Package ‘SimInf’

May 13, 2025

Title A Framework for Data-Driven Stochastic Disease Spread Simulations

Version 9.8.1.9001

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>.

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`, `http://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, mvtnorm, 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_indiv_events.R' 'run.R' 'density_ratio.R' 'abc.R' 'degree.R' 'distance.R' 'distributions.R' 'edge_properties.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

RoxygenNote 7.3.2

VignetteBuilder utils, knitr

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

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

R topics documented:

`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 = getOption("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 = getOption("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`. The default is to retrieve the global option `verbose` and use `FALSE` if it is not set.
`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: 10.1214/20-BA1211

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)
```

as.data.frame.SimInf_abc

Coerce to data frame

Description

Coerce to data frame

Usage

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

Arguments

x	any R object.
...	additional arguments to be passed to or from methods.

as.data.frame.SimInf_events

Coerce events to a data frame

Description

Coerce events to a data frame

Usage

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

Arguments

`x`	any R object.
`...`	additional arguments to be passed to or from methods.

as.data.frame.SimInf_indiv_events

Coerce to data frame

Description

Coerce to data frame

Usage

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

Arguments

`x`	any R object.
`...`	additional arguments to be passed to or from methods.

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

Description

Coerce to data frame

Usage

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

Arguments

`x`	any R object.
`...`	additional arguments to be passed to or from methods.

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

Description

Produce box-and-whisker plot(s) of the number of individuals in each model compartment.

Usage

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

Arguments

`x`	The `model` to plot
`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`) includes all compartments.
`index`	indices specifying the nodes to include when plotting data. Default `index = NULL` include all nodes in the model.
`...`	Additional arguments affecting the plot produced.

Examples

```
## Create an 'SIR' model with 10 nodes and initialise
## it with 99 susceptible individuals and one infected
## individual. Let the model 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)

## Run the model and save the result.
result <- run(model)

## Create a boxplot that includes all compartments in all nodes.
boxplot(result)

## Create a boxplot that includes the S and I compartments in
## nodes 1 and 2.
boxplot(result, ~S+I, 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 = getOption("verbose", FALSE),
  post_gen = NULL
)

## S4 method for signature 'SimInf_abc'
continue_abc(
  object,
  tolerance = NULL,
  data = NULL,
  verbose = getOption("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`. The default is to retrieve the global option `verbose` and use `FALSE` if it is not set.
`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` *Run more iterations of PMCMC*

Description

Run more iterations of PMCMC

Usage

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

Arguments

`object`	The `SimInf_pmcmc` object to continue 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. 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`	An optional function that, if non-NULL, is applied in the particle filter 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.
`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`. The default is to retrieve the global option `verbose` and use `FALSE` if it is not set. 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`.

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

Description

Extract the C code from a `SimInf_model` object

Usage

```
C_code(model)
```

Arguments

`model`	The `SimInf_model` object to extract the C code from.

Value

Character vector with C code for the model.

Examples

```
## Use the model parser to create a 'SimInf_model' object that
## expresses an SIR model, where 'b' is the transmission rate and
## 'g' is the recovery rate.
model <- mparse(transitions = c("S -> b*S*I/(S+I+R) -> I", "I -> g*I -> R"),
                compartments = c("S", "I", "R"),
                gdata = c(b = 0.16, g = 0.077),
                u0 = data.frame(S = 99, I = 1, R = 0),
                tspan = 1:10)

## View the C code.
C_code(model)
```

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

Description

Calculate the euclidian distances beween coordinates for all coordinates within the cutoff.

Usage

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

Arguments

`x`	Projected x coordinate
`y`	Projected y coordinate
`cutoff`	The distance cutoff
`min_dist`	The minimum distance to separate two nodes. If the coordinates for two nodes are identical, the min_dist must be assigned or an error is raised. Default is `NULL`, i.e., to raise an error.
`na_fail`	A logical indicating whether missing values in `x` or `y` should raise an error or assign zero to all distances involving missing values. Default is `TRUE`, i.e., to raise an error.

Value

`dgCMatrix`

Examples

```
## Generate a grid 10 x 10 and place one node in each cell
## separated by 100m.
nodes <- expand.grid(x = (0:9) * 100, y = (0:9) * 100)
plot(y ~ x, nodes)

## Define the cutoff to only include neighbors within 300m.
d <- distance_matrix(x = nodes$x, y = nodes$y, cutoff = 300)

## View the first 10 rows and columns in the distance matrix
d[1:10, 1:10]
```

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

Description

A utility function to facilitate preparing edge properties for `ldata` in a model.

Usage

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

Arguments

`edges`	a `data.frame` with properties assigned for each edge ’from’ –> ’to’, for example, weight or count. The `data.frame` must contain the columns `’from’` and `’to’` with valid indices to the nodes (1 <= index <= n_nodes).
`n_nodes`	the total number of nodes in the model. The resulting matrix will have the number of columns equal to `n_nodes`.

Details

The edge properties will be converted to a matrix where each row in `edges` will become a sequence of (index, value_1, value_2, ..., value_n) where 'index' is the zero-based index of the `from` node. The reason for a zero-based index is to facilitate it's usage in C code. The sequence will be added to the 'to' column in the matrix. There will always be at least one stop value=-1 in each column. All other values in the matrix will be set to `NaN`. See 'Examples'.

Value

a numeric matrix with the number of rows equal to `max(table(edges$to)) * (ncol(edges) - 1) + 1` and the number of columns equal to `n_nodes`.

Examples

```
## Let us consider the following edge properties.
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))

## Converting the edge properties into a matrix
edge_properties_to_matrix(edges, 6)

## Gives the following output. The first column contains first the
## properties for the edge from = 2 --> to = 1, where the first
## row is the zero-based index of from, i.e., 1. The second row
## contains the rate=0.2 and the third row count=5. On the fourth
## row starts the next sequence with the values in the second row
## in the edges data.frame. The stop value in the first column is
## on row 10. As can be seen in column 6, there are no edge
## properties for node=6.
##        [,1] [,2]  [,3] [,4] [,5] [,6]
##  [1,]  1.00    0  3.00  0.0  0.0   -1
##  [2,]  0.20    1  0.20  0.2  0.2  NaN
##  [3,]  5.00   50 10.00  5.0  5.0  NaN
##  [4,]  2.00   -1  4.00  2.0  2.0  NaN
##  [5,]  0.01  NaN  0.05  0.8  0.8  NaN
##  [6,]  5.00  NaN 10.00  5.0  5.0  NaN
##  [7,]  3.00  NaN -1.00 -1.0 -1.0  NaN
##  [8,]  0.79  NaN   NaN  NaN  NaN  NaN
##  [9,]  5.00  NaN   NaN  NaN  NaN  NaN
## [10,] -1.00  NaN   NaN  NaN  NaN  NaN
```

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

Description

Extract the scheduled events from a `SimInf_model` object.

Usage

```
events(object, ...)

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

Arguments

`object`	The `model` to extract the events from.
`...`	Additional arguments affecting the generated events.

Value

`SimInf_events` object.

Examples

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

## Extract the scheduled events from the model and display summary
summary(events(model))

## Extract the scheduled events from the model and plot them
plot(events(model))
```

`events_SEIR` *Example data to initialize events for the 'SEIR' model*

Description

Example data to initialize scheduled events for a population of 1600 nodes and demonstrate the `SEIR` model.

