plot function for a `SimInf_model` object is called with the filtered trajectory, see `plot` for more details about the specification a plot.
`...`	Other graphical parameters that are passed on to the plot function.

`plot,SimInf_pmcmc-method`

Display the PMCMC posterior distribution

Description

Display the (approximate) posterior distributions obtained from fitting a particle Markov chain Monte Carlo algorithm, or the corresponding trace plots.

Usage

```
## S4 method for signature 'SimInf_pmcmc'
plot(x, y, start = 1, end = NULL, thin = 1, ...)
```

Arguments

x	The SimInf_pmcmc object to plot.
y	The trace of all variables and logPost are plotted when y = "trace" or y = ~trace, else the posterior distributions are plotted. Default is to plot the posterier distributions.
start	The start iteration to remove some burn-in iterations. Default is start = 1.
end	the last iteration to include. Default is NULL which set end to the last iteration in the chain.
thin	keep every thin iteration after the start iteration. Default is thin = 1, i.e., keep every iteration.
...	Additional arguments affecting the plot.

---

pmcmc *Particle Markov chain Monte Carlo (PMCMC) algorithm*

Description

Estimates model parameters using the PMCMC algorithm, which combines a particle filter for likelihood evaluation with a Metropolis-Hastings MCMC sampler for parameter estimation.

Usage

```
pmcmc(
  model,
  obs_process,
  data,
  priors,
  n_particles,
  n_iterations,
  theta = NULL,
  covmat = NULL,
  adaptmix = 0.05,
  adaptive = 100,
  post_proposal = NULL,
  init_model = NULL,
  post_particle = NULL,
  chain = NULL,
  verbose = FALSE
)

## S4 method for signature 'SimInf_model'
pmcmc(
  model,
  obs_process,
  data,
```

```
  priors,
  n_particles,
  n_iterations,
  theta = NULL,
  covmat = NULL,
  adaptmix = 0.05,
  adaptive = 100,
  post_proposal = NULL,
  init_model = NULL,
  post_particle = NULL,
  chain = NULL,
  verbose = FALSE
)
```

Arguments

model	The SimInf_model object to estimate parameters for.
obs_process	Specification of the stochastic observation process. The obs_process can be specified as a formula if the model contains only one node and there is only one data point for each time in data. The left hand side of the formula must match a column name in the data data.frame and the right hand side of the formula is a character specifying the distribution of the observation process, for example, Iobs ~ poisson(I). The following distributions are supported: x ~ binomial(size, prob), x ~ poisson(rate) and x ~ uniform(min, max). The observation process can also be a function to evaluate the probability density of the observations given the simulated states. The first argument passed to the obs_process function is the result from a run of the model and it contains one trajectory with simulated data for a time-point, where the trajectory contains n_particles replicates, see trajectory,SimInf_model-method. The second argument to the obs_process function is a data.frame containing the rows for the specific time-point that the function is called for. Note that the function must return the log of the density.
data	A data.frame holding the time series data.
priors	The priors for the parameters to fit. Each prior is specified with a formula notation, for example, beta ~ uniform(0, 1) specifies that beta is uniformly distributed between 0 and 1. Use c() to provide more than one prior, for example, c(beta ~ uniform(0, 1), gamma ~ normal(10, 1)). The following distributions are supported: gamma, lognormal, normal and uniform. All parameters in priors must be only in either gdata or ldata.
n_particles	An integer with the number of particles (> 1) to use at each timestep.
n_iterations	An integer specifying the number of iterations to run the PMCMC.
theta	A named numeric vector with initial parameter values. Default is NULL, which triggers sampling from the prior distribution(s).
covmat	A named numeric (npars x npars) covariance matrix for the proposal distribution. Default is NULL, which uses diag((theta/10)^2/npars).

`adaptmix` Numeric mixing proportion (0 < value < 1) for adaptive covariance proposal. Default: 0.05. Larger values increase the likelihood of using the fixed initial covariance instead of the adaptive estimate.

`adaptive` Integer (>= 0) specifying the iteration at which to start adaptive covariance updates. Default: 100. Set to 0 to disable adaptive updates.

`post_proposal` An optional function that, if non-`NULL`, is applied on the model after the proposal has been set for the model, but before running the particle filter. The function must accept one argument of type `SimInf_model` with the current model of the fitting process. This function can be useful to update, for example, `ldata` of the model before running a trajectory with proposed parameters. The function must return the model object which is then used in the particle filter.

`init_model` Optional function applied before each particle filter run. Must accept a `SimInf_model` object and return the modified model. Useful for setting initial states (`u0`, `v0`) before running trajectories with proposed parameters.

`post_particle` An optional function that, if non-NULL, is applied after each completed particle. The function must accept three arguments: 1) an object of `SimInf_pmcmc` with the current state of the fitting process, 2) an object `SimInf_pfilter` with the last particle and one filtered trajectory attached, and 3) an integer with the iteration in the fitting process. This function can be useful to, for example, monitor, save and inspect intermediate results. Note that the second `SimInf_pfilter` argument, is non-NULL only for the first particle in the chain, and for accepted particles.

`chain` Optional `data.frame` or object coercible to one, containing a previous PMCMC chain to continue from. If provided, `theta` and `covmat` must be `NULL`, and `n_iterations` can be 0.

`verbose` prints diagnostic messages when `TRUE`. Default is `FALSE`. When `verbose=TRUE`, information is printed every 100 iterations. For pmcmc, it is possible to get information every nth information by specifying `verbose=n`, for example, `verbose=1` or `verbose=10`.

Value

A `SimInf_pmcmc` object containing the fitted parameters and diagnostic information for all iterations.

References

C. Andrieu, A. Doucet and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society, Series B* **72**, 269–342, 2010. doi:10.1111/j.14679868.2009.00736.x

G. O. Roberts and J. S. Rosenthal. Examples of adaptive MCMC. *Journal of computational and graphical statistics*, **18**(2), 349–367, 2009. doi:10.1198/jcgs.2009.06134

See Also

`continue_pmcmc` for running additional iterations.

`prevalence` *Generic function to calculate prevalence from trajectory data*

Description

Calculate the proportion of *cases* (specified on the left-hand side of the formula) relative to the *population at risk* (specified on the right-hand side) at different aggregation levels. The function supports three levels of calculation:

- **Level 1 (Population Prevalence):** The proportion of cases in the total population at risk across all nodes.
- **Level 2 (Node Prevalence):** The proportion of nodes that contain at least one case, considering only nodes where the population at risk is greater than zero.
- **Level 3 (Within-Node Prevalence):** The proportion of cases relative to the population at risk calculated separately for each node.

Usage

```
prevalence(model, formula, level = 1, index = NULL, ...)
```

Arguments

`model`	The `model` with trajectory data to calculate the prevalence from.
`formula`	A formula that specifies the compartments that define the cases with a disease or that have a specific characteristic (numerator), and the compartments that define the entire population of interest (denominator). The left-hand-side of the formula defines the cases, and the right-hand-side defines the population, for example, `I~S+I+R` in a 'SIR' model (see 'Examples'). The `.` (dot) is expanded to all compartments, for example, `I~.` is expanded to `I~S+I+R` in a 'SIR' model (see 'Examples'). The formula can also contain a condition (indicated by `|`) for each node and time step to further control the population to include in the calculation, for example, `I ~ . | R == 0` to calculate the prevalence when the recovered is zero in a 'SIR' model. The condition must evaluate to `TRUE` or `FALSE` in each node and time step. Please note, if the denominator is zero, the prevalence is `NaN`. Additionally, when `level=3` (within-node prevalence) and the formula contains a condition that evaluates to `FALSE`, the prevalence is also `NaN`.
`level`	The level at which the prevalence is calculated at each time point in `tspan`. 1 (population prevalence): calculates the proportion of the individuals (cases) in the population. 2 (node prevalence): calculates the proportion of nodes with at least one case. 3 (within-node prevalence): calculates the proportion of cases within each node. Default is `1`.
`index`	indices specifying the subset of nodes to include when extracting data. Default (`index = NULL`) is to extract data from all nodes.
`...`	Additional arguments, see `prevalence`

`prevalence,SimInf_model-method`

Calculate prevalence from a model object with trajectory data

Description

Calculate the proportion of *cases* (specified on the left-hand side of the formula) relative to the *population at risk* (specified on the right-hand side) at different aggregation levels. The function supports three levels of calculation:

- **Level 1 (Population Prevalence):** The proportion of cases in the total population at risk across all nodes.
- **Level 2 (Node Prevalence):** The proportion of nodes that contain at least one case, considering only nodes where the population at risk is greater than zero.
- **Level 3 (Within-Node Prevalence):** The proportion of cases relative to the population at risk calculated separately for each node.

Usage

```
## S4 method for signature 'SimInf_model'
prevalence(model, formula, level, index, format = c("data.frame", "matrix"))
```

Arguments

`model`	The `model` with trajectory data to calculate the prevalence from.
`formula`	A formula that specifies the compartments that define the cases with a disease or that have a specific characteristic (numerator), and the compartments that define the entire population of interest (denominator). The left-hand-side of the formula defines the cases, and the right-hand-side defines the population, for example, `I~S+I+R` in a 'SIR' model (see 'Examples'). The `.` (dot) is expanded to all compartments, for example, `I~.` is expanded to `I~S+I+R` in a 'SIR' model (see 'Examples'). The formula can also contain a condition (indicated by `|`) for each node and time step to further control the population to include in the calculation, for example, `I ~ . | R == 0` to calculate the prevalence when the recovered is zero in a 'SIR' model. The condition must evaluate to `TRUE` or `FALSE` in each node and time step. Please note, if the denominator is zero, the prevalence is `NaN`. Additionally, when `level=3` (within-node prevalence) and the formula contains a condition that evaluates to `FALSE`, the prevalence is also `NaN`.
`level`	The level at which the prevalence is calculated at each time point in `tspan`. 1 (population prevalence): calculates the proportion of the individuals (cases) in the population. 2 (node prevalence): calculates the proportion of nodes with at least one case. 3 (within-node prevalence): calculates the proportion of cases within each node. Default is `1`.
`index`	indices specifying the subset of nodes to include when extracting data. Default (`index = NULL`) is to extract data from all nodes.

`format` The default (`format = "data.frame"`) is to generate a `data.frame` with one row per time-step with the prevalence. Using `format = "matrix"` returns the result as a matrix.

Value

A `data.frame` if `format = "data.frame"`, else a matrix.

Examples

```
## For reproducibility, specify the number of threads.
set_num_threads(1)

## Create an 'SIR' model with 6 nodes.
u0 <- data.frame(
  S = 100:105,
  I = c(0, 1, 0, 2, 0, 3),
  R = rep(0, 6)
)

model <- SIR(
  u0 = u0,
  tspan = 1:10,
  beta = 0.16,
  gamma = 0.077
)

## Run the model with a fixed seed for reproducibility.
result <- run(model, seed = 12)

## 1. Population Prevalence (level = 1, default)
## Proportion of infected individuals in the total population.
prevalence(result, I ~ S + I + R)

## Shorthand: '.' represents all compartments in the model (S + I + R).
prevalence(result, I ~ .)

## 2. Node Prevalence (level = 2)
## Proportion of nodes with at least one infected individual.
prevalence(result, I ~ S + I + R, level = 2)

## 3. Within-Node Prevalence (level = 3)
## Prevalence calculated separately for each node.
prevalence(result, I ~ S + I + R, level = 3)

## 4. Conditional Prevalence
## Calculate prevalence only in nodes where the number of
## recovered is zero.
prevalence(result, I ~ S + I + R | R == 0, level = 3)
```

`prevalence,SimInf_pfilter-method`

Extract prevalence from running a particle filter

Description

Calculate the proportion of *cases* (specified on the left-hand side of the formula) relative to the *population at risk* (specified on the right-hand side) at different aggregation levels. The function supports three levels of calculation:

- **Level 1 (Population Prevalence):** The proportion of cases in the total population at risk across all nodes.
- **Level 2 (Node Prevalence):** The proportion of nodes that contain at least one case, considering only nodes where the population at risk is greater than zero.
- **Level 3 (Within-Node Prevalence):** The proportion of cases relative to the population at risk calculated separately for each node.

Usage

```
## S4 method for signature 'SimInf_pfilter'
prevalence(model, formula, level, index, format = c("data.frame", "matrix"))
```

Arguments

`model`	the `SimInf_pfilter` object to extract the prevalence from.
`formula`	A formula that specifies the compartments that define the cases with a disease or that have a specific characteristic (numerator), and the compartments that define the entire population of interest (denominator). The left-hand-side of the formula defines the cases, and the right-hand-side defines the population, for example, `I~S+I+R` in a 'SIR' model (see 'Examples'). The `.` (dot) is expanded to all compartments, for example, `I~.` is expanded to `I~S+I+R` in a 'SIR' model (see 'Examples'). The formula can also contain a condition (indicated by `|`) for each node and time step to further control the population to include in the calculation, for example, `I ~ . | R == 0` to calculate the prevalence when the recovered is zero in a 'SIR' model. The condition must evaluate to `TRUE` or `FALSE` in each node and time step. Please note, if the denominator is zero, the prevalence is `NaN`. Additionally, when `level=3` (within-node prevalence) and the formula contains a condition that evaluates to `FALSE`, the prevalence is also `NaN`.
`level`	The level at which the prevalence is calculated at each time point in `tspan`. 1 (population prevalence): calculates the proportion of the individuals (cases) in the population. 2 (node prevalence): calculates the proportion of nodes with at least one case. 3 (within-node prevalence): calculates the proportion of cases within each node. Default is `1`.
`index`	indices specifying the subset of nodes to include when extracting data. Default (`index = NULL`) is to extract data from all nodes.

format — The default (format = "data.frame") is to generate a data.frame with one row per time-step with the prevalence. Using format = "matrix" returns the result as a matrix.

Value

A data.frame if format = "data.frame", else a matrix.

---

prevalence,SimInf_pmcmc-method

Extract prevalence from fitting a PMCMC algorithm

Description

Calculate the proportion of *cases* (specified on the left-hand side of the formula) relative to the *population at risk* (specified on the right-hand side) at different aggregation levels. The function supports three levels of calculation:

- **Level 1 (Population Prevalence):** The proportion of cases in the total population at risk across all nodes.
- **Level 2 (Node Prevalence):** The proportion of nodes that contain at least one case, considering only nodes where the population at risk is greater than zero.
- **Level 3 (Within-Node Prevalence):** The proportion of cases relative to the population at risk calculated separately for each node.

Usage

```
## S4 method for signature 'SimInf_pmcmc'
prevalence(model, formula, level, index, start = 1, end = NULL, thin = 1)
```

Arguments

model — the SimInf_pmcmc object to extract the prevalence from.

formula — A formula that specifies the compartments that define the cases with a disease or that have a specific characteristic (numerator), and the compartments that define the entire population of interest (denominator). The left-hand-side of the formula defines the cases, and the right-hand-side defines the population, for example, I~S+I+R in a 'SIR' model (see 'Examples'). The . (dot) is expanded to all compartments, for example, I~. is expanded to I~S+I+R in a 'SIR' model (see 'Examples'). The formula can also contain a condition (indicated by |) for each node and time step to further control the population to include in the calculation, for example, I ~ . | R == 0 to calculate the prevalence when the recovered is zero in a 'SIR' model. The condition must evaluate to TRUE or FALSE in each node and time step. Please note, if the denominator is zero, the prevalence is NaN. Additionally, when level=3 (within-node prevalence) and the formula contains a condition that evaluates to FALSE, the prevalence is also NaN.

level	The level at which the prevalence is calculated at each time point in tspan. 1 (population prevalence): calculates the proportion of the individuals (cases) in the population. 2 (node prevalence): calculates the proportion of nodes with at least one case. 3 (within-node prevalence): calculates the proportion of cases within each node. Default is 1.
index	indices specifying the subset of nodes to include when extracting data. Default (index = NULL) is to extract data from all nodes.
start	The start iteration to remove some burn-in iterations. Default is start = 1.
end	the last iteration to include. Default is NULL which set end to the last iteration in the chain.
thin	keep every thin iteration after the start iteration. Default is thin = 1, i.e., keep every iteration.

Value

A data.frame where the first column is the iteration and the remaining columns are the result from calling prevalence with the arguments formula, level and index for each iteration.

---

punchcard<- *Set a sparse recording template for simulation results*

Description

Define a template (or "punchcard") specifying which data points (nodes, time-points, and compartments) should be recorded during a simulation. This feature is useful for saving memory when the model has many nodes or time-points, but only a subset of the data is needed for post-processing.

Usage

```
punchcard(model) <- value

## S4 replacement method for signature 'SimInf_model'
punchcard(model) <- value
```

Arguments

model	A SimInf_model object.
value	A data.frame defining the recording template, or NULL to reset the model to record all compartments for all nodes at all time-points in tspan.

Details

The template is specified as a data.frame with columns:

- time: The time-points to record.
- node: The node indices to record.

- `CompartmentName`: Logical columns (e.g., `S`, `I`, `R`) indicating whether to record that specific compartment for the corresponding row. If a compartment column is omitted, it defaults to `FALSE` (not recorded). If only `time` and `node` are provided, all compartments are recorded for those points.

Important: The specified `time` values, `node` indices, and compartment names must exist in the model. If any value in the template does not match the model's configuration (e.g., a time point not in `tspan` or a compartment not defined in the model), an **error will be raised** when the template is set.

Note that the template only affects which data is stored in the result object; it does not affect how the solver simulates the trajectory.

Examples

```
## For reproducibility, specify the number of threads.
set_num_threads(1)

## Create an 'SIR' model with 6 nodes
u0 <- data.frame(
  S = 100:105,
  I = 1:6,
  R = rep(0, 6)
)
model <- SIR(
  u0 = u0,
  tspan = 1:10,
  beta = 0.16,
  gamma = 0.077
)

## Run the model with default recording (all data).
## For reproducibility, specify the seed.
result_full <- run(model, seed = 22)
head(trajectory(result_full))

## Define a template to record only nodes 2 and 4 at times 3 and 5
## for all compartments.
df <- data.frame(
  time = c(3, 5, 3, 5),
  node = c(2, 2, 4, 4),
  S = TRUE, I = TRUE, R = TRUE
)
punchcard(model) <- df

result_sparse <- run(model, seed = 22)
trajectory(result_sparse)

## Record only specific compartments (e.g., S and R, but not I)
df <- data.frame(
  time = c(3, 5, 3, 5),
  node = c(2, 2, 4, 4),
  S = TRUE, I = FALSE, R = TRUE
```

```
)
punchcard(model) <- df
result_partial <- run(model, seed = 22)
trajectory(result_partial)

## Shortcut: If only 'time' and 'node' are specified, all
## compartments are recorded.
df <- data.frame(
  time = c(3, 5, 3, 5),
  node = c(2, 2, 4, 4)
)
punchcard(model) <- df

## Reset to record all data (equivalent to no template)
punchcard(model) <- NULL
result_reset <- run(model, seed = 22)
head(trajectory(result_reset))
```

---

run *Run a SimInf model simulation*

Description

Simulate a trajectory from a SimInf model. The function compiles and loads model-specific C code (if not already compiled), initializes the solver, and advances the simulation in continuous time from the first to the last time point in tspan. The state of the system is recorded at each time point specified in tspan. Sparse output can be requested with punchcard<- to store only selected time points or compartments.

Usage

```
run(model, ...)

## S4 method for signature 'SimInf_model'
run(model, ...)

## S4 method for signature 'SEIR'
run(model, ...)

## S4 method for signature 'SIR'
run(model, ...)

## S4 method for signature 'SIS'
run(model, ...)

## S4 method for signature 'SISe'
run(model, ...)
```

```
## S4 method for signature 'SISe3'
run(model, ...)

## S4 method for signature 'SISe3_sp'
run(model, ...)

## S4 method for signature 'SISe_sp'
run(model, ...)

## S4 method for signature 'SimInf_abc'
run(model, ...)
```

Arguments

`model`	The SimInf model to run.
`...`	Optional arguments that affect the simulation:

solver Character string specifying the numerical solver. Either `"ssm"` (default) or `"aem"`. If not provided, the `"ssm"` solver is used. See details.

seed Numeric or integer specifying the random seed for the simulation. If not provided, a seed is randomly sampled from the current R RNG state.

crn Logical. If `TRUE`, use common random numbers for the `"mssm"` solver by assigning each node and replicate a separate random number stream, enabling variance reduction across simulation runs. If not provided, `crn` is `FALSE`.

rng_type Character string specifying the random number generator algorithm. Either `"mt19937"` or `"taus2"`. If not provided, `"mt19937"` is used.

Details

The solver uses a split-step method: for each unit of time, it first integrates the continuous-time Markov chain within each node using direct SSA (Gillespie's algorithm), then processes scheduled events (exit, enter, internal transfer, and external transfer), and finally calls the post time step function to update continuous state variables; see Widgren and others (2019) for details.

Two numerical solvers are available. The default solver is `"ssm"` (split-step method), which becomes `"mssm"` (multi-model split-step method) automatically when the model contains multiple replicates (`replicates > 1`). The alternative solver is `"aem"` (all events method), which assigns each reaction channel its own random number stream for consistent operational times across simulations; see Bauer and Engblom (2015). The `"aem"` solver cannot be used with replicated models.

Value

The model object with a single stochastic trajectory attached.

References

S. Widgren, P. Bauer, R. Eriksson and S. Engblom. **SimInf**: An R Package for Data-Driven Stochastic Disease Spread Simulations. *Journal of Statistical Software*, **91**(12), 1–42, 2019. doi:10.18637/jss.v091.i12. An updated version of this paper is available as a vignette in the package.

P. Bauer, S. Engblom and S. Widgren. Fast Event-Based Epidemiological Simulations on National Scales. *International Journal of High Performance Computing Applications*, **30**(4), 438–453, 2016. doi: 10.1177/1094342016635723

P. Bauer and S. Engblom. Sensitivity Estimation and Inverse Problems in Spatial Stochastic Models of Chemical Kinetics. In: A. Abdulle, S. Deparis, D. Kressner, F. Nobile and M. Picasso (eds.), *Numerical Mathematics and Advanced Applications - ENUMATH 2013*, pp. 519–527, Lecture Notes in Computational Science and Engineering, vol 103. Springer, Cham, 2015. doi:10.1007/9783319107059_51

Examples

```
## For reproducibility, specify the number of threads to use.
set_num_threads(1)

## Create an 'SIR' model with 10 nodes.
model <- SIR(
  u0 = data.frame(
    S = rep(99, 10),
    I = rep(1, 10),
    R = rep(0, 10)
  ),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Run the model with a fixed seed for reproducibility.
result <- run(model, seed = 22)

## Plot the distribution of susceptible, infected and recovered
## individuals.
plot(result)
```

SEIR *Create an SEIR model*

Description

Create an SEIR model to be used by the simulation framework.

Usage

```
SEIR(u0, tspan, events = NULL, beta = NULL, epsilon = NULL, gamma = NULL)
```

Arguments

u0	A `data.frame` with the initial state in each node, i.e., the number of individuals in each compartment in each node when the simulation starts (see 'Details'). The parameter `u0` can also be an object that can be coerced to a `data.frame`, e.g., a named numeric vector will be coerced to a one row `data.frame`.

`tspan` A vector (length >= 1) of increasing time points where the state of each node is to be returned. Can be either an `integer` or a `Date` vector.

- If `integer`: Represents the specific time points (e.g., days, hours) at which to record the state.
- If `Date`: Coerced to a numeric vector representing the **day of the year** (1–366) relative to the first date in the vector. The original `Date` objects are preserved as names for the numeric vector, facilitating time-series plotting.

`events` a `data.frame` with the scheduled events, see `SimInf_model`.

`beta` A numeric vector with the transmission rate from susceptible to infected. Each node can have a different beta value. The vector must have length 1 or `nrow(u0)`. If the vector has length 1 but the model contains more nodes, the beta value is repeated for all nodes.

`epsilon` A numeric vector with the incubation rate from exposed to infected. Each node can have a different value. The vector must have length 1 or `nrow(u0)`. If the vector has length 1 but the model contains more nodes, the value is repeated for all nodes.

`gamma` A numeric vector with the recovery rate from infected to recovered. Each node can have a different gamma value. The vector must have length 1 or `nrow(u0)`. If the vector has length 1 but the model contains more nodes, the gamma value is repeated for all nodes.

Details

The SEIR model extends the standard SIR model by adding an **E**xposed (E) compartment for individuals who have been infected but are not yet infectious. This accounts for the latent period of the disease.

The model is defined by three state transitions:

$$S \xrightarrow{\beta SI/N} E$$

$$E \xrightarrow{\epsilon E} I$$

$$I \xrightarrow{\gamma I} R$$

where β is the transmission rate, ϵ is the incubation rate (inverse of the latent period), γ is the recovery rate, and $N = S + E + I + R$ is the total population size in each node. Here, S, E, I, and R represent the number of susceptible, exposed, infected, and recovered individuals in that specific node.

The argument `u0` must be a `data.frame` with one row for each node with the following columns:

S The number of susceptible individuals in each node

E The number of exposed individuals in each node

I The number of infected individuals in each node

R The number of recovered individuals in each node

Value

A `SimInf_model` of class `SEIR`

See Also

SEIR for the class definition. SIR, SIS, SISe, SISe3 and SISe_sp for other predefined models. mparse for creating custom models. run for running the simulation. trajectory, prevalence and plot for post-processing and visualization.

Examples

```
## Create an SEIR model object.
model <- SEIR(
  u0 = data.frame(S = 99, E = 0, I = 1, R = 0),
  tspan = 1:100,
  beta = 0.16,
  epsilon = 0.25,
  gamma = 0.077
)

## Run the SEIR model with a fixed seed for reproducibility.
result <- run(model, seed = 22)

## Plot the distribution of susceptible, exposed, infected and
## recovered individuals.
plot(result)
```

SEIR-class *Class SEIR*

Description

Class to handle the SEIR model. This class inherits from SimInf_model, meaning that SEIR objects are fully compatible with all generic functions defined for SimInf_model, such as run, plot, trajectory, and prevalence.

Details

The SEIR model extends the standard SIR model by adding an **E**xposed (E) compartment for individuals who have been infected but are not yet infectious. This accounts for the latent period of the disease.

The model is defined by three state transitions:

$$S \xrightarrow{\beta SI/N} E$$

$$E \xrightarrow{\epsilon E} I$$

$$I \xrightarrow{\gamma I} R$$

where β is the transmission rate, ϵ is the incubation rate (inverse of the latent period), γ is the recovery rate, and $N = S + E + I + R$ is the total population size in each node. Here, S, E, I, and R represent the number of susceptible, exposed, infected, and recovered individuals in that specific node.

See Also

SEIR for creating an SEIR model object, SimInf_model for the parent class definition, and SIR for the base model without the latent period.

`select_matrix` *Extract the select matrix from a* `SimInf_model` *object*

Description

Utility function to extract `events@E` from a `SimInf_model` object, see SimInf_events

Usage

```
select_matrix(model)

## S4 method for signature 'SimInf_model'
select_matrix(model)
```

Arguments

`model`	The `model` to extract the select matrix `E` from.

Value

dgCMatrix object.

Examples

```
## Create an SIR model
model <- SIR(u0 = data.frame(S = 99, I = 1, R = 0),
             tspan = 1:5, beta = 0.16, gamma = 0.077)

## Extract the select matrix from the model
select_matrix(model)
```

`select_matrix<-` *Set the select matrix for a* `SimInf_model` *object*

Description

Utility function to set `events@E` in a `SimInf_model` object, see SimInf_events.

Usage

```
select_matrix(model) <- value

## S4 replacement method for signature 'SimInf_model'
select_matrix(model) <- value
```

Arguments

model	The SimInf_model object to set the select matrix for.
value	The new value for E in the model. E is a matrix to handle scheduled events, see SimInf_events. Each row in E corresponds to one compartment in the model. The non-zero entries in a column indicate the compartments to include in an event. For the *exit*, *internal transfer* and *external transfer* events, the values in E[,select] are used as weights when sampling individuals without replacement, with probability proportional to the weight. For the *enter* event, the values in E[, select] are used as weights when determining which compartment to add individuals to. If the column E[, select] contains several non-zero entries, the compartment is sampled with probability proportional to the weight in E[, select]. The select matrix E can either be specified as a matrix, or as a data.frame. When E is specified as a data.frame, it must have one column named compartment that defines which compartment is referred to, and one column select that defines the column in E. In addition, the data.frame can contain an optional column named value with the value in E. When the value column is missing, 1 is used as the default value.

Examples

```
## Create an SIR model.
model <- SIR(u0 = data.frame(S = 99, I = 1, R = 0),
             tspan = 1:5, beta = 0.16, gamma = 0.077)

## Set the select matrix.
select_matrix(model) <- matrix(c(1, 0, 0, 1, 1, 1, 0, 0, 1), nrow = 3)

## Extract the select matrix from the model.
select_matrix(model)

## Set the select matrix using a data.frame instead.
select_matrix(model) <- data.frame(
    compartment = c("S", "S", "I", "R", "R"),
    select      = c( 1,   2,   2,   2,   3))

## Extract the select matrix from the model.
select_matrix(model)
```

---

set_num_threads *Specify the number of threads that SimInf should use*

Description

Set the number of threads to be used in SimInf code that is parallelized with OpenMP (if available).

Usage

```
set_num_threads(threads = NULL)
```

Arguments

threads	Integer specifying the maximum number of threads to use in OpenMP-parallelized functions. If NULL (default), SimInf attempts to use all available processors, subject to the limits imposed by the environment variables listed above.

Details

The number of threads is initialized when SimInf is first loaded in the R session, based on optional environment variables (see 'Details'). It can also be explicitly set by calling set_num_threads. If the environment variables affecting the thread count change, set_num_threads must be called again for the new values to take effect.

The function determines the number of available processors using omp_get_num_procs() and limits the thread count based on omp_get_thread_limit() and the following environment variables (checked in order of precedence):

- SIMINF_NUM_THREADS: Specific limit for SimInf.
- OMP_NUM_THREADS: General OpenMP limit.
- OMP_THREAD_LIMIT: Maximum thread limit.

The threads argument allows you to override these limits, provided the requested value does not exceed the maximum allowed by the environment or hardware constraints.

Value

The previous value of the thread count is returned invisibly.

---

shift_matrix *Extract the shift matrix from a* SimInf_model *object*

Description

Utility function to extract the shift matrix events@N from a SimInf_model object, see SimInf_events

Usage

```
shift_matrix(model)

## S4 method for signature 'SimInf_model'
shift_matrix(model)
```

Arguments

model	The model to extract the shift matrix events@N from.

Value

A mtrix.

Examples

```
## Create an SIR model
model <- SIR(u0 = data.frame(S = 99, I = 1, R = 0),
             tspan = 1:5, beta = 0.16, gamma = 0.077)

## Extract the shift matrix from the model
shift_matrix(model)
```

---

shift_matrix<- *Set the shift matrix for a* SimInf_model *object*

Description

Utility function to set events@N in a SimInf_model object, see SimInf_events.

Usage

```
shift_matrix(model) <- value

## S4 replacement method for signature 'SimInf_model'
shift_matrix(model) <- value
```

Arguments

model	The SimInf_model object to set the shift matrix for.
value	The new value for N in the model. N is a matrix to handle scheduled events, see SimInf_events. Each row in N corresponds to one compartment in the model. The values in a column define how to move sampled individuals before adding them to the destination. Let q <- shift, then each non-zero entry in N[, q] defines the number of rows to move sampled individuals from that compartment i.e., sampled individuals from compartment p are moved to compartment N[p, q] + p, where 1 <= N[p, q] + p <= N_compartments. This matrix is used for *enter*, *internal transfer* and *external transfer* events. The shift matrix N can either be specified as a matrix, or as a data.frame. When N is specified as a data.frame, it must have one column named compartment that defines which compartment is referred to, and one column shift that defines the column in N. In addition, the data.frame must contain a column named value with the integer value in N.

Value

SimInf_model object

Examples

```
## Create an SIR model
model <- SIR(u0 = data.frame(S = 99, I = 1, R = 0),
             tspan = 1:5, beta = 0.16, gamma = 0.077)

## Set the shift matrix.
shift_matrix(model) <- matrix(c(2, 1, 0), nrow = 3)

## Extract the shift matrix from the model.
shift_matrix(model)

## Set the shift matrix using a data.frame instead.
shift_matrix(model) <- data.frame(
    compartment = c("S", "I", "R"),
    shift = c(1, 1, 1),
    value = c(2, 1, 0))

## Extract the shift matrix from the model.
shift_matrix(model)
```

---

show,SimInf_abc-method

Brief summary of a SimInf_abc *object*

Description

Display a summary of an approximate Bayesian computation (ABC) analysis, including the number of particles per generation, the number of generations, and—for the final generation—the acceptance rate, effective sample size (ESS), and summary statistics (minimum, first quartile, median, mean, third quartile, and maximum) for each parameter in the posterior distribution.

Usage

```
## S4 method for signature 'SimInf_abc'
show(object)
```

Arguments

object	The SimInf_abc object to display.

Value

The object, returned invisibly.

See Also

abc for running an ABC analysis, continue_abc for continuing an existing ABC chain, SimInf_abc for the class definition.

show,SimInf_events-method

Brief summary of a SimInf_events *object*

Description

Display the number of scheduled events in a SimInf_events object. The count reflects the total number of event entries in the event schedule, including both internal (E1) and external (E2) events.

Usage

```
## S4 method for signature 'SimInf_events'
show(object)
```

Arguments

object	The SimInf_events object to display.

Value

The object, returned invisibly.

See Also

SimInf_events for creating event objects, SimInf_events for the class definition, SimInf_model for how events are attached to a model.

Examples

```
## Create an 'SIR' model with 1600 cattle herds (nodes) and
## scheduled events for the population of nodes with births,
## deaths and between-node movements of individuals. Define
## 'tspan' to record the state of the system at daily time
## points over 4*365 days.
model <- SIR(
  u0     = u0_SIR(),
  tspan  = 1:(4 * 365),
  events = events_SIR(),
  beta   = 0.16,
  gamma  = 0.077
)

## Brief summary of the events
events(model)
```

show,SimInf_individual_events-method

Brief summary of a SimInf_individual_events *object*

Description

Display the number of unique individuals and the total number of event records in a SimInf_individual_events object. The individual count reflects the distinct id values, while the event count represents all recorded events (exit, enter, internal transfer, and external transfer).

Usage

```
## S4 method for signature 'SimInf_individual_events'
show(object)
```

Arguments

object	The SimInf_individual_events object to display.

Value

The object, returned invisibly.

See Also

individual_events for creating SimInf_individual_events objects, SimInf_individual_events for the class definition, SimInf_events for the node-level event class.

show,SimInf_model-method

Brief summary of a SimInf_model *object*

Description

Display key characteristics of a SimInf_model object, including the model name, number of nodes, number of replicates (if greater than one), number of transitions, scheduled events, global data, local data, continuous state variables, and compartments.

Usage

```
## S4 method for signature 'SimInf_model'
show(object)
```

Arguments

object	The SimInf_model object to display.

Details

The output differs depending on whether the model has been run: before running, the result matrices (U and V) are empty; after calling run, they contain the simulated trajectory data and the summary reflects the computed results.

Value

The object, returned invisibly.

See Also

SimInf_model for creating model objects, run for simulating a trajectory from a model, SimInf_model for the class definition.

Examples

```
## For reproducibility, specify the number of threads.
set_num_threads(1)

## Create an 'SIR' model with 10 nodes and initialise
## it to run over 100 days.
model <- SIR(
  u0 = data.frame(
    S = rep(99, 10),
    I = rep(1, 10),
    R = rep(0, 10)
  ),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Brief summary of the model
model

## Run the model with a fixed seed for reproducibility.
result <- run(model, seed = 22)

## Brief summary of the result. Note that the trajectory is
## non-empty after running the model.
result
```

---

show,SimInf_pfilter-method

Brief summary of a SimInf_pfilter *object*

Description

Display a brief summary of a bootstrap particle filter analysis, including the number of particles used and the estimated log-likelihood.

Usage

```
## S4 method for signature 'SimInf_pfilter'
show(object)
```

Arguments

object	The SimInf_pfilter object to display.

Value

The object, returned invisibly.

See Also

pfilter for running a particle filter, logLik for extracting the log-likelihood, SimInf_pfilter for the class definition.