Usage

```
events_SEIR()
```

Details

Example data to initialize scheduled events (see `SimInf_events`) for a population of 1600 nodes and demonstrate the `SEIR` model. The dataset contains 466692 events for 1600 nodes distributed over 4 * 365 days. The events are divided into three types: 'Exit' events remove individuals from the population (n = 182535), 'Enter' events add individuals to the population (n = 182685), and 'External transfer' events move individuals between nodes in the population (n = 101472). The vignette contains a detailed description of how scheduled events operate on a model.

Value

A `data.frame`

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 'SEIR' model with 1600 nodes and initialize
## it to run over 4*365 days. Add one infected individual
## to the first node.
u0 <- u0_SEIR()
u0$I[1] <- 1
tspan <- seq(from = 1, to = 4*365, by = 1)
model <- SEIR(u0      = u0,
              tspan   = tspan,
              events  = events_SEIR(),
              beta    = 0.16,
              epsilon = 0.25,
              gamma   = 0.01)

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

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

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

`events_SIR` *Example data to initialize events for the 'SIR' model*

Description

Example data to initialize scheduled events for a population of 1600 nodes and demonstrate the `SIR` model.

Usage

```
events_SIR()
```

Details

Example data to initialize scheduled events (see SimInf_events) for a population of 1600 nodes and demonstrate the SIR model. The dataset contains 466692 events for 1600 nodes distributed over 4 * 365 days. The events are divided into three types: 'Exit' events remove individuals from the population (n = 182535), 'Enter' events add individuals to the population (n = 182685), and 'External transfer' events move individuals between nodes in the population (n = 101472). The vignette contains a detailed description of how scheduled events operate on a model.

Value

A `data.frame`

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 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  = 0.01)

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

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

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

`events_SIS` *Example data to initialize events for the 'SIS' model*

Description

Example data to initialize scheduled events for a population of 1600 nodes and demonstrate the `SIS` model.

Usage

```
events_SIS()
```

Details

Example data to initialize scheduled events (see `SimInf_events`) for a population of 1600 nodes and demonstrate the `SIS` model. The dataset contains 466692 events for 1600 nodes distributed over 4 * 365 days. The events are divided into three types: ‘Exit’ events remove individuals from the population (n = 182535), ‘Enter’ events add individuals to the population (n = 182685), and ‘External transfer’ events move individuals between nodes in the population (n = 101472). The vignette contains a detailed description of how scheduled events operate on a model.

Value

A `data.frame`

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 nodes and initialize
## it to run over 4*365 days. Add one infected individual
## to the first node.
u0 <- u0_SIS()
u0$I[1] <- 1
tspan <- seq(from = 1, to = 4*365, by = 1)
model <- SIS(u0     = u0,
             tspan  = tspan,
             events = events_SIS(),
             beta   = 0.16,
             gamma  = 0.01)

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

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

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

```
summary(result)
```

`events_SISe` *Example data to initialize events for the 'SISe' model*

Description

Example data to initialize scheduled events for a population of 1600 nodes and demonstrate the `SISe` model.

Usage

```
events_SISe()
```

Details

Example data to initialize scheduled events (see `SimInf_events`) for a population of 1600 nodes and demonstrate the `SISe` model. The dataset contains 466692 events for 1600 nodes distributed over 4 * 365 days. The events are divided into three types: 'Exit' events remove individuals from the population (n = 182535), 'Enter' events add individuals to the population (n = 182685), and 'External transfer' events move individuals between nodes in the population (n = 101472). The vignette contains a detailed description of how scheduled events operate on a model.

Value

A `data.frame`

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 'SISe' model with 1600 nodes and initialize
## it to run over 4*365 days. Add one infected individual
## to the first node.
u0 <- u0_SISe()
u0$I[1] <- 1
tspan <- seq(from = 1, to = 4*365, by = 1)
model <- SISe(u0 = u0, tspan = tspan, events = events_SISe(),
              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 = 91, end_t2 = 182,
              end_t3 = 273, end_t4 = 365, epsilon = 0)

## Display the number of individuals affected by each event type
## per day.
```

```
plot(events(model))

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

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

events_SISe3 *Example data to initialize events for the 'SISe3' model*

Description

Example data to initialize scheduled events for a population of 1600 nodes and demonstrate the `SISe3` model.

Usage

```
data(events_SISe3)
```

Format

A `data.frame`

Details

Example data to initialize scheduled events (see `SimInf_events`) for a population of 1600 nodes and demonstrate the `SISe3` model. The dataset contains 783773 events for 1600 nodes distributed over 4 * 365 days. The events are divided into three types: 'Exit' events remove individuals from the population (n = 182535), 'Enter' events add individuals to the population (n = 182685), 'Internal transfer' events move individuals between compartmens within one node e.g. ageing (n = 317081), and 'External transfer' events move individuals between nodes in the population (n = 101472). The vignette contains a detailed description of how scheduled events operate on a model.

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 nodes and initialize
## it to run over 4*365 days. Add one infected individual
## to the first node.
data("u0_SISe3", package = "SimInf")
data("events_SISe3", package = "SimInf")
u0_SISe3$I_1[1] <- 1
```

```
tspan <- seq(from = 1, to = 4*365, by = 1)
model <- SISe3(u0 = u0_SISe3, tspan = tspan, events = events_SISe3,
               phi = rep(0, nrow(u0_SISe3)), 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 individuals affected by each event type
## per day.
plot(events(model))

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

## 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

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

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_indiv_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_indiv_events'
get_individuals(x, time = NULL)
```

Arguments

`x`	an individual events object of class `SimInf_indiv_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

The number of nodes with inward *external transfer* events to each node.

Usage

```
indegree(model)
```

Arguments

`model`	determine in-degree for each node in the model.

Value

vector with in-degree for each node.

Examples

```
## Create an 'SIR' model with 1600 nodes and initialize
## it with example data.
model <- SIR(u0 = u0_SIR(), tspan = 1:1460, events = events_SIR(),
             beta   = 0.16, gamma  = 0.077)

## Display indegree for each node in the model.
plot(indegree(model))
```

`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_indiv_events

See Also

node_events.

`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

Examples

```
## Create an 'SISe' model with 1600 nodes.
model <- SISe(u0 = u0_SISe(), tspan = 1:100, events = events_SISe(),
              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

Length of the MCMC chain

Usage

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

Arguments

`x`	The `SimInf_pmcmc` object determine the length of the MCMC chain for.

`logLik,SimInf_pfilter-method` *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.

mparse — *Model parser to define new models to run in* `SimInf`

Description

Describe your model in a logical way in R. `mparse` creates a `SimInf_model` object with your model definition that is ready to `run`.

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
)
```

Arguments

`transitions`	character vector containing transitions on the form `"X -> ... -> Y"`. The left (right) side is the initial (final) state and the propensity is written in between the `->`-signs. The special symbol `@` is reserved for the empty set. For example, `transitions = c("S -> beta*S*I/(S+I+R) -> I", "I -> gamma*I -> R")` expresses the SIR model. It is also possible to define variables which can then be used in calculations of propensities or in calculations of other variables. A variable is defined by the operator `<-`. Using a variable for the size of the population, the SIR model can instead be written `transitions = c("S -> beta*S*I/N -> I","I -> gamma*I -> R", "N <- S+I+R")`. By default, the type of a variable is defined as a double in the generated C code, but it is possible to also define it as an integer by writing `(int)` before the variable name. For example, for the SIR model, the population size can be defined as `"(int)N <- S+I+R"`. It is also possible to explicitly use (double) in front of the variable name, but it is not needed because it is the default. Note that the order of propensities and variables does not matter.
`compartments`	contains the names of the involved compartments, for example, `compartments = c("S", "I","R")`.
`ldata`	optional data for the nodes. Can be specified as a `data.frame` with one row per node, as a numeric matrix where column `ldata[, j]` contains the local

	data vector for the node `j`, or as a as a named vector when the model only contains one node. If `ldata` is specified as a `data.frame`, each column is one parameter. If `v0` is specified as a matrix, it must have row names to identify the parameters in the transitions. If `v0` is specified as a named vector, the names identify the parameters. The local data vector is passed as an argument to the transition rate functions and the post time step function.
`gdata`	optional data that are common to all nodes in the model. Can be specified either as a optionally named numeric vector or as as a one-row data.frame. The names are used to identify the parameters in the transitions. When `gdata` is specified as a vector, it is possible to have parameters without names, however, these parameters will not be automatically identified by mparse but need to be identified in the code by the user. The global data vector is passed as an argument to the transition rate functions and the post time step function.
`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`.
`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. A `Date` vector is coerced to a numeric vector as days, 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`	A `data.frame` with the scheduled events. Default is `NULL` i.e. no scheduled events in the model.
`E`	matrix to handle scheduled events, see `SimInf_events`. Default is `NULL` i.e. no scheduled events in the model.
`N`	matrix to handle scheduled events, see `SimInf_events`. Default is `NULL` i.e. no scheduled events in the model.
`pts_fun`	optional character vector with C code for the post time step function. The C code should contain only the body of the function i.e. the code between the opening and closing curly brackets.
`use_enum`	generate enumeration constants for the indices to each parameter in the 'u', 'v', 'ldata', and 'gdata' vectors in the generated C code. The name of each enumeration constant will be transformed to the upper-case name of the corresponding parameter, for example, a parameter 'beta' will become 'BETA'. The enumeration constants 'N_COMPARTMENTS_U' and 'N_COMPARTMENTS_V' will be automatically added to facilitate indexing 'u' and 'v' in the C code. These

two enumeration constants cannot be used as a compartment or variable name. Using enumeration constants can make it easier to modify the C code afterwards, or when writing C code for the `pts_fun` parameter. Default is `FALSE`, i.e., the parameters are specified by using integer indices for the parameters.

Value

a `SimInf_model` object

Examples

```
## Not run:
## Use the model parser to create a 'SimInf_model' object that
## expresses the SIR model, where 'beta' is the transmission rate
## and 'gamma' is the recovery rate.
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)

## Run and plot the result
set.seed(22)
result <- run(model)
plot(result)

## End(Not run)
```

`nodes` *Example data with spatial distribution of nodes*

Description

Example data to initialize a population of 1600 nodes and demonstrate various models.

Usage

```
data(nodes)
```

Format

A `data.frame`

Examples

```
## Not run:
## 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 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.
result <- run(model)

## 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")
col <- ifelse(infected, "blue", "yellow")
plot(y ~ x, nodes, col = col, pch = 20, cex = 2)

## End(Not run)
```

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

Arguments

`x`	an individual events object of class `SimInf_indiv_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_indiv_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

Determine the number of compartments in a model

Usage

```
n_compartments(model)

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

Arguments

`model`	the `model` object to extract the number of compartments from.

Value

the number of compartments in the model.

Examples

```
## Create an 'SIR' model with 100 nodes, with 99 susceptible,
## 1 infected and 0 recovered in each node.
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)

## Display the number of compartments in the model.
n_compartments(model)
```

`n_generations` *Determine the number of generations*

Description

Determine the number of generations

Usage

```
n_generations(object)

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

Arguments

`object`	the `SimInf_abc` object to determine the number of generations for.

Value

an integer with the number of generations.

`n_nodes` *Determine the number of nodes in a model*

Description

Determine the number of nodes in a model

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`	the `model` object to extract the number of nodes from.

Value

the number of nodes in the model.

Examples

```
## Create an 'SIR' model with 100 nodes, with 99 susceptible,
## 1 infected and 0 recovered in each node.
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)

## Display the number of nodes in the model.
n_nodes(model)
```

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

Description

Determine the number of replicates in a model

Usage

```
n_replicates(model)

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

Arguments

`model`	the `model` object to extract the number of replicates from.

Value

the number of replicates in the model.

Examples

```
## Create an 'SIR' model with 100 nodes, with 99 susceptible,
## 1 infected and 0 recovered in each node.
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)

## Display the number of replicates in the model.
n_replicates(model)
```

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

Description

The number nodes that are connected with *external transfer* events from each node.

Usage

```
outdegree(model)
```

Arguments

`model`	determine out-degree for each node in the model.

Value

vector with out-degree for each node.

Examples

```
## Create an 'SIR' model with 1600 nodes and initialize
## it with example data.
model <- SIR(u0 = u0_SIR(), tspan = 1:1460, events = events_SIR(),
             beta   = 0.16, gamma  = 0.077)

## Display outdegree for each node in the model.
plot(outdegree(model))
```

`package_skeleton` *Create a package skeleton from a* `SimInf_model`

Description

Describe your model in a logical way in R, then `mparse` creates a `SimInf_model` object with your model definition that can be installed as an add-on R package.

Usage

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

Arguments

`model`	The `model` `SimInf_model` object with your model to create the package skeleton from.
`name`	Character string with the package name. It should contain only (ASCII) letters, numbers and dot, have at least two characters and 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 '.' i.e. the current directory.
`author`	Author of the package.
`email`	Email of the package maintainer.
`maintainer`	Maintainer of the package.
`license`	License of the package. Default is 'GPL-3'.

Value

invisible `NULL`.

References

Read the *Writing R Extensions* manual for more details.

Once you have created a *source* package you need to install it: see the *R Installation and Administration* manual, `INSTALL` and `install.packages`.

`pairs,SimInf_model-method`

Scatterplot of number of individuals in each compartment

Description

A matrix of scatterplots with the number of individuals in each compartment is produced. The `ij`th scatterplot contains `x[,i]` plotted against `x[,j]`.

Usage

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

Arguments

`x`	The `model` to plot
`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`) includes all compartments.
`index`	indices specifying the nodes to include when plotting data. Default `index = NULL` include all nodes in the model.
`...`	Additional arguments affecting the plot produced.

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 10 nodes and initialise
## it with 99 susceptible individuals and one infected
## individual. Let the model 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)
```

```
## Run the model and save the result.
result <- run(model)

## Create a scatter plot that includes all compartments in all
## nodes.
pairs(result)

## Create a scatter plot that includes the S and I compartments in
## nodes 1 and 2.
pairs(result, ~S+I, 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. 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

Display the ABC posterior distribution

Usage

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

Arguments

`x`	The `SimInf_abc` object to plot.
`y`	The generation to plot. The default is to display the last generation.
`...`	Additional arguments affecting the plot.

`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_indiv_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_indiv_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 either the median and the quantile range of the counts in all nodes, or plot the counts in specified nodes.