---

show,SimInf_pmcmc-method

Brief summary of a SimInf_pmcmc *object*

Description

Display a summary of a particle Markov chain Monte Carlo (PMCMC) analysis, including the number of iterations, number of particles, mixing proportion for the adaptive proposal, acceptance ratio, and posterior quantiles (2.5 standard deviation for each estimated parameter.

Usage

```
## S4 method for signature 'SimInf_pmcmc'
show(object)
```

Arguments

object	The SimInf_pmcmc object to display.

Value

The object, returned invisibly.

See Also

pmcmc for running a PMCMC analysis, continue_pmcmc for continuing an existing PMCMC chain, SimInf_pmcmc for the class definition.

SimInf *A Framework for Data-Driven Stochastic Disease Spread Simulations*

Description

The **SimInf** package provides a flexible framework for data-driven spatio-temporal disease spread modeling, combining the speed of compiled C code with the flexibility of R. It is designed to efficiently simulate disease transmission dynamics alongside population demographics and dynamic contact networks.

Details

The **SimInf** framework models infection dynamics within each subpopulation (node) as continuous-time Markov chains (CTMC) using the Gillespie stochastic simulation algorithm (SSA). Additionally, SimInf can incorporate data—such as births, deaths, and movements—as scheduled events. These events trigger at predefined time points and modify the state of subpopulations by randomly selecting a specified number of individuals from the affected compartments. This capability allows simulations to be driven by empirical records or synthetic scenarios while maintaining the stochastic nature of population demographics and movement networks.

The package supports both predefined models (e.g., `SIR`, `SIS`) and custom model specifications via the `mparse` function. `mparse` serves as the primary interface for defining custom compartment models, allowing users to describe transitions using a simple, human-readable string syntax in R. The function then parses this description, generates model-specific C code, and returns a `SimInf_model` object ready for simulation. This approach combines the flexibility of R with the computational speed of compiled code, making it well-suited for models with complex propensity functions, multiple compartments, or node-specific parameters. See the vignette "Getting started with mparse" for a detailed tutorial on defining custom models.

To facilitate the use of real livestock data, SimInf provides functionality for preparing individual event records (e.g., births, deaths, and movements) for simulation. The `individual_events` function cleans and validates raw livestock events, resolving inconsistencies and structuring the data into the format required by the simulation. This process transforms complex, individual-level logs into the scheduled events that drive population demographics and movement networks. See the vignette "Scheduled events" for a detailed guide on processing livestock data and integrating it into disease spread models.

After a model is created, a simulation is executed using the `run` method. Upon successful completion, `run` returns a new `SimInf_model` object containing the original configuration plus the simulated stochastic trajectory.

To inspect and analyze the results, SimInf provides a suite of utility functions:

- `summary` and `show` for a quick overview of the model structure and simulation results.
- `plot` for visualizing the time series of compartments and continuous state variables.
- `trajectory` for extracting the full time series data for custom analysis.
- `prevalence` for calculating and summarizing disease prevalence across nodes and time.

These functions facilitate both rapid exploratory analysis and detailed post-processing of simulation outcomes. See the vignette "Post-process data in a trajectory" for a comprehensive tutorial on extracting and analyzing simulation results.

Beyond simulation, the package provides functionality to fit models to time series data using two Bayesian inference methods:

- Approximate Bayesian Computation Sequential Monte Carlo (ABC-SMC), implemented in `abc`, based on the approach by Toni and others (2009) doi:10.1098/rsif.2008.0172.
- Particle Markov Chain Monte Carlo (PMCMC), implemented in `pmcmc`, based on the approach by Andrieu and others (2010) doi:10.1111/j.14679868.2009.00736.x.

Both methods enable parameter estimation in stochastic models where the likelihood function is intractable, by using simulated data to estimate the posterior distributions of model parameters.

Finally, one of the core design goals of SimInf is extensibility. The package provides functionality to generate the required C and R code from a model specification, enabling users to create custom R packages that leverage the numerical solvers of SimInf. This facilitates complex epidemiological research by allowing researchers to encapsulate their specific model structures within a standard R package, ensuring reproducibility and ease of sharing. See `package_skeleton` for details on creating a new R package based on a custom model specification.

Author(s)

Maintainer: Stefan Widgren <stefan.widgren@gmail.com> (ORCID)

Authors:

- Stefan Widgren <stefan.widgren@gmail.com> (ORCID)
- Robin Eriksson (ORCID)
- Stefan Engblom (ORCID)
- Pavol Bauer (ORCID)

Other contributors:

- Thomas Rosendal (ORCID) [contributor]
- Ivana Rodriguez Ewerlöf (ORCID) [contributor]
- Attractive Chaos (Author of 'kvec.h'.) [copyright holder]

References

S. Widgren, P. Bauer, R. Eriksson and S. Engblom. **SimInf**: An R Package for Data-Driven Stochastic Disease Spread Simulations. *Journal of Statistical Software*, **91**(12), 1–42, 2019. doi:10.18637/jss.v091.i12. An updated version of this paper is available as a vignette in the package.

See Also

Useful links:

- `https://github.com/stewid/SimInf`
- `https://stewid.github.io/SimInf/`
- Report bugs at `https://github.com/stewid/SimInf/issues`

SimInf_abc-class *Class* SimInf_abc

Description

Storage class for the results of an Approximate Bayesian Computation (ABC) parameter estimation using Sequential Monte Carlo (SMC). The SimInf_abc class holds the model definition, prior distributions, accepted parameter values (particles), weights, distances, and convergence diagnostics.

Slots

model A SimInf_model object containing the model structure (transitions, compartments, etc.) for which parameters are being estimated.

priors A data.frame defining the prior distributions for the parameters. It contains four columns:

- parameter: The name of the parameter in the model.
- distribution: The prior distribution type. Valid values are "gamma", "lognormal", "normal", or "uniform".
- p1: The first hyperparameter:
  - "gamma": *shape*
  - "lognormal": *meanlog* (mean on the log scale)
  - "normal": *mean*
  - "uniform": *lower* bound
- p2: The second hyperparameter:
  - "gamma": *rate*
  - "lognormal": *sdlog* (standard deviation on the log scale)
  - "normal": *sd* (standard deviation)
  - "uniform": *upper* bound

target Character vector ("gdata" or "ldata") that determines if the ABC-SMC method estimates parameters in model@gdata (global data) or in model@ldata (local data).

pars An integer vector with the indices of the parameters in target that are being estimated.

nprop An integer vector with the number of simulated proposals (particles) generated in each generation.

fn A function used to calculate summary statistics from the simulated trajectory and compute the distance for each particle. See abc for details on the required function signature.

tolerance A numeric matrix (number of summary statistics $\times$ number of generations) where each column contains the tolerances for a generation and each row contains a sequence of gradually decreasing tolerances.

x A numeric array (number of particles $\times$ number of parameters $\times$ number of generations) with the parameter values for the accepted particles in each generation. Each row is one particle.

weight A numeric matrix (number of particles $\times$ number of generations) with the weights for the particles x in the corresponding generation.

`distance` A numeric array (number of particles × number of summary statistics × number of generations) with the distance for the particles `x` in each generation. Each row contains the distance for a particle and each column contains the distance for a summary statistic.

`ess` A numeric vector with the effective sample size (ESS) in each generation. The effective sample size is computed as

$$\left(\sum_{i=1}^{N}(w_g^{(i)})^2\right)^{-1},$$

where $w_g^{(i)}$ is the normalized weight of particle i in generation g.

`init_model` An optional function that, if non-NULL, is applied before running each proposal. The function must accept one argument of type `SimInf_model` with the current model of the fitting process and return a modified model. This function can be useful to specify the initial state of `u0` or `v0` of the model before running a trajectory with proposed parameters.

See Also

`abc` for the main ABC function and `continue_abc` for continuing an ABC run.

`SimInf_events` *Create a* `SimInf_events` *object*

Description

The argument events must be a `data.frame` with the following columns:

event Four event types are supported by the current solvers: *exit*, *enter*, *internal transfer*, and *external transfer*. When assigning the events, they can either be coded as a numerical value or a character string: *exit;* `0` or `'exit'`, *enter;* `1` or `'enter'`, *internal transfer;* `2` or `'intTrans'`, and *external transfer;* `3` or `'extTrans'`. Internally in **SimInf**, the event type is coded as a numerical value.

time When the event occurs i.e., the event is processed when time is reached in the simulation. Can be either an `integer` or a `Date` vector. A `Date` vector is coerced to a numeric vector as days, where `t0` determines the offset to match the time of the events to the model `tspan` vector.

node The node that the event operates on. Also the source node for an *external transfer* event. 1 <= `node[i]` <= Number of nodes.

dest The destination node for an *external transfer* event i.e., individuals are moved from `node` to `dest`, where 1 <= `dest[i]` <= Number of nodes. Set `event = 0` for the other event types. `dest` is an integer vector.

n The number of individuals affected by the event. n[i] >= 0.

proportion If `n[i]` equals zero, the number of individuals affected by `event[i]` is calculated by sampling the number of individuals from a binomial distribution using the `proportion[i]` and the number of individuals in the compartments. Numeric vector. 0 <= proportion[i] <= 1.

select To process an `event[i]`, the compartments affected by the event are specified with `select[i]` together with the matrix `E`, where `select[i]` determines which column in `E` to use. The specific individuals affected by the event are sampled from the compartments corresponding to the non-zero entries in the specified column in `E[, select[i]]`, where `select` is an integer vector.

shift Determines how individuals in *enter*, *internal transfer* and *external transfer* events are shifted to enter another compartment. The sampled individuals are shifted according to column `shift[i]` in matrix `N` i.e., `N[, shift[i]]`, where `shift` is an integer vector. See above for a description of `N`. Unsued for the other event types.

Usage

```
SimInf_events(E = NULL, N = NULL, events = NULL, t0 = NULL)
```

Arguments

`E`	A matrix where each row corresponds to one compartment in the model. The non-zero entries in a column indicate the compartments to include in an event. For the *exit*, *internal transfer* and *external transfer* events, the values in `E[, select]` are used as weights when sampling individuals without replacement, with probability proportional to the weight. For the *enter* event, the values in `E[, select]` are used as weights when determining which compartment to add individuals to. If the column `E[, select]` contains several non-zero entries, the compartment is sampled with probability proportional to the weight in `E[, select]`. `E` is sparse matrix of class `dgCMatrix`.
`N`	Determines how individuals in *internal transfer* and *external transfer* events are shifted to enter another compartment. Each row corresponds to one compartment in the model. The values in a column are added to the current compartment of sampled individuals to specify the destination compartment, for example, a value of `1` in an entry means that sampled individuals in this compartment are moved to the next compartment. Which column to use for each event is specified by the `shift` vector (see below). `N` is an integer matrix.
`events`	A `data.frame` with events.
`t0`	If `events$time` is a `Date` vector, then `t0` determines the offset to match the time of the events to the model `tspan` vector, see details. If `events$time` is a numeric vector, then `t0` must be `NULL`.

Value

S4 class `SimInf_events`

Examples

```
## Let us illustrate how movement events can be used to transfer
## individuals from one node to another.  Use the built-in SIR
## model and start with 2 nodes where all individuals are in the
## first node (100 per compartment).
u0 <- data.frame(
  S = c(100, 0),
```

```
  I = c(100, 0),
  R = c(100, 0)
)

## Then create 300 movement events to transfer all individuals,
## one per day, from the first node to the second node. Use the
## fourth column in the select matrix where all compartments
## can be sampled with equal weight.
events <- data.frame(
  event      = rep("extTrans", 300),
  time       = 1:300,
  node       = 1,
  dest       = 2,
  n          = 1,
  proportion = 0,
  select     = 4,
  shift      = 0
)

## Create an SIR model without disease transmission to
## demonstrate the events.
model <- SIR(
  u0      = u0,
  tspan  = 1:300,
  events = events,
  beta   = 0,
  gamma  = 0
)

## Run the model and plot the number of individuals in
## the second node.  As can be seen in the figure, all
## indivuduals have been moved to the second node when
## t = 300.
plot(run(model), index = 1:2, range = FALSE)

## Let us now double the weight to sample from the 'I'
## compartment and rerun the model.
model@events@E[2, 4] <- 2
plot(run(model), index = 1:2, range = FALSE)

## And much larger weight to sample from the I compartment.
model@events@E[2, 4] <- 10
plot(run(model), index = 1:2, range = FALSE)

## Increase the weight for the R compartment.
model@events@E[3, 4] <- 4
plot(run(model), index = 1:2, range = FALSE)
```

SimInf_events-class *Class* SimInf_events

Description

Class to hold data for scheduled events that modify the discrete state of individuals in a node at a pre-defined time `t`.

Slots

`E` The **select matrix** (sparse matrix of class `dgCMatrix`). Each row corresponds to a model compartment.

- **Sampling (Exit, Internal/External Transfer):** Non-zero entries in a column indicate which compartments individuals are sampled from. The values in `E[, select]` act as **weights** for sampling individuals without replacement (probability proportional to weight).
- **Targeting (Enter):** Non-zero entries in a column indicate which compartments new individuals are added to. The values in `E[, select]` act as **weights** for distributing new individuals among the target compartments.

`N` The **shift matrix** (integer matrix). Determines how individuals are moved between compartments during *enter*, *internal transfer*, and *external transfer* events.

- Each row corresponds to a source compartment.
- Each column corresponds to a specific `shift` value.
- If `q <- shift`, the entry `N[p, q]` defines the **offset** (number of rows to move) for individuals sampled from compartment `p`.
- The destination compartment is calculated as: `destination = p + N[p, q]`.
- Constraint: `1 <= destination <= number of compartments`.

`event` Integer vector specifying the event type for each row:

- `0`: *exit* (remove individuals).
- `1`: *enter* (add individuals).
- `2`: *internal transfer* (move within node).
- `3`: *external transfer* (move between nodes).

Other values are reserved for future use.

`time` Integer vector specifying the time step when each event occurs.

`node` Integer vector specifying the **source node** for the event. For *external transfer*, this is the node individuals are moved *from*. Range: `1 <= node <= number of nodes`.

`dest` Integer vector specifying the **destination node** for *external transfer* events (individuals moved *to*). For other event types, this value is ignored (typically set to 0). Range: `1 <= dest <= number of nodes`.

`n` Integer vector specifying the **number of individuals** affected by the event. Must be `n >= 0`.

`proportion` Numeric vector. If `n[i] == 0`, the number of individuals is sampled from a binomial distribution using `proportion[i]` and the current population size in the selected compartments. Range: `0 <= proportion <= 1`.

`select` Integer vector specifying which **column** of the matrix `E` to use for sampling/targeting for each event. The specific individuals are chosen based on the non-zero entries in `E[, select[i]]`.

`shift` Integer vector specifying which **column** of the matrix `N` to use for shifting individuals for each event. Unused for *exit* events.

See Also

SimInf_model for the main model class that holds the events. SIR, SEIR, SIS, SISe for examples of how events are passed to model constructors. SimInf_model (constructor) for details on the E and N matrices used to define event behavior. run for executing the simulation with scheduled events. Vignette "Scheduled events" for detailed examples of defining and using events.

SimInf_individual_events-class

Class SimInf_individual_events

Description

Storage class for cleaned individual-level event data, such as births, deaths, and movements. This class serves as an intermediate step in the data preparation pipeline, holding raw records that will later be aggregated into the scheduled events (SimInf_events) required by a SimInf_model at predefined time-points.

Details

The typical workflow is:

1. Collect raw individual event data (e.g., from a database).
2. Clean and validate it using individual_events, which returns a SimInf_individual_events object.
3. Aggregate the individual events into node-level scheduled events for the model, for example using u0_from_individual_events to derive the initial state.

Slots

`id` An integer or character vector serving as a unique identifier for each individual.

`event` The type of event. Four types are supported: *exit*, *enter*, *internal transfer*, and *external transfer*. These can be specified as either a numeric code or a character string:

- `0` or `"exit"`: Individual leaves the system.
- `1` or `"enter"`: Individual enters the system.
- `2` or `"intTrans"`: Individual moves within the same node.
- `3` or `"extTrans"`: Individual moves to a different node.

`time` A numeric, character, or `Date` vector indicating when the event occurred. Character strings must be coercible to `Date` (e.g., "2023-01-15").

`node` An integer or character vector identifying the **source** node for the event.

`dest` An integer or character vector identifying the **destination** node. For *exit* and *enter* events, this value is typically `NA` or unused.

See Also

individual_events for cleaning and processing raw event data into this class format, u0_from_individual_events for deriving the initial state from these events, and SimInf_events for the node-level aggregated event format used by the solver.

SimInf_model *Create a* SimInf_model *object*

Description

Construct a low-level SimInf_model object. This function is typically used internally by model constructors (e.g., SIR(), mparse()) or for advanced usage where custom model definitions (e.g., user-provided C code or non-standard matrices) are required.

Usage

```
SimInf_model(
  G,
  S,
  tspan,
  events = NULL,
  ldata = NULL,
  gdata = NULL,
  U = NULL,
  u0 = NULL,
  v0 = NULL,
  V = NULL,
  E = NULL,
  N = NULL,
  replicates = NULL,
  C_code = NULL
)
```

Arguments

G	**Dependency Graph**. Indicates which transition rates need updating after a state transition. Can be provided as a sparse matrix (class dgCMatrix) or a dense matrix. If a dense matrix is provided, it is automatically converted to a sparse format internally. See SimInf_model for detailed matrix layout.
S	**State Transition Matrix**. Defines the change in the state vector for each transition. Can be provided as a sparse matrix (class dgCMatrix) or a dense matrix. If a dense matrix is provided, it is automatically converted to a sparse format internally. See SimInf_model for detailed matrix layout.
tspan	**Time Span** (numeric or Date vector). Increasing time points for output. If Date, converted to days with names, where tspan[1] becomes the day of the year of the first year of tspan. The dates are added as names to the numeric vector.
events	**Scheduled Events**. A data.frame defining the event schedule (see SimInf_events).
ldata	**Local Data**. Parameters specific to each node. Can be: • A data.frame with one row per node. • A matrix where each column ldata[, j] is the data vector for node j.

Passed to transition rate and post-step functions.

`gdata` **Global Data** (numeric vector). Parameters common to all nodes. Passed to transition rate and post-step functions.

`U` **Result Matrix** (integer matrix). Usually empty at creation. See `SimInf_model` for detailed matrix layout.