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,
  ...
)
```

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.
`...`	Other graphical parameters that are passed on to the plot function.

Examples

```
## Not run:
## 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 100 nodes and initialise
## it with 990 susceptible individuals and 10 infected
## individuals in each node. Run the model over 100 days.
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 and save the result.
result <- run(model)

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

## Plot the median and the middle 95\
## number of susceptible, infected and recovered individuals.
plot(result, range = 0.95)

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

## Use the formula notation instead to plot the median and
## interquartile range of the number of infected individuals.
plot(result, ~I)

## Plot the number of susceptible, infected
## and recovered individuals in the first
## three nodes.
plot(result, index = 1:3, range = FALSE)

## Use plot type line instead.
plot(result, index = 1:3, range = FALSE, type = "l")

## Plot the number of infected individuals in the first node.
plot(result, "I", index = 1, range = FALSE)

## Plot the proportion of infected individuals (cases)
## in the population.
plot(result, I ~ S + I + R)

## Plot the proportion of nodes with infected individuals.
plot(result, I ~ S + I + R, level = 2)

## Plot the median and interquartile range of the proportion
## of infected individuals in each node
plot(result, I ~ S + I + R, level = 3)

## Plot the proportion of infected individuals in the first
## three nodes.
plot(result, I ~ S + I + R, level = 3, index = 1:3, range = FALSE)

## End(Not run)
```

`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,SimInf_model-method` 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

Particle Markov chain Monte Carlo (PMCMC) algorithm

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 = getOption("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 = getOption("verbose", FALSE)
)
```

Arguments

`model`	The model 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. 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 vector of initial values for the parameters of the model. Default is `NULL`, and then these are sampled from the prior distribution(s).
`covmat`	A named numeric `(npars x npars)` matrix with covariances to use as initial proposal matrix. If left unspecified then defaults to `diag((theta/10)^2/npars)`.
`adaptmix`	Mixing proportion for adaptive proposal. Must be a value between zero and one. Default is `adaptmix = 0.05`.
`adaptive`	Controls when to start adaptive update. Must be greater or equal to zero. If `adaptive=0`, then adaptive update is not performed. Default is `adaptive = 100`.
`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`	An optional function that, if non-NULL, is applied in the particle filter 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.
`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`	An optional chain to start from. Must be a `data.frame` or an object that can be coerced to a `data.frame`. Only the columns in `chain` with a name that matches the names that will be used if this argument is not provided will be used. When this argument is provided, `n_iterations` can be 0.
`verbose`	prints diagnostic messages when `TRUE`. The default is to retrieve the global option `verbose` and use `FALSE` if it is not set. 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`.

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`.

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

Description

Calculate the proportion of individuals with disease in the population, or the proportion of nodes with at least one diseased individual, or the proportion of individuals with disease in 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,SimInf_model-method`

`prevalence,SimInf_model-method` *Calculate prevalence from a model object with trajectory data*

Description

Calculate the proportion of individuals with disease in the population, or the proportion of nodes with at least one diseased individual, or the proportion of individuals with disease in 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

```
## Create an 'SIR' model with 6 nodes and initialize
## it to run over 10 days.
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 to generate a single stochastic trajectory.
result <- run(model)

## Determine the proportion of infected individuals (cases)
## in the population at the time-points in 'tspan'.
prevalence(result, I ~ S + I + R)

## Identical result is obtained with the shorthand 'I~.'
prevalence(result, I ~ .)
```

```
## Determine the proportion of nodes with infected individuals at
## the time-points in 'tspan'.
prevalence(result, I ~ S + I + R, level = 2)

## Determine the proportion of infected individuals in each node
## at the time-points in 'tspan'.
prevalence(result, I ~ S + I + R, level = 3)

## Determine the proportion of infected individuals in each node
## at the time-points in 'tspan' when 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

Extract prevalence from running a particle filter

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

Extract prevalence from the filtered trajectories from a particle Markov chain Monte Carlo algorithm.

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,SimInf_model-method` with the arguments `formula`, `level` and `index` for each iteration.

`punchcard<-` *Set a template for where to record result during a simulation*

Description

Using a sparse result matrix can save a lot of memory if the model contains many nodes and time-points, but where only a few of the data points are of interest for post-processing.

Usage

```
punchcard(model) <- value

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

Arguments

`model`	The `model` to set a template for where to record result.
`value`	A `data.frame` that specify the nodes, time-points and compartments to record the number of individuals at `tspan`. Use `NULL` to reset the model to record the number of inidividuals in each compartment in every node at each time-point in tspan.

Details

Using a sparse result matrix can save a lot of memory if the model contains many nodes and time-points, but where only a few of the data points are of interest for post-processing. To use this feature, a template has to be defined for which data points to record. This is done using a `data.frame` that specifies the time-points (column 'time') and nodes (column 'node') to record the state of the compartments, see 'Examples'. The specified time-points, nodes and compartments must exist in the model, or an error is raised. Note that specifying a template only affects which data-points are recorded for post-processing, it does not affect how the solver simulates the trajectory.

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 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.
result <- run(model)

## Display the trajectory with data for every node at each
## time-point in tspan.
trajectory(result)

## Assume we are only interested in nodes '2' and '4' at the
## time-points '3' and '5'
df <- data.frame(time = c(3, 5, 3, 5),
                 node = c(2, 2, 4, 4),
                 S = c(TRUE, TRUE, TRUE, TRUE),
                 I = c(TRUE, TRUE, TRUE, TRUE),
                 R = c(TRUE, TRUE, TRUE, TRUE))
punchcard(model) <- df
result <- run(model)
trajectory(result)

## We can also specify to record only some of the compartments in
## each time-step.
df <- data.frame(time = c(3, 5, 3, 5),
                 node = c(2, 2, 4, 4),
                 S = c(FALSE, TRUE, TRUE, TRUE),
                 I = c(TRUE, FALSE, TRUE, FALSE),
                 R = c(TRUE, FALSE, TRUE, TRUE))
punchcard(model) <- df
result <- run(model)
trajectory(result)

## A shortcut to specify to record all of the compartments in
## each time-step is to only inlude node and time.
df <- data.frame(time = c(3, 5, 3, 5),
                 node = c(2, 2, 4, 4))
punchcard(model) <- df
result <- run(model)
trajectory(result)

## It is possible to use an empty 'data.frame' to specify
## that no data-points should be recorded for the trajectory.
punchcard(model) <- data.frame()
result <- run(model)
```

```
trajectory(result)

## Use 'NULL' to reset the model to record data for every node at
## each time-point in tspan.
punchcard(model) <- NULL
result <- run(model)
trajectory(result)
```

`run` *Run the SimInf stochastic simulation algorithm*

Description

Run the SimInf stochastic simulation algorithm

Usage

```
run(model, ...)

## S4 method for signature 'SimInf_model'
run(model, solver = c("ssm", "aem"), ...)

## S4 method for signature 'SEIR'
run(model, solver = c("ssm", "aem"), ...)

## S4 method for signature 'SIR'
run(model, solver = c("ssm", "aem"), ...)

## S4 method for signature 'SIS'
run(model, solver = c("ssm", "aem"), ...)

## S4 method for signature 'SISe'
run(model, solver = c("ssm", "aem"), ...)

## S4 method for signature 'SISe3'
run(model, solver = c("ssm", "aem"), ...)

## S4 method for signature 'SISe3_sp'
run(model, solver = c("ssm", "aem"), ...)

## S4 method for signature 'SISe_sp'
run(model, solver = c("ssm", "aem"), ...)

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

Arguments

`model`	The SimInf model to run.
`...`	Additional arguments.
`solver`	Which numerical solver to utilize. Default is 'ssm'.