`u0` **Initial State**. Initial number of individuals per compartment/node. Can be:

- A matrix ($N_c \times N_n$).
- A `data.frame` with columns corresponding to compartments.
- Any object coercible to a `data.frame` (e.g., a named numeric vector will be coerced to a one-row `data.frame`).

`v0` **Initial Continuous State** (numeric matrix). Initial values for continuous states per node.

`V` **Continuous State Result Matrix** (numeric matrix). Usually empty at creation. See `SimInf_model` for layout.

`E` **Select Matrix** (matrix or `data.frame`). Defines which compartments are affected by events and their sampling weights.

- **Matrix**: Standard sparse matrix.
- **data.frame**: Must have columns `compartment` and `select`. Optional column `value` (default `1`) sets the weight.

See `SimInf_events` for usage details.

`N` **Shift Matrix** (matrix or `data.frame`). Defines how individuals are moved between compartments during events.

- **Matrix**: Standard integer matrix.
- **data.frame**: Must have columns `compartment`, `shift`, and `value` (integer offset).

See `SimInf_events` for usage details.

`replicates` Number of model replicates to simulate (default `NULL`, treated as `1L`). When `replicates > 1L`, each replicate is simulated independently using its own initial state (from `u0`), but shares the same parameters (`gdata`, `ldata`), scheduled events, and structure (transitions, compartments).

The `u0` argument must contain initial states for all replicates. Each replicate requires n nodes, where n is the number of nodes in the model. Thus, `u0` must have `replicates * n` nodes total. Nodes are grouped by replicate: the first n nodes belong to replicate 1, the next n to replicate 2, and so on. This allows different starting conditions per replicate if desired.

`ldata` remains unchanged: its data per node is shared across all replicates. Scheduled events are also shared—the same event schedule applies to each replicate.

Use this when you need multiple independent stochastic trajectories from the same model in a single simulation run. For identical starting conditions across replicates, simply repeat the same node pattern in `u0`.

`C_code` **C Source Code** (character vector). Optional C code for custom transition rates. If provided, it is compiled and loaded when `run()` is called.

Value

A SimInf_model object.

See Also

SIR, SEIR, SIS, SISe for examples of compartment model constructors that handle argument validation and matrix setup. mparse for creating custom models using a simple string syntax. SimInf_model for details on the class structure and slots. SimInf_events for details on the event schedule format.

SimInf_model-class *Class* SimInf_model

Description

The core class for storing the state, parameters, and results of a stochastic simulation in **SimInf**. This class holds the model definition (transition graphs, matrices), initial conditions, scheduled events, and the simulation output.

Slots

G **Dependency Graph** (sparse matrix, class dgCMatrix). Indicates which transition rates need to be updated after a state transition occurs. A non-zero entry G[i, j] means that transition rate i must be recalculated if transition j occurs. This optimizes performance by avoiding unnecessary updates. Dimensions: $N_t \times N_t$, where N_t is the number of transitions.

S **State Transition Matrix** (sparse matrix, class dgCMatrix). Defines the change in the state vector for each transition. Executing transition j adds the column S[, j] to the state vector of the affected node. Dimensions: $N_c \times N_t$, where N_c is the number of compartments.

U **Discrete State Result Matrix** (integer matrix). Contains the number of individuals in each compartment for every node at each time point in tspan.

- U[, j]: State at time tspan[j].
- Rows are ordered by node, then by compartment:
  - Rows 1:Nc: Node 1.
  - Rows (Nc+1):(2*Nc): Node 2.
  - ... and so on.

Dimensions: $N_n \times N_c \times$ length(tspan). *Note: If the model was run with sparse output, this slot is empty.*

U_sparse **Sparse Discrete State Result** (sparse matrix, class dgCMatrix). Contains the simulation results if the model was configured for sparse output. The layout is identical to U, but stored as a sparse matrix to save memory. *Note: Only one of* U *or* U_sparse *will contain data.*

V **Continuous State Result Matrix** (numeric matrix). Contains the values of continuous state variables (e.g., environmental pathogen load) for every node at each time point. Dimensions: $N_n \times N_{ld} \times$ length(tspan), where N_{ld} is the number of local data variables (continuous states). *Note: If sparse output was used, this slot is empty.*

`V_sparse` **Sparse Continuous State Result** (sparse matrix, class `dgCMatrix`). Contains the continuous state results if sparse output was enabled. Layout identical to `V`. *Note: Only one of* `V` *or* `V_sparse` *will contain data.*

`ldata` **Local Data Matrix** (numeric matrix). Parameters specific to each node (e.g., node-specific transmission rates). Column `ldata[, j]` contains the local data vector for node `j`. Passed to transition rate functions and post-step functions. Dimensions: $N_{ld} \times N_n$.

`gdata` **Global Data Vector** (numeric vector). Parameters common to all nodes (e.g., global recovery rate). Passed to transition rate functions and post-step functions.

`tspan` **Time Span** (numeric vector). Increasing time points where the state of each node is recorded.

`u0` **Initial State Matrix** (integer matrix). Initial number of individuals in each compartment for every node. Dimensions: $N_c \times N_n$.

`v0` **Initial Continuous State Matrix** (numeric matrix). Initial values for continuous state variables for every node. Dimensions: $N_{ld} \times N_n$.

`events` **Scheduled Events** (`SimInf_events`). Object containing the schedule of discrete events (e.g., movements, births).

`replicates` **Number of Replicates** (integer). Number of parallel replicates simulated for this model (used in filtering algorithms).

`C_code` **C Source Code** (character vector). Optional C code defining the model's transition rates. If non-empty, this code is written to a temporary file, compiled, and loaded when `run()` is called. Typically generated by `mparse`.

See Also

`SimInf_model` (constructor) for creating model objects. `SIR`, `SEIR`, `SIS`, `SISe` for compartment model constructors that automatically set up the required slots. `mparse` for creating custom models using a simple string syntax. `run` for executing the simulation. `trajectory`, `prevalence`, and `plot` for extracting and visualizing results. `SimInf_events` for details on the event schedule structure.

SimInf_pfilter-class *Class* SimInf_pfilter

Description

Storage class for the result of a bootstrap particle filter analysis. The class holds the model with a sampled trajectory attached, the number of particles used, the estimated log-likelihood, and the effective sample size (ESS) at each time-point.

Slots

`model` A `SimInf_model` object with one complete sampled trajectory attached. At the end of the filtering, a single particle is drawn from the ensemble at the final time-point, and its full state evolution across all time-points is reconstructed.

`n_particles` An integer with the number of particles that was used at each time-step.

`loglik` The estimated log-likelihood.

`ess` A numeric vector with the effective sample size (ESS) at each time-point. The effective sample size is computed as

$$\left(\sum_{i=1}^{N}(w_t^i)^2\right)^{-1},$$

where w_t^i is the normalized weight of particle i at time t.

See Also

`pfilter` for running a bootstrap particle filter and creating objects of this class, `logLik` for extracting the log-likelihood, `trajectory` for extracting the filtered trajectory, `prevalence` for computing prevalence from the filtered trajectory, `SimInf_model` for the underlying model class.

SimInf_pmcmc-class *Class* SimInf_pmcmc

Description

Class `SimInf_pmcmc`

Slots

`model` The `SimInf_model` object to estimate parameters in.

`priors` A `data.frame` defining the prior distributions for the parameters. It contains four columns:

- `parameter`: The name of the parameter in the model.
- `distribution`: The prior distribution type. Valid values are `"gamma"`, `"lognormal"`, `"normal"`, or `"uniform"`.
- `p1`: The first hyperparameter:
  - `"gamma"`: *shape*
  - `"lognormal"`: *meanlog* (mean on the log scale)
  - `"normal"`: *mean*
  - `"uniform"`: *lower* bound
- `p2`: The second hyperparameter:
  - `"gamma"`: *rate*
  - `"lognormal"`: *sdlog* (standard deviation on the log scale)
  - `"normal"`: *sd* (standard deviation)
  - `"uniform"`: *upper* bound

`target` Character vector (`"gdata"` or `"ldata"`) that determines if the ABC-SMC method estimates parameters in `model@gdata` (global data) or in `model@ldata` (local data).

`pars` An integer vector with the indices of the parameters in `target` that are being estimated.

`n_particles` An integer with the number of particles (> 1) to use in the bootstrap particle filter.

`data` A `data.frame` holding the time series data for the observation process.

`chain` A matrix where each row contains `logPost`, `logLik`, `logPrior`, `accept`, and the `parameters` for each iteration.

`covmat` A named numeric `(npars x npars)` matrix with covariances to use as initial proposal matrix.

`adaptmix` A numeric scalar specifying the mixing proportion for the adaptive proposal distribution.

`adaptive` An integer specifying when to start the adaptive update of the proposal distribution (iteration number).

See Also

`pmcmc` for the main PMCMC function, `continue_pmcmc` for continuing an existing PMCMC run.

`SIR` *Create an SIR model*

Description

Create an SIR model to be used by the simulation framework.

Usage

```
SIR(u0, tspan, events = NULL, beta = NULL, gamma = NULL)
```

Arguments

`u0`	A `data.frame` with the initial state in each node, i.e., the number of individuals in each compartment in each node when the simulation starts (see 'Details'). The parameter `u0` can also be an object that can be coerced to a `data.frame`, e.g., a named numeric vector will be coerced to a one row `data.frame`.
`tspan`	A vector (length >= 1) of increasing time points where the state of each node is to be returned. Can be either an `integer` or a `Date` vector. • If `integer`: Represents the specific time points (e.g., days, hours) at which to record the state. • If `Date`: Coerced to a numeric vector representing the **day of the year** (1–366) relative to the first date in the vector. The original `Date` objects are preserved as names for the numeric vector, facilitating time-series plotting.
`events`	a `data.frame` with the scheduled events, see `SimInf_model`.
`beta`	A numeric vector with the transmission rate from susceptible to infected. Each node can have a different beta value. The vector must have length 1 or `nrow(u0)`. If the vector has length 1 but the model contains more nodes, the beta value is repeated for all nodes.
`gamma`	A numeric vector with the recovery rate from infected to recovered. Each node can have a different gamma value. The vector must have length 1 or `nrow(u0)`. If the vector has length 1 but the model contains more nodes, the gamma value is repeated for all nodes.

Details

The SIR model is a compartmental model for infectious diseases that divides the population into three states: **S**usceptible, **I**nfected, and **R**ecovered. It assumes that individuals gain permanent immunity after recovery.

The model is defined by two state transitions:

$$S \xrightarrow{\beta SI/N} I$$

$$I \xrightarrow{\gamma I} R$$

where β is the transmission rate, γ is the recovery rate, and $N = S + I + R$ is the total population size in each node. Here, S, I, and R represent the number of susceptible, infected, and recovered individuals in that specific node.

The argument `u0` must be a `data.frame` with one row for each node with the following columns:

S The number of susceptible individuals in each node

I The number of infected individuals in each node

R The number of recovered individuals in each node

Value

A `SimInf_model` of class `SIR`

See Also

`SIR` for the class definition. `SEIR`, `SIS`, `SISe`, `SISe3` and `SISe_sp` for other predefined models. `mparse` for creating custom models. `run` for running the simulation. `trajectory`, `prevalence` and `plot` for post-processing and visualization.

Examples

```
## Create an SIR model object.
model <- SIR(
  u0 = data.frame(S = 99, I = 1, R = 0),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Run the SIR model with a fixed seed for reproducibility.
result <- run(model, seed = 22)

## Plot the distribution of susceptible, infected and recovered
## individuals.
plot(result)
```

SIR-class *Class SIR*

Description

Class to handle the SIR model. This class inherits from SimInf_model, meaning that SIR objects are fully compatible with all generic functions defined for SimInf_model, such as run, plot, trajectory, and prevalence.

Details

The SIR model is a compartmental model for infectious diseases that divides the population into three states: **S**usceptible, **I**nfected, and **R**ecovered. It assumes that individuals gain permanent immunity after recovery.

The model is defined by two state transitions:

$$S \xrightarrow{\beta SI/N} I$$

$$I \xrightarrow{\gamma I} R$$

where β is the transmission rate, γ is the recovery rate, and $N = S + I + R$ is the total population size in each node. Here, S, I, and R represent the number of susceptible, infected, and recovered individuals in that specific node.

See Also

SIR for creating an SIR model object, SimInf_model for the parent class definition, SEIR for a model including a latent period, and SIS for a model without immunity.

SIS *Create an SIS model*

Description

Create an SIS model to be used by the simulation framework.

Usage

```
SIS(u0, tspan, events = NULL, beta = NULL, gamma = NULL)
```

Arguments

u0	A data.frame with the initial state in each node, i.e., the number of individuals in each compartment in each node when the simulation starts (see 'Details'). The parameter u0 can also be an object that can be coerced to a data.frame, e.g., a named numeric vector will be coerced to a one row data.frame.
tspan	A vector (length >= 1) of increasing time points where the state of each node is to be returned. Can be either an integer or a Date vector. • If integer: Represents the specific time points (e.g., days, hours) at which to record the state. • If Date: Coerced to a numeric vector representing the **day of the year** (1–366) relative to the first date in the vector. The original Date objects are preserved as names for the numeric vector, facilitating time-series plotting.
events	a data.frame with the scheduled events, see SimInf_model.
beta	A numeric vector with the transmission rate from susceptible to infected. Each node can have a different beta value. The vector must have length 1 or nrow(u0). If the vector has length 1 but the model contains more nodes, the beta value is repeated for all nodes.
gamma	A numeric vector with the recovery rate from infected to recovered. Each node can have a different gamma value. The vector must have length 1 or nrow(u0). If the vector has length 1 but the model contains more nodes, the gamma value is repeated for all nodes.

Details

The SIS model is a compartmental model for infectious diseases where individuals do not gain permanent immunity after recovery. Instead, they return to the susceptible state. It divides the population into two states: **S**usceptible and **I**nfected.

The model is defined by two state transitions:

$$S \xrightarrow{\beta SI/N} I$$

$$I \xrightarrow{\gamma I} S$$

where β is the transmission rate, γ is the recovery rate, and $N = S + I$ is the total population size in each node. Here, S and I represent the number of susceptible and infected individuals in that specific node.

The argument u0 must be a data.frame with one row for each node with the following columns:

S The number of susceptible individuals in each node

I The number of infected individuals in each node

Value

A SimInf_model of class SIS

See Also

SIS for the class definition. SIR, SEIR, SISe, SISe3 and SISe_sp for other predefined models. mparse for creating custom models. run for running the simulation. trajectory, prevalence and plot for post-processing and visualization.

Examples

```
## Create an SIS model object.
model <- SIS(
  u0 = data.frame(S = 99, I = 1),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Run the SIS model with a fixed seed for reproducibility.
result <- run(model, seed = 22)

## Plot the distribution of susceptible and infected individuals.
plot(result)
```

SIS-class *Class SIS*

Description

Class to handle the SIS model. This class inherits from SimInf_model, meaning that SIS objects are fully compatible with all generic functions defined for SimInf_model, such as run, plot, trajectory, and prevalence.

Details

The SIS model is a compartmental model for infectious diseases where individuals do not gain permanent immunity after recovery. Instead, they return to the susceptible state. It divides the population into two states: **S**usceptible and **I**nfected.

The model is defined by two state transitions:

$$S \xrightarrow{\beta SI/N} I$$

$$I \xrightarrow{\gamma I} S$$

where β is the transmission rate, γ is the recovery rate, and $N = S + I$ is the total population size in each node. Here, S and I represent the number of susceptible and infected individuals in that specific node.

See Also

SIS for creating an SIS model object, SimInf_model for the parent class definition, SIR for a model with permanent immunity, and SEIR for a model including a latent period.

SISe *Create an SISe model*

Description

Create an SISE model to be used by the simulation framework.

Usage

```
SISe(
  u0,
  tspan,
  events = NULL,
  phi = NULL,
  upsilon = NULL,
  gamma = NULL,
  alpha = NULL,
  beta_t1 = NULL,
  beta_t2 = NULL,
  beta_t3 = NULL,
  beta_t4 = NULL,
  end_t1 = NULL,
  end_t2 = NULL,
  end_t3 = NULL,
  end_t4 = NULL,
  epsilon = NULL
)
```

Arguments

u0	A data.frame with the initial state in each node, i.e., the number of individuals in each compartment in each node when the simulation starts (see 'Details'). The parameter u0 can also be an object that can be coerced to a data.frame, e.g., a named numeric vector will be coerced to a one row data.frame.
tspan	A vector (length >= 1) of increasing time points where the state of each node is to be returned. Can be either an integer or a Date vector. • If integer: Represents the specific time points (e.g., days, hours) at which to record the state. • If Date: Coerced to a numeric vector representing the **day of the year** (1–366) relative to the first date in the vector. The original Date objects are preserved as names for the numeric vector, facilitating time-series plotting.
events	a data.frame with the scheduled events, see SimInf_model.
phi	A numeric vector with the initial environmental infectious pressure in each node. Will be repeated to the length of nrow(u0). Default is NULL which gives 0 in each node.

upsilon	Indirect transmission rate of the environmental infectious pressure
gamma	A numeric vector with the recovery rate from infected to susceptible. Each node can have a different gamma value. The vector must have length 1 or nrow(u0). If the vector has length 1 but the model contains more nodes, the value is repeated for all nodes.
alpha	Shedding rate of the pathogen to the environment per infected individual.
beta_t1	The decay of the environmental infectious pressure in interval 1.
beta_t2	The decay of the environmental infectious pressure in interval 2.
beta_t3	The decay of the environmental infectious pressure in interval 3.
beta_t4	The decay of the environmental infectious pressure in interval 4.
end_t1	vector with the non-inclusive day of the year that ends interval 1 in each node. Will be repeated to the length of nrow(u0).
end_t2	vector with the non-inclusive day of the year that ends interval 2 in each node. Will be repeated to the length of nrow(u0).
end_t3	vector with the non-inclusive day of the year that ends interval 3 in each node. Will be repeated to the length of nrow(u0).
end_t4	vector with the non-inclusive day of the year that ends interval 4 in each node. Will be repeated to the length of nrow(u0).
epsilon	The background environmental infectious pressure

Details

The SISe model contains two compartments: **S**usceptible (S) and **I**nfected (I). Additionally, it includes a continuous **environmental** compartment (φ) to model the shedding of a pathogen to the environment.

The model is defined by two state transitions:

$$S \xrightarrow{\upsilon\varphi S} I$$
$$I \xrightarrow{\gamma I} S$$

where the transition rate from susceptible to infected is proportional to the environmental contamination φ and the transmission rate υ. The recovery rate γ moves individuals from infected back to susceptible.

The environmental infectious pressure $\varphi(t)$ in each node evolves according to:

$$\frac{d\varphi(t)}{dt} = \frac{\alpha I(t)}{N(t)} - \beta(t)\varphi(t) + \epsilon$$

where:

- α is the shedding rate per infected individual.
- $N(t) = S + I$ is the total population size in the node.
- $\beta(t)$ is the seasonal decay/removal rate, which varies throughout the year.
- ϵ is the background infectious pressure.

The environmental pressure is evolved using the Euler forward method and saved at time points in `tspan`.

Seasonal Decay ($\beta(t)$): The decay rate $\beta(t)$ is piecewise constant, defined by four intervals determined by the parameters `end_t1`, `end_t2`, `end_t3`, and `end_t4` (days of the year, where `0 <= day < 365`). The year is divided into four intervals based on the sorted order of these endpoints. The interval that wraps around the year boundary (from the last endpoint to day 365, then from day 0 to the first endpoint) receives the same rate as the interval preceding the first endpoint. Three orderings are supported:

Case 1: `end_t1 < end_t2 < end_t3 < end_t4`

- Interval 1: `[0, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1 (wrap-around): `[end_t4, 365)` with rate `beta_t1`

Case 2: `end_t3 < end_t4 < end_t1 < end_t2`

- Interval 3: `[0, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3 (wrap-around): `[end_t2, 365)` with rate `beta_t3`

Case 3: `end_t4 < end_t1 < end_t2 < end_t3`

- Interval 4: `[0, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4 (wrap-around): `[end_t3, 365)` with rate `beta_t4`

These different orderings allow the model to handle seasonal patterns where, for example, a winter peak crosses the year boundary.

The argument `u0` must be a `data.frame` with one row for each node with the following columns:

S The number of susceptible in each node

I The number of infected in each node

Value

`SISe`

See Also

`SISe` for the class definition. `SIR`, `SEIR`, `SIS`, `SISe3` and `SISe_sp` for other predefined models. `mparse` for creating custom models. `run` for running the simulation. `trajectory`, `prevalence` and `plot` for post-processing and visualization.

SISe-class *Class SISe*

Description

Class to handle the SISE model. This class inherits from SimInf_model, meaning that SISE objects are fully compatible with all generic functions defined for SimInf_model, such as run, plot, trajectory, and prevalence.

Details

The SISe model contains two compartments: **S**usceptible (S) and **I**nfected (I). Additionally, it includes a continuous **environmental** compartment (φ) to model the shedding of a pathogen to the environment.

The model is defined by two state transitions:

$$S \xrightarrow{\upsilon\varphi S} I$$
$$I \xrightarrow{\gamma I} S$$

where the transition rate from susceptible to infected is proportional to the environmental contamination φ and the transmission rate υ. The recovery rate γ moves individuals from infected back to susceptible.

The environmental infectious pressure $\varphi(t)$ in each node evolves according to:

$$\frac{d\varphi(t)}{dt} = \frac{\alpha I(t)}{N(t)} - \beta(t)\varphi(t) + \epsilon$$

where:

- α is the shedding rate per infected individual.
- $N(t) = S + I$ is the total population size in the node.
- $\beta(t)$ is the seasonal decay/removal rate, which varies throughout the year.
- ϵ is the background infectious pressure.

The environmental pressure is evolved using the Euler forward method and saved at time points in tspan.

Seasonal Decay ($\beta(t)$): The decay rate $\beta(t)$ is piecewise constant, defined by four intervals determined by the parameters end_t1, end_t2, end_t3, and end_t4 (days of the year, where 0 <= day < 365). The year is divided into four intervals based on the sorted order of these endpoints. The interval that wraps around the year boundary (from the last endpoint to day 365, then from day 0 to the first endpoint) receives the same rate as the interval preceding the first endpoint. Three orderings are supported:

Case 1: end_t1 < end_t2 < end_t3 < end_t4

- Interval 1: [0, end_t1) with rate beta_t1

- Interval 2: [end_t1, end_t2) with rate beta_t2
- Interval 3: [end_t2, end_t3) with rate beta_t3
- Interval 4: [end_t3, end_t4) with rate beta_t4
- Interval 1 (wrap-around): [end_t4, 365) with rate beta_t1

Case 2: end_t3 < end_t4 < end_t1 < end_t2

- Interval 3: [0, end_t3) with rate beta_t3
- Interval 4: [end_t3, end_t4) with rate beta_t4
- Interval 1: [end_t4, end_t1) with rate beta_t1
- Interval 2: [end_t1, end_t2) with rate beta_t2
- Interval 3 (wrap-around): [end_t2, 365) with rate beta_t3

Case 3: end_t4 < end_t1 < end_t2 < end_t3

- Interval 4: [0, end_t4) with rate beta_t4
- Interval 1: [end_t4, end_t1) with rate beta_t1
- Interval 2: [end_t1, end_t2) with rate beta_t2
- Interval 3: [end_t2, end_t3) with rate beta_t3
- Interval 4 (wrap-around): [end_t3, 365) with rate beta_t4

These different orderings allow the model to handle seasonal patterns where, for example, a winter peak crosses the year boundary.

See Also

SISe for creating an SISE model object and SimInf_model for the parent class definition.

SISe3 *Create a* SISe3 *model*

Description

Create a SISE3 model to be used by the simulation framework.

Usage

```
SISe3(
  u0,
  tspan,
  events = NULL,
  phi = NULL,
  upsilon_1 = NULL,
  upsilon_2 = NULL,
  upsilon_3 = NULL,
  gamma_1 = NULL,
```

```
  gamma_2 = NULL,
  gamma_3 = NULL,
  alpha = NULL,
  beta_t1 = NULL,
  beta_t2 = NULL,
  beta_t3 = NULL,
  beta_t4 = NULL,
  end_t1 = NULL,
  end_t2 = NULL,
  end_t3 = NULL,
  end_t4 = NULL,
  epsilon = NULL
)
```

Arguments

u0	A data.frame with the initial state in each node, i.e., the number of individuals in each compartment in each node when the simulation starts (see 'Details'). The parameter u0 can also be an object that can be coerced to a data.frame, e.g., a named numeric vector will be coerced to a one row data.frame.
tspan	A vector (length >= 1) of increasing time points where the state of each node is to be returned. Can be either an integer or a Date vector. • If integer: Represents the specific time points (e.g., days, hours) at which to record the state. • If Date: Coerced to a numeric vector representing the **day of the year** (1–366) relative to the first date in the vector. The original Date objects are preserved as names for the numeric vector, facilitating time-series plotting.
events	a data.frame with the scheduled events, see SimInf_model.
phi	A numeric vector with the initial environmental infectious pressure in each node. Will be repeated to the length of nrow(u0). Default is NULL which gives 0 in each node.
upsilon_1	Indirect transmission rate of the environmental infectious pressure in age category 1
upsilon_2	Indirect transmission rate of the environmental infectious pressure in age category 2
upsilon_3	Indirect transmission rate of the environmental infectious pressure in age category 3
gamma_1	The recovery rate from infected to susceptible for age category 1
gamma_2	The recovery rate from infected to susceptible for age category 2
gamma_3	The recovery rate from infected to susceptible for age category 3
alpha	Shedding rate of the pathogen to the environment per infected individual.
beta_t1	The decay of the environmental infectious pressure in interval 1.
beta_t2	The decay of the environmental infectious pressure in interval 2.
beta_t3	The decay of the environmental infectious pressure in interval 3.

beta_t4	The decay of the environmental infectious pressure in interval 4.
end_t1	vector with the non-inclusive day of the year that ends interval 1 in each node. Will be repeated to the length of nrow(u0).
end_t2	vector with the non-inclusive day of the year that ends interval 2 in each node. Will be repeated to the length of nrow(u0).
end_t3	vector with the non-inclusive day of the year that ends interval 3 in each node. Will be repeated to the length of nrow(u0).
end_t4	vector with the non-inclusive day of the year that ends interval 4 in each node. Will be repeated to the length of nrow(u0).
epsilon	The background environmental infectious pressure

Details

The SISe3 model contains two compartments in three age categories: **S**usceptible (S_1, S_2, S_3) and **I**nfected (I_1, I_2, I_3). Additionally, it includes a continuous **environmental** compartment (φ) to model the shedding of a pathogen to the environment.

The model is defined by six state transitions:

$$S_1 \xrightarrow{\upsilon_1 \varphi S_1} I_1$$

$$I_1 \xrightarrow{\gamma_1 I_1} S_1$$

$$S_2 \xrightarrow{\upsilon_2 \varphi S_2} I_2$$

$$I_2 \xrightarrow{\gamma_2 I_2} S_2$$

$$S_3 \xrightarrow{\upsilon_3 \varphi S_3} I_3$$

$$I_3 \xrightarrow{\gamma_3 I_3} S_3$$

where the transition rate from susceptible to infected in age category k is proportional to the environmental contamination φ and the transmission rate υ_k. The recovery rate γ_k moves individuals from infected back to susceptible.

The environmental infectious pressure $\varphi(t)$ in each node evolves according to:

$$\frac{d\varphi(t)}{dt} = \frac{\alpha \left(I_1(t) + I_2(t) + I_3(t)\right)}{N(t)} - \beta(t)\varphi(t) + \epsilon$$

where:

- α is the shedding rate per infected individual.
- $N(t) = S_1 + S_2 + S_3 + I_1 + I_2 + I_3$ is the total population size in the node.
- $\beta(t)$ is the seasonal decay/removal rate, which varies throughout the year.
- ϵ is the background infectious pressure.

The environmental infectious pressure $\varphi(t)$ is evolved using the Euler forward method and saved at time points in `tspan`.

Seasonal Decay ($\beta(t)$): The decay rate $\beta(t)$ is piecewise constant, defined by four intervals determined by the parameters `end_t1`, `end_t2`, `end_t3`, and `end_t4` (days of the year, where `0 <= day < 365`). The year is divided into four intervals based on the sorted order of these endpoints. The interval that wraps around the year boundary (from the last endpoint to day 365, then from day 0 to the first endpoint) receives the same rate as the interval preceding the first endpoint. Three orderings are supported:

Case 1: `end_t1 < end_t2 < end_t3 < end_t4`

- Interval 1: `[0, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1 (wrap-around): `[end_t4, 365)` with rate `beta_t1`

Case 2: `end_t3 < end_t4 < end_t1 < end_t2`

- Interval 3: `[0, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3 (wrap-around): `[end_t2, 365)` with rate `beta_t3`

Case 3: `end_t4 < end_t1 < end_t2 < end_t3`

- Interval 4: `[0, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4 (wrap-around): `[end_t3, 365)` with rate `beta_t4`

These different orderings allow the model to handle seasonal patterns where, for example, a winter peak crosses the year boundary.

The argument `u0` must be a `data.frame` with one row for each node with the following columns:

S_1 The number of susceptible in age category 1

I_1 The number of infected in age category 1

S_2 The number of susceptible in age category 2

I_2 The number of infected in age category 2

S_3 The number of susceptible in age category 3

I_3 The number of infected in age category 3

Value

`SISe3`

See Also

SISe3 for the class definition. SIR, SEIR, SIS, SISe and SISe_sp for other predefined models. mparse for creating custom models. run for running the simulation. trajectory, prevalence and plot for post-processing and visualization.

SISe3-class *Class SISe3*

Description

Class to handle the SISE3 model. This class inherits from SimInf_model, meaning that SISE3 objects are fully compatible with all generic functions defined for SimInf_model, such as run, plot, trajectory, and prevalence.

Details

The SISe3 model contains two compartments in three age categories: **S**usceptible (S_1, S_2, S_3) and **I**nfected (I_1, I_2, I_3). Additionally, it includes a continuous **environmental** compartment (φ) to model the shedding of a pathogen to the environment.

The model is defined by six state transitions:

$$S_1 \xrightarrow{\upsilon_1 \varphi S_1} I_1$$
$$I_1 \xrightarrow{\gamma_1 I_1} S_1$$
$$S_2 \xrightarrow{\upsilon_2 \varphi S_2} I_2$$
$$I_2 \xrightarrow{\gamma_2 I_2} S_2$$
$$S_3 \xrightarrow{\upsilon_3 \varphi S_3} I_3$$
$$I_3 \xrightarrow{\gamma_3 I_3} S_3$$

where the transition rate from susceptible to infected in age category k is proportional to the environmental contamination φ and the transmission rate υ_k. The recovery rate γ_k moves individuals from infected back to susceptible.

The environmental infectious pressure $\varphi(t)$ in each node evolves according to:

$$\frac{d\varphi(t)}{dt} = \frac{\alpha \left(I_1(t) + I_2(t) + I_3(t)\right)}{N(t)} - \beta(t)\varphi(t) + \epsilon$$

where:

- α is the shedding rate per infected individual.
- $N(t) = S_1 + S_2 + S_3 + I_1 + I_2 + I_3$ is the total population size in the node.
- $\beta(t)$ is the seasonal decay/removal rate, which varies throughout the year.
- ϵ is the background infectious pressure.

The environmental infectious pressure $\varphi(t)$ is evolved using the Euler forward method and saved at time points in `tspan`.

Seasonal Decay ($\beta(t)$): The decay rate $\beta(t)$ is piecewise constant, defined by four intervals determined by the parameters `end_t1`, `end_t2`, `end_t3`, and `end_t4` (days of the year, where `0 <= day < 365`). The year is divided into four intervals based on the sorted order of these endpoints. The interval that wraps around the year boundary (from the last endpoint to day 365, then from day 0 to the first endpoint) receives the same rate as the interval preceding the first endpoint. Three orderings are supported:

Case 1: `end_t1 < end_t2 < end_t3 < end_t4`

- Interval 1: `[0, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1 (wrap-around): `[end_t4, 365)` with rate `beta_t1`

Case 2: `end_t3 < end_t4 < end_t1 < end_t2`

- Interval 3: `[0, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3 (wrap-around): `[end_t2, 365)` with rate `beta_t3`

Case 3: `end_t4 < end_t1 < end_t2 < end_t3`

- Interval 4: `[0, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4 (wrap-around): `[end_t3, 365)` with rate `beta_t4`

These different orderings allow the model to handle seasonal patterns where, for example, a winter peak crosses the year boundary.

See Also

`SISe3` for creating an SISE3 model object and `SimInf_model` for the parent class definition.

SISe3_sp *Create an SISe3_sp model*

Description

Create an SISe3_sp model to be used by the simulation framework.

Usage

```
SISe3_sp(
  u0,
  tspan,
  events = NULL,
  phi = NULL,
  upsilon_1 = NULL,
  upsilon_2 = NULL,
  upsilon_3 = NULL,
  gamma_1 = NULL,
  gamma_2 = NULL,
  gamma_3 = NULL,
  alpha = NULL,
  beta_t1 = NULL,
  beta_t2 = NULL,
  beta_t3 = NULL,
  beta_t4 = NULL,
  end_t1 = NULL,
  end_t2 = NULL,
  end_t3 = NULL,
  end_t4 = NULL,
  distance = NULL,
  coupling = NULL
)
```

Arguments

u0	A data.frame with the initial state in each node, i.e., the number of individuals in each compartment in each node when the simulation starts (see 'Details'). The parameter u0 can also be an object that can be coerced to a data.frame, e.g., a named numeric vector will be coerced to a one row data.frame.
tspan	A vector (length >= 1) of increasing time points where the state of each node is to be returned. Can be either an integer or a Date vector. • If integer: Represents the specific time points (e.g., days, hours) at which to record the state. • If Date: Coerced to a numeric vector representing the **day of the year** (1–366) relative to the first date in the vector. The original Date objects are preserved as names for the numeric vector, facilitating time-series plotting.

events	a data.frame with the scheduled events, see SimInf_model.
phi	A numeric vector with the initial environmental infectious pressure in each node. Will be repeated to the length of nrow(u0). Default is NULL which gives 0 in each node.
upsilon_1	Indirect transmission rate of the environmental infectious pressure in age category 1
upsilon_2	Indirect transmission rate of the environmental infectious pressure in age category 2
upsilon_3	Indirect transmission rate of the environmental infectious pressure in age category 3
gamma_1	The recovery rate from infected to susceptible for age category 1
gamma_2	The recovery rate from infected to susceptible for age category 2
gamma_3	The recovery rate from infected to susceptible for age category 3
alpha	Shedding rate of the pathogen to the environment per infected individual.
beta_t1	The decay of the environmental infectious pressure in interval 1.
beta_t2	The decay of the environmental infectious pressure in interval 2.
beta_t3	The decay of the environmental infectious pressure in interval 3.
beta_t4	The decay of the environmental infectious pressure in interval 4.
end_t1	vector with the non-inclusive day of the year that ends interval 1 in each node. Will be repeated to the length of nrow(u0).
end_t2	vector with the non-inclusive day of the year that ends interval 2 in each node. Will be repeated to the length of nrow(u0).
end_t3	vector with the non-inclusive day of the year that ends interval 3 in each node. Will be repeated to the length of nrow(u0).
end_t4	vector with the non-inclusive day of the year that ends interval 4 in each node. Will be repeated to the length of nrow(u0).
distance	The distance matrix between neighboring nodes
coupling	The coupling between neighboring nodes

Details

The SISe3_sp model contains two compartments in three age categories: **S**usceptible (S_1, S_2, S_3) and **I**nfected (I_1, I_2, I_3). Additionally, it includes a continuous **environmental** compartment (φ) to model shedding of a pathogen to the environment. Moreover, it includes a spatial coupling of the environmental contamination among proximal nodes to capture between-node spread unrelated to moving infected individuals.

The model is defined by six state transitions:

$$S_1 \xrightarrow{\upsilon_1 \varphi S_1} I_1$$

$$I_1 \xrightarrow{\gamma_1 I_1} S_1$$

$$S_2 \xrightarrow{\upsilon_2 \varphi S_2} I_2$$

$$I_2 \xrightarrow{\gamma_2 I_2} S_2$$
$$S_3 \xrightarrow{\upsilon_3 \varphi S_3} I_3$$
$$I_3 \xrightarrow{\gamma_3 I_3} S_3$$

where the transition rate from susceptible to infected in age category k is proportional to the environmental contamination φ and the transmission rate υ_k. The recovery rate γ_k moves individuals from infected back to susceptible.