Value

`SimInf_model` object with result from simulation.

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. 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, 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 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)

## Run the model and save the result.
result <- run(model)

## Plot the proportion 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. A `Date` vector is coerced to a numeric vector as days, 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`	a `data.frame` with the scheduled events, see `SimInf_model`.
`beta`	A numeric vector with the transmission rate from susceptible to infected where 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 in all nodes.
`epsilon`	A numeric vector with the incubation rate from exposed to infected where each node can have a different epsilon value. The vector must have length 1 or `nrow(u0)`. If the vector has length 1, but the model contains more nodes, the epsilon value is repeated in all nodes.
`gamma`	A numeric vector with the recovery rate from infected to recovered where 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 beta value is repeated in all nodes.

Details

The SEIR model contains four compartments; number of susceptible (S), number of exposed (E) (those who have been infected but are not yet infectious), number of infectious (I), and number of recovered (R). Moreover, it has three state transitions,

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

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

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

where β is the transmission rate, ϵ is the incubation rate, γ is the recovery rate, and $N = S + E + I + R$.

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

S The number of sucsceptible in each node

E The number of exposed in each node

I The number of infected in each node

R The number of recovered in each node

Value

A `SimInf_model` of class `SEIR`

Examples

```
## Create a 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 and plot the result.
set.seed(3)
result <- run(model)
plot(result)
```

`SEIR-class` *Definition of the 'SEIR' model*

Description

Class to handle the SEIR `SimInf_model`.

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 `model` to set the select matrix for.
`value`	A matrix.

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_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). The number of threads is initialized when SimInf is first loaded in the R session using optional envioronment variables (see 'Details'). It is also possible to specify the number of threads by calling `set_num_threads`. If the environment variables that affect the number of threads change, then `set_num_threads` must be called again for it to take effect.

Usage

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

Arguments

`threads`	integer with maximum number of threads to use in functions that are parallelized with OpenMP (if available). Default is NULL, i.e. to use all available processors and then check for limits in the environment varibles (see 'Details').

Details

The `omp_get_num_procs()` function is used to determine the number of processors that are available to the device at the time the routine is called. The number of threads is then limited by `omp_get_thread_limit()` and the current values of the environmental variables (if set)

- `Sys.getenv("OMP_THREAD_LIMIT")`
- `Sys.getenv("OMP_NUM_THREADS")`
- `Sys.getenv("SIMINF_NUM_THREADS")`

Additionally, the maximum number of threads can be controlled by the `threads` argument, given that its value is not above any of the limits described above.

Value

The previous value is returned (invisible).

`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 `model` to set the shift matrix `events@N`.
`value`	A matrix.

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)
```

`show,SimInf_abc-method` *Print summary of a `SimInf_abc` object*

Description

Print summary of a `SimInf_abc` object

Usage

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

Arguments

`object`	The `SimInf_abc` object.

Value

`invisible(object)`.

`show,SimInf_events-method` *Brief summary of* `SimInf_events`

Description

Shows the number of scheduled events.

Usage

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

Arguments

`object`	The SimInf_events `object`

Value

None (invisible 'NULL').

`show,SimInf_indiv_events-method` *Print summary of a* `SimInf_indiv_events` *object*

Description

Print summary of a `SimInf_indiv_events` object

Usage

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

Arguments

`object`	The `SimInf_indiv_events` object.

Value

`invisible(object).`

show,SimInf_model-method
Brief summary of `SimInf_model`

Description

Brief summary of `SimInf_model`

Usage

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

Arguments

`object`	The SimInf_model `object`

Value

None (invisible 'NULL').

Examples

```
## 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 and save the result
result <- run(model)