The environmental infectious pressure $\varphi(t)$ in each node evolves according to:

$$\frac{d\varphi_i(t)}{dt} = \frac{\alpha\left(I_{i,1}(t) + I_{i,2}(t) + I_{i,3}(t)\right)}{N_i(t)} + \sum_k \frac{\varphi_k(t)N_k(t) - \varphi_i(t)N_i(t)}{N_i(t)} \cdot \frac{D}{d_{ik}} - \beta(t)\varphi_i(t)$$

where:

- α is the shedding rate per infected individual.
- $N(t) = S_1 + S_2 + S_3 + I_1 + I_2 + I_3$ is the total population size in the node.
- $\beta(t)$ is the seasonal decay/removal rate, which varies throughout the year.
- the last term is the spatial coupling among proximal nodes. D is the rate of the local spread and d_{ik} the distance between holdings i and k.

The environmental infectious pressure $\varphi(t)$ is evolved using the Euler forward method and saved at time points in `tspan`.

Seasonal Decay ($\beta(t)$): The decay rate $\beta(t)$ is piecewise constant, defined by four intervals determined by the parameters `end_t1`, `end_t2`, `end_t3`, and `end_t4` (days of the year, where `0 <= day < 365`). The year is divided into four intervals based on the sorted order of these endpoints. The interval that wraps around the year boundary (from the last endpoint to day 365, then from day 0 to the first endpoint) receives the same rate as the interval preceding the first endpoint. Three orderings are supported:

Case 1: `end_t1 < end_t2 < end_t3 < end_t4`

- Interval 1: `[0, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1 (wrap-around): `[end_t4, 365)` with rate `beta_t1`

Case 2: `end_t3 < end_t4 < end_t1 < end_t2`

- Interval 3: `[0, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3 (wrap-around): `[end_t2, 365)` with rate `beta_t3`

Case 3: `end_t4 < end_t1 < end_t2 < end_t3`

- Interval 4: `[0, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4 (wrap-around): `[end_t3, 365)` with rate `beta_t4`

These different orderings allow the model to handle seasonal patterns where, for example, a winter peak crosses the year boundary.

The argument `u0` must be a `data.frame` with one row for each node with the following columns:

S_1 The number of susceptible in age category 1

I_1 The number of infected in age category 1

S_2 The number of susceptible in age category 2

I_2 The number of infected in age category 2

S_3 The number of susceptible in age category 3

I_3 The number of infected in age category 3

Value

`SISe3_sp`

See Also

`SISe3_sp` for the class definition. `SIR`, `SEIR`, `SIS`, `SISe3` and `SISe_sp` for other predefined models. `mparse` for creating custom models. `run` for running the simulation. `trajectory`, `prevalence` and `plot` for post-processing and visualization.

`SISe3_sp-class` *Class SISe3_sp*

Description

Class to handle the SISE3_SP model. This class inherits from `SimInf_model`, meaning that SISE3_SP objects are fully compatible with all generic functions defined for `SimInf_model`, such as `run`, `plot`, `trajectory`, and `prevalence`.

Details

The `SISe3_sp` model contains two compartments in three age categories: **S**usceptible (S_1, S_2, S_3) and **I**nfected (I_1, I_2, I_3). Additionally, it includes a continuous **environmental** compartment (φ) to model shedding of a pathogen to the environment. Moreover, it includes a spatial coupling of the environmental contamination among proximal nodes to capture between-node spread unrelated to moving infected individuals.

The model is defined by six state transitions:

$$S_1 \xrightarrow{\upsilon_1 \varphi S_1} I_1$$
$$I_1 \xrightarrow{\gamma_1 I_1} S_1$$
$$S_2 \xrightarrow{\upsilon_2 \varphi S_2} I_2$$
$$I_2 \xrightarrow{\gamma_2 I_2} S_2$$
$$S_3 \xrightarrow{\upsilon_3 \varphi S_3} I_3$$
$$I_3 \xrightarrow{\gamma_3 I_3} S_3$$

where the transition rate from susceptible to infected in age category k is proportional to the environmental contamination φ and the transmission rate υ_k. The recovery rate γ_k moves individuals from infected back to susceptible.

The environmental infectious pressure $\varphi(t)$ in each node evolves according to:

$$\frac{d\varphi_i(t)}{dt} = \frac{\alpha \left(I_{i,1}(t) + I_{i,2}(t) + I_{i,3}(t)\right)}{N_i(t)} + \sum_k \frac{\varphi_k(t) N_k(t) - \varphi_i(t) N_i(t)}{N_i(t)} \cdot \frac{D}{d_{ik}} - \beta(t)\varphi_i(t)$$

where:

- α is the shedding rate per infected individual.
- $N(t) = S_1 + S_2 + S_3 + I_1 + I_2 + I_3$ is the total population size in the node.
- $\beta(t)$ is the seasonal decay/removal rate, which varies throughout the year.
- the last term is the spatial coupling among proximal nodes. D is the rate of the local spread and d_{ik} the distance between holdings i and k.

The environmental infectious pressure $\varphi(t)$ is evolved using the Euler forward method and saved at time points in `tspan`.

Seasonal Decay ($\beta(t)$): The decay rate $\beta(t)$ is piecewise constant, defined by four intervals determined by the parameters `end_t1`, `end_t2`, `end_t3`, and `end_t4` (days of the year, where `0 <= day < 365`). The year is divided into four intervals based on the sorted order of these endpoints. The interval that wraps around the year boundary (from the last endpoint to day 365, then from day 0 to the first endpoint) receives the same rate as the interval preceding the first endpoint. Three orderings are supported:

Case 1: `end_t1 < end_t2 < end_t3 < end_t4`

- Interval 1: `[0, end_t1)` with rate `beta_t1`

- Interval 2: [end_t1, end_t2) with rate beta_t2
- Interval 3: [end_t2, end_t3) with rate beta_t3
- Interval 4: [end_t3, end_t4) with rate beta_t4
- Interval 1 (wrap-around): [end_t4, 365) with rate beta_t1

Case 2: end_t3 < end_t4 < end_t1 < end_t2

- Interval 3: [0, end_t3) with rate beta_t3
- Interval 4: [end_t3, end_t4) with rate beta_t4
- Interval 1: [end_t4, end_t1) with rate beta_t1
- Interval 2: [end_t1, end_t2) with rate beta_t2
- Interval 3 (wrap-around): [end_t2, 365) with rate beta_t3

Case 3: end_t4 < end_t1 < end_t2 < end_t3

- Interval 4: [0, end_t4) with rate beta_t4
- Interval 1: [end_t4, end_t1) with rate beta_t1
- Interval 2: [end_t1, end_t2) with rate beta_t2
- Interval 3: [end_t2, end_t3) with rate beta_t3
- Interval 4 (wrap-around): [end_t3, 365) with rate beta_t4

These different orderings allow the model to handle seasonal patterns where, for example, a winter peak crosses the year boundary.

See Also

SISe3_sp for creating an SISE3_SP model object and SimInf_model for the parent class definition.

SISe_sp *Create an SISe_sp model*

Description

Create a SISe_sp model to be used by the simulation framework.

Usage

```
SISe_sp(
  u0,
  tspan,
  events = NULL,
  phi = NULL,
  upsilon = NULL,
  gamma = NULL,
  alpha = NULL,
  beta_t1 = NULL,
```

```
  beta_t2 = NULL,
  beta_t3 = NULL,
  beta_t4 = NULL,
  end_t1 = NULL,
  end_t2 = NULL,
  end_t3 = NULL,
  end_t4 = NULL,
  coupling = NULL,
  distance = NULL
)
```

Arguments

u0	A data.frame with the initial state in each node, i.e., the number of individuals in each compartment in each node when the simulation starts (see 'Details'). The parameter u0 can also be an object that can be coerced to a data.frame, e.g., a named numeric vector will be coerced to a one row data.frame.
tspan	A vector (length >= 1) of increasing time points where the state of each node is to be returned. Can be either an integer or a Date vector. • If integer: Represents the specific time points (e.g., days, hours) at which to record the state. • If Date: Coerced to a numeric vector representing the **day of the year** (1–366) relative to the first date in the vector. The original Date objects are preserved as names for the numeric vector, facilitating time-series plotting.
events	a data.frame with the scheduled events, see SimInf_model.
phi	A numeric vector with the initial environmental infectious pressure in each node. Will be repeated to the length of nrow(u0). Default is NULL which gives 0 in each node.
upsilon	Indirect transmission rate of the environmental infectious pressure
gamma	A numeric vector with the recovery rate from infected to susceptible. Each node can have a different gamma value. The vector must have length 1 or nrow(u0). If the vector has length 1 but the model contains more nodes, the value is repeated for all nodes.
alpha	Shedding rate of the pathogen to the environment per infected individual.
beta_t1	The decay of the environmental infectious pressure in interval 1.
beta_t2	The decay of the environmental infectious pressure in interval 2.
beta_t3	The decay of the environmental infectious pressure in interval 3.
beta_t4	The decay of the environmental infectious pressure in interval 4.
end_t1	vector with the non-inclusive day of the year that ends interval 1 in each node. Will be repeated to the length of nrow(u0).
end_t2	vector with the non-inclusive day of the year that ends interval 2 in each node. Will be repeated to the length of nrow(u0).
end_t3	vector with the non-inclusive day of the year that ends interval 3 in each node. Will be repeated to the length of nrow(u0).

end_t4	vector with the non-inclusive day of the year that ends interval 4 in each node. Will be repeated to the length of nrow(u0).
coupling	The coupling between neighboring nodes
distance	The distance matrix between neighboring nodes

Details

The SISe_sp model contains two compartments; number of susceptible (S) and number of infectious (I). Additionally, it contains an environmental compartment to model shedding of a pathogen to the environment. Moreover, it also includes a spatial coupling of the environmental contamination among proximal nodes to capture between-node spread unrelated to moving infected individuals. Consequently, the model has two state transitions,

$$S \xrightarrow{\upsilon\varphi S} I$$

$$I \xrightarrow{\gamma I} S$$

where the transition rate per unit of time from susceptible to infected is proportional to the concentration of the environmental contamination φ in each node. Moreover, the transition rate from infected to susceptible is the recovery rate γ, measured per individual and per unit of time. Finally, the environmental infectious pressure in each node is evolved by,

$$\frac{d\varphi_i(t)}{dt} = \frac{\alpha I_i(t)}{N_i(t)} + \sum_k \frac{\varphi_k(t)N_k(t) - \varphi_i(t)N_i(t)}{N_i(t)} \cdot \frac{D}{d_{ik}} - \beta(t)\varphi_i(t)$$

where α is the average shedding rate of the pathogen to the environment per infected individual and $N = S + I$ the size of the node. Next comes the spatial coupling among proximal nodes, where D is the rate of the local spread and d_{ik} the distance between holdings i and k. The seasonal decay and removal of the pathogen is captured by $\beta(t)$. The environmental infectious pressure $\varphi(t)$ in each node is evolved each time unit by the Euler forward method. The value of $\varphi(t)$ is saved at the time-points specified in tspan.

Seasonal Decay ($\beta(t)$): The decay rate $\beta(t)$ is piecewise constant, defined by four intervals determined by the parameters end_t1, end_t2, end_t3, and end_t4 (days of the year, where 0 <= day < 365). The year is divided into four intervals based on the sorted order of these endpoints. The interval that wraps around the year boundary (from the last endpoint to day 365, then from day 0 to the first endpoint) receives the same rate as the interval preceding the first endpoint. Three orderings are supported:

Case 1: end_t1 < end_t2 < end_t3 < end_t4

- Interval 1: [0, end_t1) with rate beta_t1
- Interval 2: [end_t1, end_t2) with rate beta_t2
- Interval 3: [end_t2, end_t3) with rate beta_t3
- Interval 4: [end_t3, end_t4) with rate beta_t4
- Interval 1 (wrap-around): [end_t4, 365) with rate beta_t1

Case 2: end_t3 < end_t4 < end_t1 < end_t2

- Interval 3: [0, end_t3) with rate beta_t3
- Interval 4: [end_t3, end_t4) with rate beta_t4
- Interval 1: [end_t4, end_t1) with rate beta_t1
- Interval 2: [end_t1, end_t2) with rate beta_t2
- Interval 3 (wrap-around): [end_t2, 365) with rate beta_t3

Case 3: end_t4 < end_t1 < end_t2 < end_t3

- Interval 4: [0, end_t4) with rate beta_t4
- Interval 1: [end_t4, end_t1) with rate beta_t1
- Interval 2: [end_t1, end_t2) with rate beta_t2
- Interval 3: [end_t2, end_t3) with rate beta_t3
- Interval 4 (wrap-around): [end_t3, 365) with rate beta_t4

These different orderings allow the model to handle seasonal patterns where, for example, a winter peak crosses the year boundary.

The argument u0 must be a data.frame with one row for each node with the following columns:

S The number of susceptible

I The number of infected

Value

SISe_sp

See Also

SISe_sp for the class definition. SIR, SEIR, SIS, SISe and SISe3_sp for other predefined models. mparse for creating custom models. run for running the simulation. trajectory, prevalence and plot for post-processing and visualization.

SISe_sp-class *Class SISe_sp*

Description

Class to handle the SISE_SP model. This class inherits from SimInf_model, meaning that SISE_SP objects are fully compatible with all generic functions defined for SimInf_model, such as run, plot, trajectory, and prevalence.

Details

The `SISe_sp` model contains two compartments; number of susceptible (S) and number of infectious (I). Additionally, it contains an environmental compartment to model shedding of a pathogen to the environment. Moreover, it also includes a spatial coupling of the environmental contamination among proximal nodes to capture between-node spread unrelated to moving infected individuals. Consequently, the model has two state transitions,

$$S \xrightarrow{\upsilon\varphi S} I$$

$$I \xrightarrow{\gamma I} S$$

where the transition rate per unit of time from susceptible to infected is proportional to the concentration of the environmental contamination φ in each node. Moreover, the transition rate from infected to susceptible is the recovery rate γ, measured per individual and per unit of time. Finally, the environmental infectious pressure in each node is evolved by,

$$\frac{d\varphi_i(t)}{dt} = \frac{\alpha I_i(t)}{N_i(t)} + \sum_k \frac{\varphi_k(t)N_k(t) - \varphi_i(t)N_i(t)}{N_i(t)} \cdot \frac{D}{d_{ik}} - \beta(t)\varphi_i(t)$$

where α is the average shedding rate of the pathogen to the environment per infected individual and $N = S + I$ the size of the node. Next comes the spatial coupling among proximal nodes, where D is the rate of the local spread and d_{ik} the distance between holdings i and k. The seasonal decay and removal of the pathogen is captured by $\beta(t)$. The environmental infectious pressure $\varphi(t)$ in each node is evolved each time unit by the Euler forward method. The value of $\varphi(t)$ is saved at the time-points specified in `tspan`.

Seasonal Decay ($\beta(t)$): The decay rate $\beta(t)$ is piecewise constant, defined by four intervals determined by the parameters `end_t1`, `end_t2`, `end_t3`, and `end_t4` (days of the year, where `0 <= day < 365`). The year is divided into four intervals based on the sorted order of these endpoints. The interval that wraps around the year boundary (from the last endpoint to day 365, then from day 0 to the first endpoint) receives the same rate as the interval preceding the first endpoint. Three orderings are supported:

Case 1: `end_t1 < end_t2 < end_t3 < end_t4`

- Interval 1: `[0, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1 (wrap-around): `[end_t4, 365)` with rate `beta_t1`

Case 2: `end_t3 < end_t4 < end_t1 < end_t2`

- Interval 3: `[0, end_t3)` with rate `beta_t3`
- Interval 4: `[end_t3, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`

- Interval 3 (wrap-around): `[end_t2, 365)` with rate `beta_t3`

Case 3: `end_t4` < `end_t1` < `end_t2` < `end_t3`

- Interval 4: `[0, end_t4)` with rate `beta_t4`
- Interval 1: `[end_t4, end_t1)` with rate `beta_t1`
- Interval 2: `[end_t1, end_t2)` with rate `beta_t2`
- Interval 3: `[end_t2, end_t3)` with rate `beta_t3`
- Interval 4 (wrap-around): `[end_t3, 365)` with rate `beta_t4`

These different orderings allow the model to handle seasonal patterns where, for example, a winter peak crosses the year boundary.

See Also

`SISe_sp` for creating an SISE_SP model object and `SimInf_model` for the parent class definition.

`summary,SimInf_abc-method`

Detailed summary of a `SimInf_abc` *object*

Description

Display a detailed summary of an approximate Bayesian computation (ABC) analysis, including the number of particles per generation, the number of generations, and—for each generation—the acceptance rate, effective sample size (ESS), and summary statistics (minimum, first quartile, median, mean, third quartile, and maximum) for each parameter in the posterior distribution.

Usage

```
## S4 method for signature 'SimInf_abc'
summary(object, ...)
```

Arguments

`object`	The `SimInf_abc` object to summarize.
`...`	Additional arguments affecting the summary produced. Currently ignored.

Details

Compared to `show`, the summary additionally displays posterior statistics for all generations, not just the final one.

Value

`NULL`, returned invisibly.

See Also

abc for running an ABC analysis, continue_abc for continuing an existing ABC chain, SimInf_abc for the class definition, show for a brief summary.

---

summary,SimInf_events-method

Detailed summary of a SimInf_events *object*

Description

Display a summary of the scheduled events in a SimInf_events object. The output includes the total number of events and summary statistics (count, minimum, maximum, and average number of individuals affected by each event).

Usage

```
## S4 method for signature 'SimInf_events'
summary(object, ...)
```

Arguments

object	The SimInf_events object to summarize.
...	Additional arguments affecting the summary produced. Currently ignored.

Details

Four event types are reported:

- **Exit**: Events where individuals are removed from a node (e.g., deaths).
- **Enter**: Events where individuals are added to a node (e.g., births).
- **Internal transfer**: Events moving individuals between compartments within a node (e.g., vaccination, ageing).
- **External transfer**: Events moving individuals between different nodes (e.g., livestock movements).

Value

NULL, returned invisibly.

See Also

SimInf_events for creating event objects, SimInf_events for the class definition, SimInf_model for how events are attached to a model, events for extracting events from a model.