## Brief summary of the result. Note that 'U' and 'V' are
## non-empty after running the model.
result
```

`show,SimInf_pfilter-method` *Brief summary of a* `SimInf_pfilter` *object*

Description

Brief summary of a `SimInf_pfilter` object

Usage

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

Arguments

`object`	The `SimInf_pfilter` object.

Value

`invisible(object)`.

`show,SimInf_pmcmc-method` *Brief summary of a* `SimInf_pmcmc` *object*

Description

Brief summary of a `SimInf_pmcmc` object

Usage

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

Arguments

`object`	The `SimInf_pmcmc` object.

Value

`invisible(object)`.

`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, designed to efficiently handle population demographics and network data. The framework integrates infection dynamics in each subpopulation as continuous-time Markov chains (CTMC) using the Gillespie stochastic simulation algorithm (SSA) and incorporates available data such as births, deaths or movements as scheduled events. A scheduled event is used to modify the state of a subpopulation at a predefined time-point.

Details

The `SimInf_model` is central and provides the basis for the framework. A `SimInf_model` object supplies the state-change matrix, the dependency graph, the scheduled events, and the initial state of the system.

All predefined models in SimInf have a generating function, with the same name as the model, for example `SIR`.

A model can also be created from a model specification using the `mparse` method.

After a model is created, a simulation is started with a call to the `run` method and if execution is successful, it returns a modified `SimInf_model` object with a single stochastic solution trajectory attached to it.

SimInf provides several utility functions to inspect simulated data, for example, `show`, `summary` and `plot`. To facilitate custom analysis, it provides the `trajectory,SimInf_model-method` and `prevalence` methods.

One of our design goal was to make SimInf extendable and enable usage of the numerical solvers from other R extension packages in order to facilitate complex epidemiological research. To support this, SimInf has functionality to generate the required C and R code from a model specification, see `package_skeleton`

Author(s)

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

Authors:

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

`SimInf_abc-class` *Class* `"SimInf_abc"`

Description

Class `"SimInf_abc"`

Slots

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

`priors` A `data.frame` containing the four columns `parameter`, `distribution`, `p1` and `p2`. The column `parameter` gives the name of the parameter referred to in the model. The column `distribution` contains the name of the prior distribution. Valid distributions are 'gamma', 'normal' or 'uniform'. The column `p1` is a numeric vector with the first hyperparameter for each prior: 'gamma') shape, 'lognormal') logmean, 'normal') mean, and 'uniform') lower bound. The column `p2` is a numeric vector with the second hyperparameter for each prior: 'gamma') rate, 'lognormal') standard deviation on the log scale, 'normal') standard deviation, and 'uniform') upper bound.

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

`pars` Index to the parameters in `target`.

`nprop` An integer vector with the number of simulated proposals in each generation.

`fn` A function for calculating the summary statistics for the simulated trajectory and determine the distance for each particle, see `abc` for more details.

`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 $\times$ number of summary statistics $\times$ 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. 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` and `continue_abc`.

`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 *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`	Each row corresponds to one compartment in the model. The non-zero entries in a column indicates the compartments to include in an event. For the *exit*, *internal transfer* and *external transfer* events, a non-zero entry indicate the compartments to sample individuals from. For the *enter* event, all individuals enter first non-zero compartment. `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 to modify the discrete state of individuals in a node at a pre-defined time t.

Slots

`E` Each row corresponds to one compartment in the model. The non-zero entries in a column indicates the compartments to include in an event. For the *exit*, *internal transfer* and *external transfer* events, a non-zero entry indicate the compartments to sample individuals from. For the *enter* event, all individuals enter first non-zero compartment. `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.

`event` Type of event: 0) *exit*, 1) *enter*, 2) *internal transfer*, and 3) *external transfer*. Other values are reserved for future event types and not supported by the current solvers. Integer vector.

`time` Time of when the event occurs i.e., the event is processed when time is reached in the simulation. `time` is an integer vector.

`node` The node that the event operates on. Also the source node for an *external transfer* event. Integer vector. 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. Integer vector. 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 `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 proportionally 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 *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.

`SimInf_indiv_events-class`
Class `"SimInf_indiv_events"`

Description

Class `"SimInf_indiv_events"`

Slots

`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.

`SimInf_model` *Create a* `SimInf_model`

Description

Create a `SimInf_model`

Usage

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

Arguments

`G`	Dependency graph that indicates the transition rates that need to be updated after a given state transition has occured. A non-zero entry in element `G[i, i]` indicates that transition rate `i` needs to be recalculated if the state transition `j` occurs. Sparse matrix ($Nt \times Nt$) of object class `dgCMatrix`.
`S`	Each column corresponds to a transition, and execution of state transition `j` amounts to adding the `S[, j]` to the state vector of the node where the state transition occurred. Sparse matrix ($Nc \times Nt$) of object class `dgCMatrix`.

`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. A `Date` vector is coerced to a numeric vector as days, 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`	A `data.frame` with the scheduled events.
`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.
`gdata`	A numeric vector with global data 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.
`U`	The result matrix with the number of individuals in each disease state in every node ($N_n N_c \times$ `length(tspan)`). `U[, j]` contains the number of individuals in each disease state at `tspan[j]`. `U[1:Nc, j]` contains the state of node `1` at `tspan[j]`. `U[(Nc + 1):(2 * Nc), j]` contains the state of node `2` at `tspan[j]` etc.
`u0`	The initial state vector. Either a matrix ($N_c \times N_n$) or a a `data.frame` with the number of individuals in each compartment in every node.
`v0`	The initial continuous state vector in every node. (`dim(ldata)[1]` $\times N_N$). The continuous state vector is updated by the specific model during the simulation in the post time step function.
`V`	The result matrix for the real-valued continous compartment state (N_n`dim(ldata)[1]` $\times$ `length(tspan)`). `V[, j]` contains the real-valued state of the system at `tspan[j]`.
`E`	Sparse matrix to handle scheduled events, see `SimInf_events`.
`N`	Sparse matrix to handle scheduled events, see `SimInf_events`.
`C_code`	Character vector with optional model C code. If non-empty, the C code is written to a temporary C-file when the `run` method is called. The temporary C-file is compiled and the resulting DLL is dynamically loaded. The DLL is unloaded and the temporary files are removed after running the model.

Value

SimInf_model

`SimInf_model-class` *Class* `"SimInf_model"`

Description

Class to handle data for the `SimInf_model`.

Slots

`G` Dependency graph that indicates the transition rates that need to be updated after a given state transition has occured. A non-zero entry in element `G[i, i]` indicates that transition rate `i` needs to be recalculated if the state transition `j` occurs. Sparse matrix ($Nt \times Nt$) of object class `dgCMatrix`.

`S` Each column corresponds to a state transition, and execution of state transition `j` amounts to adding the `S[, j]` column to the state vector `u[, i]` of node *i* where the transition occurred. Sparse matrix ($Nc \times Nt$) of object class `dgCMatrix`.

`U` The result matrix with the number of individuals in each compartment in every node. `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. Integer matrix ($N_n N_c \times$ `length(tspan)`).

`U_sparse` If the model was configured to write the solution to a sparse matrix (`dgCMatrix`) the `U_sparse` contains the data and `U` is empty. The layout of the data in `U_sparse` is identical to `U`. Please note that `U_sparse` is numeric and `U` is integer.

`V` The result matrix for the real-valued continuous state. `V[, j]` contains the real-valued state of the system at `tspan[j]`. Numeric matrix (N_n`dim(ldata)[1]` $\times$ `length(tspan)`).

`V_sparse` If the model was configured to write the solution to a sparse matrix (`dgCMatrix`) the `V_sparse` contains the data and `V` is empty. The layout of the data in `V_sparse` is identical to `V`.

`ldata` A matrix with local data for the nodes. The 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.

`gdata` A numeric vector with global data 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.

`tspan` A vector of increasing time points where the state of each node is to be returned.

`u0` The initial state vector ($N_c \times N_n$) with the number of individuals in each compartment in every node.

`v0` The initial value for the real-valued continuous state. Numeric matrix (`dim(ldata)[1]` $\times N_n$).

`events` Scheduled events `SimInf_events`

`replicates` Number of replicates of the model.

`C_code` Character vector with optional model C code. If non-empty, the C code is written to a temporary C-file when the `run` method is called. The temporary C-file is compiled and the resulting DLL is dynamically loaded. The DLL is unloaded and the temporary files are removed after running the model.

`SimInf_pfilter-class` *Class* `"SimInf_pfilter"`

Description

Class `"SimInf_pfilter"`

Slots

`model` A `SimInf_model` object with one filtered trajectory attached.

`n_particles` An integer with the number of particles that was used at each timestep.

`loglik` The estimated log likelihood.

`ess` A numeric vector with the effective sample size (ESS). 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.

`SimInf_pmcmc-class` *Class* `"SimInf_pmcmc"`

Description

Class `"SimInf_pmcmc"`

Slots

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

`priors` A `data.frame` containing the four columns `parameter`, `distribution`, `p1` and `p2`. The column `parameter` gives the name of the parameter referred to in the model. The column `distribution` contains the name of the prior distribution. Valid distributions are 'gamma', 'normal' or 'uniform'. The column `p1` is a numeric vector with the first hyperparameter for each prior: 'gamma') shape, 'lognormal') logmean, 'normal') mean, and 'uniform') lower bound. The column `p2` is a numeric vector with the second hyperparameter for each prior: 'gamma') rate, 'lognormal') standard deviation on the log scale, 'normal') standard deviation, and 'uniform') upper bound.

`target` Character vector (`gdata` or `ldata`) that determines if the `pmcmc` method estimates parameters in `model@gdata` or in `model@ldata`.

`pars` Index to the parameters in `target`.

`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` Mixing proportion for adaptive proposal.

`adaptive` Controls when to start adaptive update.

See Also

`pmcmc` and `continue_pmcmc`.

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. A `Date` vector is coerced to a numeric vector as days, 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`	a `data.frame` with the scheduled events, see `SimInf_model`.
`beta`	A numeric vector with the transmission rate from susceptible to infected where 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 in all nodes.
`gamma`	A numeric vector with the recovery rate from infected to recovered where 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 beta value is repeated in all nodes.

Details

The SIR model contains three compartments; number of susceptible (S), number of infectious (I), and number of recovered (R). Moreover, it has 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 $N = S + I + R$.

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

S The number of sucsceptible in each node

I The number of infected in each node

R The number of recovered in each node

Value

A `SimInf_model` of class `SIR`

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 and plot the result.
set.seed(22)
result <- run(model)
plot(result)
```

`SIR-class` *Definition of the* SIR *model*

Description

Class to handle the SIR `SimInf_model`.