Examples

```
## Create an 'SIR' model with 1600 cattle herds (nodes) and
## scheduled events for the population of nodes with births,
## deaths and between-node movements of individuals. Define
## 'tspan' to record the state of the system at daily time
## points over 4*365 days.
model <- SIR(
  u0     = u0_SIR(),
  tspan  = 1:(4 * 365),
  events = events_SIR(),
  beta   = 0.16,
  gamma  = 0.077
)

## Detailed summary of the events
summary(events(model))
```

---

summary,SimInf_individual_events-method

Detailed summary of a SimInf_individual_events *object*

Description

Display a summary of a SimInf_individual_events object, including the number of unique individuals, the total number of event records, and the number of events by event type (exit, enter, internal transfer, and external transfer).

Usage

```
## S4 method for signature 'SimInf_individual_events'
summary(object, ...)
```

Arguments

object	The SimInf_individual_events object to summarize.
...	Additional arguments affecting the summary produced. Currently ignored.

Value

NULL, returned invisibly.

See Also

individual_events for creating SimInf_individual_events objects, SimInf_individual_events for the class definition, SimInf_events for the node-level event class, show for a brief summary.

summary,SimInf_model-method

Detailed summary of a SimInf_model *object*

Description

Display a summary of a SimInf_model object, including the model name, number of nodes, number of replicates (if greater than one), transitions, global data, local data, scheduled events, continuous state variables, and compartments. Compared to show, the summary additionally displays transition details, global and local data summaries, and scheduled events overview.

Usage

```
## S4 method for signature 'SimInf_model'
summary(object, ...)
```

Arguments

object	The SimInf_model object to summarize.
...	Additional arguments affecting the summary produced. Currently ignored.

Details

As with the show method, the output differs depending on whether the model has been run: before running, the result matrices (U and V) are empty; after calling run, they contain the simulated trajectory data.

Value

NULL, returned invisibly.

See Also

SimInf_model for creating model objects, run for simulating a trajectory from a model, SimInf_model for the class definition, show for a brief summary.

Examples

```
## For reproducibility, specify the number of threads.
set_num_threads(1)

## Create an 'SIR' model with 10 nodes and initialise
## it to run over 100 days.
model <- SIR(
  u0 = data.frame(
    S = rep(99, 10),
    I = rep(1, 10),
    R = rep(0, 10)
```

```
  ),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Detailed summary of the model
model

## Run the model with a fixed seed for reproducibility.
result <- run(model, seed = 22)

## Detailed summary of the result. Note that the trajectory is
## non-empty after running the model.
result
```

`summary,SimInf_pfilter-method`

Detailed summary of a `SimInf_pfilter` *object*

Description

Display a detailed summary of a bootstrap particle filter analysis, including the number of particles, log-likelihood estimate, and model characteristics (model name, number of nodes, scheduled events, transitions, global data, local data, continuous state variables, and compartments).

Usage

```
## S4 method for signature 'SimInf_pfilter'
summary(object, ...)
```

Arguments

`object`	The `SimInf_pfilter` object to summarize.
`...`	Additional arguments affecting the summary produced. Currently ignored.

Details

Compared to `show`, the summary additionally displays full model characteristics from the underlying `SimInf_model` object.

Value

`NULL`, returned invisibly.

See Also

`pfilter` for running a particle filter, `logLik` for extracting the log-likelihood, `SimInf_pfilter` for the class definition, `SimInf_model` for the model class embedded in the filter, `show` for a brief summary.

summary,SimInf_pmcmc-method

Detailed summary of a SimInf_pmcmc *object*

Description

Display a detailed summary of a particle Markov chain Monte Carlo (PMCMC) analysis, including the number of iterations, number of particles, acceptance ratio, model characteristics (name, number of nodes, transitions), and posterior quantiles (2.5 75 parameter.

Usage

```
## S4 method for signature 'SimInf_pmcmc'
summary(object, ...)
```

Arguments

object	The SimInf_pmcmc object to summarize.
...	Additional arguments affecting the summary produced. Currently ignored.

Details

Compared to show, the summary additionally displays model characteristics such as the model name, number of nodes, and transition details.

Value

NULL, returned invisibly.

See Also

pmcmc for running a PMCMC analysis, continue_pmcmc for continuing an existing PMCMC chain, SimInf_pmcmc for the class definition, show for a brief summary.

trajectory

Generic function to extract data from a simulated trajectory

Description

Generic function to extract data from a simulated trajectory

Usage

```
trajectory(model, compartments = NULL, index = NULL, ...)
```

Arguments

model	the object to extract the trajectory from.
compartments	specify the names of the compartments to extract data from. The compartments can be specified as a character vector e.g. compartments = c('S', 'I', 'R'), or as a formula e.g. compartments = ~S+I+R (see 'Examples'). Default (compartments=NULL) is to extract the number of individuals in each compartment i.e. the data from all discrete state compartments in the model. In models that also have continuous state variables e.g. the SISe model, they are also included.
index	indices specifying the subset of nodes to include when extracting data. Default (index = NULL) is to extract data from all nodes.
...	Additional arguments, see trajectory

---

trajectory,SimInf_model-method

Extract data from a simulated trajectory

Description

Extract the number of individuals in each compartment in every node after generating a single stochastic trajectory with run.

Usage

```
## S4 method for signature 'SimInf_model'
trajectory(model, compartments, index, format = c("data.frame", "matrix"))
```

Arguments

model	the SimInf_model object to extract the result from.
compartments	specify the names of the compartments to extract data from. The compartments can be specified as a character vector e.g. compartments = c('S', 'I', 'R'), or as a formula e.g. compartments = ~S+I+R (see 'Examples'). Default (compartments=NULL) is to extract the number of individuals in each compartment i.e. the data from all discrete state compartments in the model. In models that also have continuous state variables e.g. the SISe model, they are also included.
index	indices specifying the subset of nodes to include when extracting data. Default (index = NULL) is to extract data from all nodes.
format	the default (format = "data.frame") is to generate a data.frame with one row per node and time-step with the number of individuals in each compartment. When the model contains multiple replicates of each node, the data.frame also contains one column replicate. Using format = "matrix" returns the result as a matrix, which is the internal format (see 'Details').

Value

A `data.frame` if `format = "data.frame"`, else a matrix.

Internal format of the discrete state variables

Description of the layout of the internal matrix (`U`) that is returned if `format = "matrix"`. `U[, j]` contains the number of individuals in each compartment at `tspan[j]`. `U[1:Nc, j]` contains the number of individuals in node 1 at `tspan[j]`. `U[(Nc + 1):(2 * Nc), j]` contains the number of individuals in node 2 at `tspan[j]` etc, where `Nc` is the number of compartments in the model. The dimension of the matrix is $N_n N_c \times$ `length(tspan)` where N_n is the number of nodes. Since version 10, the internal format of `U` has been expanded to also allow replicates of each node. This new functionality is used by the bootstrap filtering algorithm. Each replicate adds new columns to `U` so that the data for each replicate is in blocks of `length(tspan)` columns.

Internal format of the continuous state variables

Description of the layout of the matrix that is returned if `format = "matrix"`. The result matrix for the real-valued continuous state. `V[, j]` contains the real-valued state of the system at `tspan[j]`. The dimension of the matrix is N_n`dim(ldata)[1]` $\times$ `length(tspan)`. Since version 10, the internal format of `V` has been expanded to also allow replicates of each node. This new functionality is used by the bootstrap filtering algorithm. Each replicate adds new columns to `V` so that the data for each replicate is in blocks of `length(tspan)` columns.

Examples

```
## Create an 'SIR' model with 6 nodes and initialize
## it to run over 10 days.
u0 <- data.frame(S = 100:105, I = 1:6, R = rep(0, 6))
model <- SIR(u0 = u0, tspan = 1:10, beta = 0.16, gamma = 0.077)

## Run the model to generate a single stochastic trajectory.
result <- run(model)

## Extract the number of individuals in each compartment at the
## time-points in 'tspan'.
trajectory(result)

## Extract the number of recovered individuals in the first node
## at the time-points in 'tspan'.
trajectory(result, compartments = "R", index = 1)

## Extract the number of recovered individuals in the first and
## third node at the time-points in 'tspan'.
trajectory(result, compartments = "R", index = c(1, 3))

## Create an 'SISe' model with 6 nodes and initialize
## it to run over 10 days.
u0 <- data.frame(S = 100:105, I = 1:6)
model <- SISe(u0 = u0, tspan = 1:10, phi = rep(0, 6),
    upsilon = 0.02, gamma = 0.1, alpha = 1, epsilon = 1.1e-5,
    beta_t1 = 0.15, beta_t2 = 0.15, beta_t3 = 0.15, beta_t4 = 0.15,
```

```
    end_t1 = 91, end_t2 = 182, end_t3 = 273, end_t4 = 365)

## Run the model
result <- run(model)

## Extract the continuous state variable 'phi' which represents
## the environmental infectious pressure.
trajectory(result, "phi")
```

`trajectory,SimInf_pfilter-method`
Extract filtered trajectory from running a particle filter

Description

Extract filtered trajectory from running a particle filter

Usage

```
## S4 method for signature 'SimInf_pfilter'
trajectory(model, compartments, index, format = c("data.frame", "matrix"))
```

Arguments

`model`	the `SimInf_pfilter` object to extract the result from.
`compartments`	specify the names of the compartments to extract data from. The compartments can be specified as a character vector e.g. `compartments = c('S', 'I', 'R')`, or as a formula e.g. `compartments = ~S+I+R` (see 'Examples'). Default (`compartments=NULL`) is to extract the number of individuals in each compartment i.e. the data from all discrete state compartments in the model. In models that also have continuous state variables e.g. the `SISe` model, they are also included.
`index`	indices specifying the subset of nodes to include when extracting data. Default (`index = NULL`) is to extract data from all nodes.
`format`	the default (`format = "data.frame"`) is to generate a `data.frame` with one row per node and time-step with the number of individuals in each compartment. Using `format = "matrix"` returns the result as a matrix, which is the internal format (see 'Details' in `trajectory`).

Value

A `data.frame` if `format = "data.frame"`, else a matrix.

u0 *Get the initial compartment state (u0) in each node*

Description

Extract the initial state vector (u0) from a SimInf_model object as a data.frame.

Usage

```
u0(object, ...)

## S4 method for signature 'SimInf_model'
u0(object, ...)
```

Arguments

object	A SimInf_model object.
...	Additional arguments.

Details

The returned data frame has one row per node and one column per compartment (e.g., S, I, R). This format is convenient for inspection, modification, or exporting the initial conditions.

Value

a data.frame with the initial compartment state.

Examples

```
## Create an SIR model object.
model <- SIR(
  u0 = data.frame(S = 99, I = 1, R = 0),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Get the initial compartment state.
u0(model)

## Modify the initial state (e.g., add 10 infected individuals to
## node 1).
new_u0 <- u0(model)
new_u0[1, "I"] <- new_u0[1, "I"] + 10

## Create a new model with the modified initial state.
new_model <- SIR(
  u0 = new_u0,
```

```
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Alternatively, update the existing model using the setter:
u0(model) <- new_u0
```

---

u0<- *Update the initial compartment state (u0) in each node*

Description

Replace the initial state vector (u0) of a SimInf_model object with new data. This allows you to modify the starting conditions of a model without recreating the object.

Usage

```
u0(model) <- value

## S4 replacement method for signature 'SimInf_model'
u0(model) <- value
```

Arguments

model	A SimInf_model object.
value	An object containing the new initial state. Can be a data.frame, matrix, or named numeric vector. Non-data.frame inputs will be coerced to a data.frame.

Details

The value argument accepts a data.frame, matrix, or named numeric vector. If the input is not a data.frame, it will be automatically coerced to one. The function handles the following formats:

- **Single Node**: If value is a named vector or a one-row matrix/data.frame, it is applied to the single node in the model.
- **Multiple Nodes**: If value is a matrix or data.frame with multiple rows, each row corresponds to one node. The number of rows must exactly match the number of nodes in the model.
- **Column Matching**: Column names must match the compartment names defined in the model (e.g., "S", "I", "R"). Only matching columns are used; extra columns are ignored, and missing compartments will trigger an error.

The function validates the input and ensures the new state is consistent with the model structure before updating.

Value

The modified SimInf_model object.

See Also

v0<- for updating the initial continuous state.

Examples

```
## Create a single-node SIR model.
model <- SIR(
  u0 = data.frame(
    S = 99,
    I = 1,
    R = 0
  ),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Update u0 using a named vector (automatically coerced to one
## row).
u0(model) <- c(
  S = 990,
  I = 10,
  R = 0
)

## Run the model with a fixed seed for reproducibility.
result <- run(model, seed = 22)
plot(result)

## Create a multi-node model (2 nodes).
model_multi <- SIR(
  u0 = data.frame(
    S = c(100, 50),
    I = c(1, 0),
    R = c(0, 0)
  ),
  tspan = 1:100,
  beta = 0.16,
  gamma = 0.077
)

## Update u0 using a data.frame with multiple rows.
u0(model_multi) <- data.frame(
  S = c(200, 100),
  I = c(5, 2),
  R = c(0, 0)
)

result <- run(model_multi, seed = 22)
plot(result)
```

u0_from_individual_events *Derive the initial compartment state from individual events*

Description

Compute the initial number of individuals in each compartment for each node based on a set of individual events. The function sums the net effect of all events occurring before a specified `time` point to determine the starting state of the simulation.

Usage

```
u0_from_individual_events(events, time = NULL, target = NULL, age = NULL)

## S4 method for signature 'SimInf_individual_events'
u0_from_individual_events(events, time = NULL, target = NULL, age = NULL)

## S4 method for signature 'data.frame'
u0_from_individual_events(events, time = NULL, target = NULL, age = NULL)
```

Arguments

`events`	A `SimInf_individual_events` object or a `data.frame` containing individual events. • If a `SimInf_individual_events` object is provided, it is used directly. • If a `data.frame` is provided, it is automatically cleaned and processed using `individual_events`. The data frame must conform to the input format required by `individual_events` (see its documentation for details on required columns).
`time`	A numeric scalar specifying the time point at which to calculate the initial state. If `NULL` (default), the earliest time point among the events is used.
`target`	A character string specifying the target model type (e.g., `"SIR"`, `"SEIR"`, `"SISe3"`). If provided, the function ensures the output `u0` includes all required compartments for that model (e.g., adding zero-initialized `I`, `R`, or age-specific compartments like `S_1`, `S_2`). If `NULL` (default), the output contains only the compartments derived from the events (typically `S` or age-stratified `S_*`), which may require manual renaming or expansion for specific models.
`age`	An integer vector of break points (in days) defining age categories. The intervals are defined as: • Category 1: Age < `age[1]` • Category 2: `age[1]` <= Age < `age[2]` • ... • Last Category: Age >= `tail(age, 1)` If `NULL` (default), all individuals are assigned to a single non-age-stratified susceptible compartment (`S`). If provided, the output will include columns `S_1`, `S_2`, etc., corresponding to the defined age intervals.

Details

This function accepts two types of input for the `events` argument:

- A `SimInf_individual_events` object (already cleaned by `individual_events`).
- A raw `data.frame` of events. If a data frame is provided, it is automatically cleaned and processed using `individual_events` before the initial state is calculated.

This is particularly useful for initializing models from historical movement or demographic data, ensuring the simulation starts with the correct population structure derived from the event log.

Value

A `data.frame` with one row per node and columns representing the initial state. The columns include:

- `key`: The original node identifier from the events.
- `node`: A sequential integer node index (1, 2, ...).
- `S_*`: Columns for susceptible individuals (either `S` or age-stratified `S_1`, `S_2`, etc.).
- Additional compartments (e.g., `I`, `R`, `E`) if `target` is specified, initialized to zero.

See Also

`individual_events` for processing raw event data, `u0` for retrieving the initial state of a model, and `u0<-` for updating the initial state.

---

u0_SEIR *Example initial population data for the SEIR model*

Description

Synthetic dataset containing the initial number of susceptible, exposed, infected, and recovered cattle (individuals) across 1,600 cattle herds (nodes). Provides a heterogeneous population structure for demonstrating SEIR model simulations in a compartmental modeling context.

Usage

```
u0_SEIR()
```

Details

This dataset represents initial disease states in a population of 1,600 cattle herds (nodes). Each row represents a single herd (node), derived from a synthetic population structure by adding an exposed compartment to the SIR model framework.

The data contains:

S Total susceptible cattle (individuals) in the node

E Total exposed cattle (individuals) (initialized to zero)

I Total infected cattle (individuals) (initialized to zero)

R Total recovered cattle (individuals) (initialized to zero)

The herd size distribution is synthetically generated to reflect heterogeneity typical of large-scale populations, making it suitable for illustrating how to incorporate scheduled events in the SimInf framework.

Value

A `data.frame` with 1,600 rows (one per node) and 4 columns:

S Number of susceptible cattle (individuals) in the herd (node)

E Number of exposed cattle (individuals) in the herd (node) (all zero at start)

I Number of infected cattle (individuals) in the herd (node) (all zero at start)

R Number of recovered cattle (individuals) in the herd (node) (all zero at start)

See Also

`SEIR` for creating SEIR models with this initial state and `events_SEIR` for associated movement and demographic events

Examples

```
## For reproducibility, call the set.seed() function and specify the
## number of threads to use. To use all available threads, remove the
## set_num_threads() call.
set.seed(123)
set_num_threads(1)

## Create a 'SEIR' model with 1600 cattle herds (nodes) and initialize
## it to run over 4*365 days. Add ten exposed animals to the first
## herd. Define 'tspan' to record the state of the system at weekly
## time-points. Load scheduled events for the population of nodes with
## births, deaths and between-node movements of individuals.
u0 <- u0_SEIR()
u0$E[1] <- 10
model <- SEIR(
    u0      = u0,
    tspan   = seq(from = 1, to = 4*365, by = 7),
    events  = events_SEIR(),
    beta    = 0.16,
    epsilon = 0.25,
    gamma   = 0.01
)

## Display the number of cattle affected by each event type per day.
plot(events(model))

## Run the model to generate a single stochastic trajectory.
result <- run(model)
```

```
## Plot the median and interquartile range of the number of
## susceptible, exposed, infected and recovered individuals.
plot(result)

## Plot the trajectory for the first herd.
plot(result, index = 1)

## Summarize the trajectory. The summary includes the number of events
## by event type.
summary(result)
```

`u0_SIR` *Example initial population data for the SIR model*

Description

Synthetic dataset containing the initial number of susceptible, infected, and recovered cattle (individuals) across 1,600 cattle herds (nodes). Provides a heterogeneous population structure for demonstrating SIR model simulations in a compartmental modeling context.

Usage

```
u0_SIR()
```

Details

This dataset represents initial disease states in a synthetic population of 1,600 cattle herds (nodes). Each row represents a single herd (node).

The data contains:

S Total susceptible cattle (individuals) in the node

I Total infected cattle (individuals) (initialized to zero)

R Total recovered cattle (individuals) (initialized to zero)

The herd size distribution is synthetically generated to reflect heterogeneity typical of large-scale populations, making it suitable for illustrating how to incorporate scheduled events in the SimInf framework.

Value

A `data.frame` with 1,600 rows (one per node) and 3 columns:

S Number of susceptible cattle (individuals) in the herd (node)

I Number of infected cattle (individuals) in the herd (node) (all zero at start)

R Number of recovered cattle (individuals) in the herd (node) (all zero at start)

See Also

SIR for creating SIR models with this initial state and events_SIR for associated movement and demographic events

Examples

```
## For reproducibility, call the set.seed() function and specify the
## number of threads to use. To use all available threads, remove the
## set_num_threads() call.
set.seed(123)
set_num_threads(1)

## Create an 'SIR' model with 1600 cattle herds (nodes) and initialize
## it to run over 4*365 days. Add one infected animal to the first
## herd to seed the outbreak. Define 'tspan' to record the state of
## the system at daily time-points. Load scheduled events for the
## population of nodes with births, deaths and between-node movements
## of individuals.
u0 <- u0_SIR()
u0$I[1] <- 1
model <- SIR(
    u0     = u0,
    tspan  = seq(from = 1, to = 4*365, by = 1),
    events = events_SIR(),
    beta   = 0.16,
    gamma  = 0.01
)

## Display the number of cattle affected by each event type per day.
plot(events(model))

## Run the model to generate a single stochastic trajectory.
result <- run(model)

## Plot the median and interquartile range of the number of
## susceptible, infected and recovered individuals.
plot(result)

## Plot the trajectory for the first herd.
plot(result, index = 1)

## Summarize the trajectory. The summary includes the number of events
## by event type.
summary(result)
```

---

u0_SIS *Example initial population data for the SIS model*

Description

Synthetic dataset containing the initial number of susceptible, and infected cattle (individuals) across 1,600 cattle herds (nodes). Provides a heterogeneous population structure for demonstrating SIS model simulations in a compartmental modeling context.

Usage

```
u0_SIS()
```

Details

This dataset represents initial disease states in a synthetic population of 1,600 cattle herds (nodes). Each row represents a single herd (node).

The data contains:

S Total susceptible cattle (individuals) in the node

I Total infected cattle (individuals) (initialized to zero)

The herd size distribution is synthetically generated to reflect heterogeneity typical of large-scale populations, making it suitable for illustrating how to incorporate scheduled events in the SimInf framework.

Value

A `data.frame` with 1,600 rows (one per node) and 2 columns:

S Number of susceptible cattle (individuals) in the herd (node)

I Number of infected cattle (individuals) in the herd (node) (all zero at start)

See Also

`SIS` for creating SIS models with this initial state and `events_SIS` for associated movement and demographic events

Examples

```
## For reproducibility, call the set.seed() function and specify the
## number of threads to use. To use all available threads, remove the
## set_num_threads() call.
set.seed(123)
set_num_threads(1)

## Create an 'SIS' model with 1600 cattle herds (nodes) and initialize
## it to run over 4*365 days. Add one infected animal to the first
## herd to seed the outbreak. Define 'tspan' to record the state of
## the system at daily time-points. Load scheduled events for the
## population of nodes with births, deaths and between-node movements
## of individuals.
u0 <- u0_SIS()
u0$I[1] <- 1
```

```
model <- SIS(
    u0     = u0,
    tspan  = seq(from = 1, to = 4*365, by = 1),
    events = events_SIS(),
    beta   = 0.16,
    gamma  = 0.01
)

## Display the number of cattle affected by each event type per day.
plot(events(model))

## Run the model to generate a single stochastic trajectory.
result <- run(model)

## Plot the median and interquartile range of the number of
## susceptible and infected individuals.
plot(result)

## Plot the trajectory for the first herd.
plot(result, index = 1)

## Summarize the trajectory. The summary includes the number of events
## by event type.
summary(result)
```

`u0_SISe3` *Example initial population data for the SISe3 model*

Description

Dataset containing the initial number of susceptible and infected cattle (individuals) across three age categories in 1,600 herds (nodes). Provides a heterogeneous population structure for demonstrating SISe3 model simulations in a compartmental modeling context.

Usage

```
data(u0_SISe3)
```

Format

A `data.frame`

Details

This dataset represents initial disease states in a synthetic population of 1,600 cattle herds (nodes) stratified into three age categories. Each row represents a single herd (node). The SISe3 model extends the SISe model with age-structured compartments (S_1, I_1, S_2, I_2, S_3, I_3) and an environmental compartment for pathogen shedding. This is appropriate for diseases where transmission rates or recovery rates differ by age group.

The data contains:

S_1 Total susceptible cattle (individuals) in age category 1 in the node

I_1 Total infected cattle in age category 1 (initialized to zero)

S_2 Total susceptible cattle in age category 2

I_2 Total infected cattle in age category 2 (initialized to zero)

S_3 Total susceptible cattle in age category 3

I_3 Total infected cattle in age category 3 (initialized to zero)

The herd size distribution and age structure is synthetically generated to reflect heterogeneity typical of large-scale populations, making it suitable for illustrating how to incorporate scheduled events in the SimInf framework.

See Also

SISe3 for creating SISe3 models with this initial state and events_SISe3 for associated cattle movement and demographic events

Examples

```
## For reproducibility, call the set.seed() function and specify the
## number of threads to use. To use all available threads, remove the
## set_num_threads() call.
set.seed(123)
set_num_threads(1)

## Create an 'SISe3' model with 1600 cattle herds (nodes) stratified
## by age, initialize it to run over 4*365 days and record data at
## weekly time-points. Add ten infected animals to age category 1 in
## the first herd to seed the outbreak.  Define 'tspan' to record the
## state of the system at weekly time-points. Load scheduled events
## events for the population of nodes with births, deaths and
## between-node movements of individuals.
u0 <- u0_SISe3
u0$I_1[1] <- 10
model <- SISe3(
    u0 = u0,
    tspan = seq(from = 1, to = 4*365, by = 7),
    events = events_SISe3,
    phi = rep(0, nrow(u0)),
    upsilon_1 = 1.8e-2,
    upsilon_2 = 1.8e-2,
    upsilon_3 = 1.8e-2,
    gamma_1 = 0.1,
    gamma_2 = 0.1,
    gamma_3 = 0.1,
    alpha = 1,
    beta_t1 = 1.0e-1,
    beta_t2 = 1.0e-1,
    beta_t3 = 1.25e-1,
    beta_t4 = 1.25e-1,
    end_t1 = 91,
    end_t2 = 182,
```

```
    end_t3 = 273,
    end_t4 = 365,
    epsilon = 0
)

## Display the number of cattle affected by each event type per day.
plot(events(model))

## Run the model to generate a single stochastic trajectory.
result <- run(model)

## Plot the median and interquartile range of the number of
## susceptible and infected individuals.
plot(result)

## Plot the proportion of nodes with at least one infected individual.
plot(result, I_1 + I_2 + I_3 ~ ., level = 2, type = "l")

## Plot the trajectory for the first herd.
plot(result, index = 1)

## Summarize the trajectory. The summary includes the number of events
## by event type.
summary(result)
```

`v0<-` *Update the initial continuous state (`v0`) in each node*

Description

Replace the initial continuous state vector (`v0`) of a `SimInf_model` object with new data. This allows you to modify the starting conditions of continuous variables (e.g., environmental pathogen concentration) without recreating the object.

Usage

```
v0(model) <- value

## S4 replacement method for signature 'SimInf_model'
v0(model) <- value
```

Arguments

`model`	A `SimInf_model` object.
`value`	An object containing the new initial continuous state. Can be a `data.frame`, `matrix`, or named `numeric vector`. Non-data.frame inputs will be coerced to a `data.frame`.

Details

The value argument accepts a data.frame, matrix, or named numeric vector. If the input is not a data.frame, it will be automatically coerced to one. The function handles the following formats:

- **Single Node**: If value is a named vector or a one-row matrix/data.frame, it is applied to the single node in the model.
- **Multiple Nodes**: If value is a matrix or data.frame with multiple rows, each row corresponds to one node. The number of rows must exactly match the number of nodes in the model.
- **Column Matching**: Column names must match the continuous state variable names defined in the model (e.g., "phi"). Only matching columns are used; extra columns are ignored, and missing variables will trigger an error.

The function validates the input and ensures the new state is consistent with the model structure before updating.

Value

The modified SimInf_model object.

See Also

u0<- for updating the initial discrete compartment state.

Examples

```
## Create an 'SISe' model with no infected individuals and no
## infectious pressure (phi = 0).
model <- SISe(
  u0 = data.frame(S = 100, I = 0),
  tspan = 1:100,
  phi = 0,
  upsilon = 0.02,
  gamma = 0.1,
  alpha = 1,
  epsilon = 0,
  beta_t1 = 0.15,
  beta_t2 = 0.15,
  beta_t3 = 0.15,
  beta_t4 = 0.15,
  end_t1 = 91,
  end_t2 = 182,
  end_t3 = 273,
  end_t4 = 365
)

## Run the 'SISe' model and plot the result.
result <- run(model)
plot(result)

## Update the infectious pressure 'phi' in 'v0' using a named
## vector.  (Automatically coerced to one row for the single
```

```
## node). Run the model with a fixed seed for reproducibility.
v0(model) <- c(phi = 1)
result <- run(model, seed = 22)
plot(result)

## For a multi-node model, use a data.frame with multiple rows:
## v0(model) <- data.frame(phi = c(1.0, 0.5, 0.0))
```

Index

∗ **dataset**
events_SISe3, 28
nodes, 42
u0_SISe3, 138

abc, 5, *10*, *77*, *83–85*, *121*
abc,SimInf_model-method *(*abc*)*, 5
add_spatial_coupling_to_ldata, 8
as.data.frame.SimInf_abc, 10
as.data.frame.SimInf_events, 11
as.data.frame.SimInf_individual_events, 12
as.data.frame.SimInf_pmcmc, 13

boxplot,SimInf_model-method, 14

C_code, 18
continue_abc, *7*, *10*, 15, *77*, *85*, *121*
continue_abc,SimInf_abc-method *(*continue_abc*)*, 15
continue_pmcmc, *13*, 16, *60*, *81*, *95*, *125*
continue_pmcmc,SimInf_pmcmc-method *(*continue_pmcmc*)*, 16

dgCMatrix, *19*, *73*, *86*, *88*
distance_matrix, *9*, 19

edge_properties_to_matrix, *9*, 20
events, *11*, 21, *121*
events,SimInf_model-method *(*events*)*, 21
events_SEIR, *22*, 22, *134*
events_SIR, *22*, 24, *34*, *48*, *136*
events_SIS, 26, *137*
events_SISe3, *22*, 28, *139*

gdata, 31, *37*
gdata,SimInf_model-method *(*gdata*)*, 31
gdata<-, 32
gdata<-,SimInf_model-method *(*gdata<-*)*, 32
get_individuals, 33
get_individuals,SimInf_individual_events-method *(*get_individuals*)*, 33

indegree, 34, *48*
individual_events, *13*, 35, *44*, *79*, *82*, *89*, *122*, *132*, *133*
INSTALL, *50*
install.packages, *50*

lambertW0, 36
ldata, *31*, *32*, 37
ldata,SimInf_model-method *(*ldata*)*, 37
length,SimInf_pmcmc-method, 38
logLik, *81*, *94*, *124*
logLik,SimInf_pfilter-method, 38

mparse, *18*, *22*, 39, *50*, *72*, *82*, *92*, *93*, *96*, *99*, *102*, *108*, *113*, *118*

n_compartments, 44
n_compartments,SimInf_model-method *(*n_compartments*)*, 44
n_generations, 45
n_generations,SimInf_abc-method *(*n_generations*)*, 45
n_nodes, 46
n_nodes,SimInf_model-method *(*n_nodes*)*, 46
n_nodes,SimInf_pfilter-method *(*n_nodes*)*, 46
n_nodes,SimInf_pmcmc-method *(*n_nodes*)*, 46
n_replicates, 47
n_replicates,SimInf_model-method *(*n_replicates*)*, 47
node_events, *36*, 43
node_events,SimInf_individual_events-method *(*node_events*)*, 43
nodes, 42

outdegree, *34*, 48

package_skeleton, *41*, 49, *83*
pairs,SimInf_model-method, 50
pfilter, *39*, *47*, 51, *81*, *94*, *124*
pfilter,SimInf_model-method *(*pfilter*)*, 51
plot, *57*, *72*, *82*, *93*, *96, 97*, *99*, *102, 103*, *108*, *113*, *118*
plot,SimInf_abc-method, 53
plot,SimInf_events-method, 54
plot,SimInf_individual_events-method, 54
plot,SimInf_model-method, 55
plot,SimInf_pfilter-method, 57
plot,SimInf_pmcmc-method, 57
pmcmc, *13*, *17*, 58, *81*, *83*, *95*, *125*
pmcmc,SimInf_model-method *(*pmcmc*)*, 58
prevalence, *61*, 61, *66*, *72*, *82*, *93, 94*, *96, 97*, *99*, *102, 103*, *108*, *113*, *118*
prevalence,SimInf_model-method, 62
prevalence,SimInf_pfilter-method, 64
prevalence,SimInf_pmcmc-method, 65
punchcard<-, 66
punchcard<-,SimInf_model-method *(*punchcard<-*)*, 66

run, *22*, 68, *72*, *80*, *82*, *89*, *93*, *96, 97*, *99*, *102, 103*, *108*, *113*, *118*, *123*, *126*
run,SEIR-method *(*run*)*, 68
run,SimInf_abc-method *(*run*)*, 68
run,SimInf_model-method *(*run*)*, 68
run,SIR-method *(*run*)*, 68
run,SIS-method *(*run*)*, 68
run,SISe-method *(*run*)*, 68
run,SISe3-method *(*run*)*, 68
run,SISe3_sp-method *(*run*)*, 68
run,SISe_sp-method *(*run*)*, 68

SEIR, *23*, *41*, 70, *72, 73*, *89*, *92, 93*, *96, 97*, *99*, *102*, *108*, *113*, *118*, *134*
SEIR-class, 72
select_matrix, 73
select_matrix,SimInf_model-method *(*select_matrix*)*, 73
select_matrix<-, 73
select_matrix<-,SimInf_model-method *(*select_matrix<-*)*, 73
set_num_threads, 74
shift_matrix, 75
shift_matrix,SimInf_model-method *(*shift_matrix*)*, 75
shift_matrix<-, 76
shift_matrix<-,SimInf_model-method *(*shift_matrix<-*)*, 76
show, *82*, *120–125*
show,SimInf_abc-method, 77
show,SimInf_events-method, 78
show,SimInf_individual_events-method, 79
show,SimInf_model-method, 79
show,SimInf_pfilter-method, 80
show,SimInf_pmcmc-method, 81
SimInf, 82
SimInf-package *(*SimInf*)*, 82
SimInf_abc, *10*, *77*, *120, 121*
SimInf_abc-class, 84
SimInf_events, *11*, *22, 23*, *25*, *27*, *30*, *34*, *40*, *48*, *73–76*, *78, 79*, *85*, 85, *89–93*, *121, 122*
SimInf_events-class, 87
SimInf_individual_events, *13*, *35*, *79*, *122*
SimInf_individual_events-class, 89
SimInf_model, *31, 32*, *37*, *39–41*, *49, 50*, *71–73*, *78–80*, *82*, *84*, *89, 90*, 90, *91–100*, *103–105*, *108, 109*, *111*, *113*, *115, 116*, *118*, *120, 121*, *123*, *124*
SimInf_model-class, 92
SimInf_pfilter, *38, 39*, *81*, *124*
SimInf_pfilter-class, 93
SimInf_pmcmc, *13*, *16, 17*, *60*, *81*, *125*
SimInf_pmcmc-class, 94
SIR, *25*, *41*, *72, 73*, *82*, *89*, *92, 93*, 95, *96, 97*, *99*, *102*, *108*, *113*, *118*, *136*
SIR-class, 97
SIS, *27*, *41*, *72*, *82*, *89*, *92, 93*, *96, 97*, 97, *99*, *102*, *108*, *113*, *118*, *137*
SIS-class, 99
SISe, *41*, *72*, *89*, *92, 93*, *96*, *99*, 100, *102*, *104*, *108*, *118*
SISe-class, 103
SISe3, *30*, *72*, *96*, *99*, *102*, 104, *108, 109*, *113*, *139*
SISe3-class, 108
SISe3_sp, 110, *113*, *115*, *118*
SISe3_sp-class, 113
SISe_sp, *72*, *96*, *99*, *102*, *108*, *113*, 115, *118*,

120
SISe_sp-class, 118
summary, *82*
summary,SimInf_abc-method, 120
summary,SimInf_events-method, 121
summary,SimInf_individual_events-method, 122
summary,SimInf_model-method, 123
summary,SimInf_pfilter-method, 124
summary,SimInf_pmcmc-method, 125

trajectory, *68*, *72*, *82*, *93*, *94*, *96*, *97*, *99*, *102*, *103*, *108*, *113*, *118*, 125, *126*, *128*
trajectory,SimInf_model-method, 126
trajectory,SimInf_pfilter-method, 128

U, *80*, *123*
u0, 129, *133*
u0,SimInf_model-method *(*u0*)*, 129
u0<-, 130
u0<-,SimInf_model-method *(*u0<-*)*, 130
u0_from_individual_events, *89*, 132
u0_from_individual_events,data.frame-method *(*u0_from_individual_events*)*, 132
u0_from_individual_events,SimInf_individual_events-method *(*u0_from_individual_events*)*, 132
u0_SEIR, *23*, 133
u0_SIR, *25*, 135
u0_SIS, *27*, 136
u0_SISe3, *30*, 138

V, *80*, *123*
v0<-, 140
v0<-,SimInf_model-method *(*v0<-*)*, 140