Details

The SIR model contains three compartments; number of susceptible (S), number of infectious (I), and number of recovered (R). Moreover, it has 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 $N = S + I + R$.

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 and plot the result.
set.seed(22)
result <- run(model)
plot(result)
```

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. A `Date` vector is coerced to a numeric vector as days, 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`	a `data.frame` with the scheduled events, see `SimInf_model`.
`beta`	A numeric vector with the transmission rate from susceptible to infected where 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 in all nodes.
`gamma`	A numeric vector with the recovery rate from infected to recovered where 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 beta value is repeated in all nodes.

Details

The SIS model contains two compartments; number of susceptible (S), and number of infectious (I). Moreover, it has 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 $N = S + I$.

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

S The number of sucsceptible in each node

I The number of infected in each node

Value

A `SimInf_model` of class `SIS`

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 and plot the result.
set.seed(22)
result <- run(model)
plot(result)
```

`SIS-class` *Definition of the* SIS *model*

Description

Class to handle the SIS `SimInf_model`.

Details

The SIS model contains two compartments; number of susceptible (S), and number of infectious (I). Moreover, it has 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 $N = S + I$.

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 and plot the result.
set.seed(22)
result <- run(model)
plot(result)
```

`SISe` *Create a 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. A `Date` vector is coerced to a numeric vector as days, 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`	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`	The recovery rate from infected to susceptible
`alpha`	Shed rate from infected individuals
`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; number of susceptible (S) and number of infectious (I). Additionally, it contains an environmental compartment to model shedding of a pathogen to the environment. 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,

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

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. The seasonal decay and removal of the pathogen is captured by $\beta(t)$. It is also possible to include a small background infectious pressure ϵ to allow for other indirect sources of environmental contamination. 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`.

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

S The number of sucsceptible in each node

I The number of infected in each node

Value

`SISe`

Beta

The time dependent beta is divided into four intervals of the year

```
where 0 <= day < 365

Case 1: END_1 < END_2 < END_3 < END_4
INTERVAL_1 INTERVAL_2     INTERVAL_3     INTERVAL_4     INTERVAL_1
[0, END_1) [END_1, END_2) [END_2, END_3) [END_3, END_4) [END_4, 365)

Case 2: END_3 < END_4 < END_1 < END_2
INTERVAL_3 INTERVAL_4     INTERVAL_1     INTERVAL_2     INTERVAL_3
[0, END_3) [END_3, END_4) [END_4, END_1) [END_1, END_2) [END_2, 365)

Case 3: END_4 < END_1 < END_2 < END_3
INTERVAL_4 INTERVAL_1     INTERVAL_2     INTERVAL_3     INTERVAL_4
[0, END_4) [END_4, END_1) [END_1, END_2) [END_2, END_3) [END_3, 365)
```

`SISe-class` *Definition of the* `SISe` *model*

Description

Class to handle the SISe `SimInf_model`.

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. A Date vector is coerced to a numeric vector as days, 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	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`	Shed rate from infected individuals
`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; number of susceptible (S_1, S_2, S_3) and number of infectious (I_1, I_2, I_3). Additionally, it contains an environmental compartment to model shedding of a pathogen to the environment. Consequently, the model has 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 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 $\gamma_1, \gamma_2, \gamma_3$, measured per individual and per unit of time. Finally, the environmental infectious pressure in each node is evolved by,

$$\frac{d\varphi(t)}{dt} = \frac{\alpha \left(I_1(t) + I_2(t) + I_3(t)\right)}{N(t)} - \beta(t)\varphi(t) + \epsilon$$

where α is the average shedding rate of the pathogen to the environment per infected individual and $N = S_1 + S_2 + S_3 + I_1 + I_2 + I_3$ the size of the node. The seasonal decay and removal of the pathogen is captured by $\beta(t)$. It is also possible to include a small background infectious pressure ϵ to allow for other indirect sources of environmental contamination. 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`.

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

S_1 The number of sucsceptible in age category 1

I_1 The number of infected in age category 1

S_2 The number of sucsceptible in age category 2

I_2 The number of infected in age category 2

S_3 The number of sucsceptible in age category 3

I_3 The number of infected in age category 3

Value

`SISe3`

Beta

The time dependent beta is divided into four intervals of the year

```
where 0 <= day < 365

Case 1: END_1 < END_2 < END_3 < END_4
INTERVAL_1 INTERVAL_2     INTERVAL_3     INTERVAL_4     INTERVAL_1
[0, END_1) [END_1, END_2) [END_2, END_3) [END_3, END_4) [END_4, 365)

Case 2: END_3 < END_4 < END_1 < END_2
INTERVAL_3 INTERVAL_4     INTERVAL_1     INTERVAL_2     INTERVAL_3
[0, END_3) [END_3, END_4) [END_4, END_1) [END_1, END_2) [END_2, 365)

Case 3: END_4 < END_1 < END_2 < END_3
INTERVAL_4 INTERVAL_1     INTERVAL_2     INTERVAL_3     INTERVAL_4
[0, END_4) [END_4, END_1) [END_1, END_2) [END_2, END_3) [END_3, 365)
```

SISe3-class *Definition of the 'SISe3' model*

Description

Class to handle the SISe3 SimInf_model model.

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. A `Date` vector is coerced to a numeric vector as days, 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`	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`	Shed rate from infected individuals
`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; number of susceptible (S_1, S_2, S_3) and number of infectious (I_1, I_2, I_3). 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 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 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 $\gamma_1, \gamma_2, \gamma_3$, measured per individual and per unit of time. Finally, the environmental infectious pressure in each node is evolved by,

$$\frac{d\varphi_i(t)}{dt} = \frac{\alpha \left(I_{i,1}(t) + I_{i,2}(t) + I_{i,3}(t)\right)}{N_i(t)} + \sum_k \frac{\varphi_k(t)N_k(t) - \varphi_i(t)N_i(t)}{N_i(t)} \cdot \frac{D}{d_{ik}} - \beta(t)\varphi_i(t)$$

where α is the average shedding rate of the pathogen to the environment per infected individual and $N = S_1 + S_2 + S_3 + I_1 + I_2 + I_3$ 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`.

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

S_1 The number of sucsceptible in age category 1

I_1 The number of infected in age category 1

S_2 The number of sucsceptible in age category 2

I_2 The number of infected in age category 2

S_3 The number of sucsceptible in age category 3

I_3 The number of infected in age category 3

Value

`SISe3_sp`

Beta

The time dependent beta is divided into four intervals of the year

```
where 0 <= day < 365

Case 1: END_1 < END_2 < END_3 < END_4
INTERVAL_1 INTERVAL_2     INTERVAL_3     INTERVAL_4     INTERVAL_1
[0, END_1) [END_1, END_2) [END_2, END_3) [END_3, END_4) [END_4, 365)

Case 2: END_3 < END_4 < END_1 < END_2
INTERVAL_3 INTERVAL_4     INTERVAL_1     INTERVAL_2     INTERVAL_3
[0, END_3) [END_3, END_4) [END_4, END_1) [END_1, END_2) [END_2, 365)

Case 3: END_4 < END_1 < END_2 < END_3
INTERVAL_4 INTERVAL_1     INTERVAL_2     INTERVAL_3     INTERVAL_4
[0, END_4) [END_4, END_1) [END_1, END_2) [END_2, END_3) [END_3, 365)
```

`SISe3_sp-class` *Definition of the 'SISe3_sp' model*

Description

Class to handle the SISe3_sp `SimInf_model` model.

`SISe_sp` *Create a `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. A `Date` vector is coerced to a numeric vector as days, 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`	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`	The recovery rate from infected to susceptible
`alpha`	Shed rate from infected individuals
`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`.

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

S The number of sucsceptible

I The number of infected

Value

`SISe_sp`

Beta

The time dependent beta is divided into four intervals of the year

```
where 0 <= day < 365

Case 1: END_1 < END_2 < END_3 < END_4
INTERVAL_1 INTERVAL_2     INTERVAL_3     INTERVAL_4     INTERVAL_1
[0, END_1) [END_1, END_2) [END_2, END_3) [END_3, END_4) [END_4, 365)

Case 2: END_3 < END_4 < END_1 < END_2
INTERVAL_3 INTERVAL_4     INTERVAL_1     INTERVAL_2     INTERVAL_3
[0, END_3) [END_3, END_4) [END_4, END_1) [END_1, END_2) [END_2, 365)
```

```
Case 3: END_4 < END_1 < END_2 < END_3
INTERVAL_4 INTERVAL_1     INTERVAL_2     INTERVAL_3     INTERVAL_4
[0, END_4) [END_4, END_1) [END_1, END_2) [END_2, END_3) [END_3, 365)
```

SISe_sp-class — *Definition of the* SISe_sp *model*

Description

Class to handle the SISe_sp SimInf_model.

summary,SimInf_abc-method — *Detailed summary of a* SimInf_abc *object*

Description

Detailed summary of a SimInf_abc object

Usage

```
## S4 method for signature 'SimInf_abc'
summary(object, ...)
```

Arguments

object	The SimInf_abc object
...	Additional arguments affecting the summary produced.

Value

None (invisible 'NULL').

`summary,SimInf_events-method` *Detailed summary of a* `SimInf_events` *object*

Description

Shows the number of scheduled events and the number of scheduled events per event type.

Usage

```
## S4 method for signature 'SimInf_events'
summary(object, ...)
```

Arguments

`object`	The `SimInf_events` object
`...`	Additional arguments affecting the summary produced.

Value

None (invisible 'NULL').

`summary,SimInf_indiv_events-method` *Detailed summary of a* `SimInf_indiv_events` *object*

Description

Detailed summary of a `SimInf_indiv_events` object

Usage

```
## S4 method for signature 'SimInf_indiv_events'
summary(object, ...)
```

Arguments

`object`	The `SimInf_indiv_events` object
`...`	Additional arguments affecting the summary produced.

Value

None (invisible 'NULL').

`summary,SimInf_model-method` *Detailed summary of a* `SimInf_model` *object*

Description

Detailed summary of a `SimInf_model` object

Usage

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

Arguments

`object`	The `SimInf_model` object
`...`	Additional arguments affecting the summary produced.

Value

None (invisible 'NULL').

`summary,SimInf_pfilter-method` *Detailed summary of a* `SimInf_pfilter` *object*

Description

Detailed summary of a `SimInf_pfilter` object

Usage

```
## S4 method for signature 'SimInf_pfilter'
summary(object, ...)
```

Arguments

`object`	The `SimInf_pfilter` object.
`...`	Unused additional arguments.

Value

`invisible(NULL)`.

`summary,SimInf_pmcmc-method`
Detailed summary of a `SimInf_pmcmc` *object*

Description

Detailed summary of a `SimInf_pmcmc` object

Usage

```
## S4 method for signature 'SimInf_pmcmc'
summary(object, ...)
```

Arguments

`object`	The `SimInf_pmcmc` object
`...`	Not used.

Value

None (invisible 'NULL').

`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,SimInf_model-method`

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. 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.

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)`.

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,SimInf_model-method`).

Value

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

`trajectory,SimInf_pmcmc-method`

Extract filtered trajectories from fitting a PMCMC algorithm

Description

Extract filtered trajectories from a particle Markov chain Monte Carlo algorithm.

Usage

```
## S4 method for signature 'SimInf_pmcmc'
trajectory(model, compartments, index, start = 1, end = NULL, thin = 1)
```

Arguments

`model`	the `SimInf_pmcmc` object to extract the filtered trajectories 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.
`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 `trajectory,SimInf_model-method` with the arguments `compartments` and `index` for each iteration.

`u0` *Get the initial compartment state*

Description

Get the initial compartment state

Usage

```
u0(object, ...)

## S4 method for signature 'SimInf_model'
u0(object, ...)

## S4 method for signature 'SimInf_indiv_events'
u0(object, time = NULL, target = NULL, age = NULL)
```

Arguments

`object`	The object to get the initial compartment state `u0` from.
`...`	Additional arguments.
`time`	Only used when object is of class `SimInf_indiv_events` object. The time-point that will be used to create u0. If left empty (the default), the earliest time among the events will be used.
`target`	Only used when object is of class `SimInf_indiv_events` object. The SimInf model ('SEIR', 'SIR', 'SIS', 'SISe3', 'SISe3_sp', 'SISe', or 'SISe_sp') to target the events and u0 for. The default, `NULL`, creates an `u0`, but where the compartments might have to be renamed and post-processed to fit the specific use case.

`age`	Only used when object is of class `SimInf_indiv_events` object. An integer vector with break points in days for the ageing events. The default, `NULL`, creates an `u0` where all individuals belong to the same age category.

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)
```

`u0<-` *Update the initial compartment state u0 in each node*

Description

Update the initial compartment state u0 in each node

Usage

```
u0(model) <- value

## S4 replacement method for signature 'SimInf_model'
u0(model) <- value
```

Arguments

`model`	The model to update the initial compartment state `u0`.
`value`	A `data.frame` with the initial state in each node. Each row is one node, and the number of rows in `u0` must match the number of nodes in `model`. Only the columns in `u0` with a name that matches a compartment in the `model` will be used.

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 and plot the result.
set.seed(22)
result <- run(model)
plot(result)

## Update u0 and run the model again
u0(model) <- data.frame(S = 990, I = 10, R = 0)
result <- run(model)
plot(result)
```

---

u0_SEIR *Example data to initialize the 'SEIR' model*

Description

Example data to initialize a population of 1600 nodes and demonstrate the SEIR model.

Usage

```
u0_SEIR()
```

Details

A `data.frame` with the number of individuals in the 'S', 'E', 'I' and 'R' compartments in 1600 nodes. Note that the 'E', 'I' and 'R' compartments are zero.

Value

A `data.frame`

Examples

```
## Not run:
## 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 'SEIR' model with 1600 nodes and initialize it to
## run over 4*365 days and record data at weekly time-points.
## Add ten infected individuals to the first node.
u0 <- u0_SEIR()
u0$I[1] <- 10
tspan <- seq(from = 1, to = 4*365, by = 7)
model <- SEIR(u0      = u0,
              tspan   = tspan,
              events  = events_SEIR(),
              beta    = 0.16,
              epsilon = 0.25,
              gamma   = 0.01)

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

## Summarize trajectory
summary(result)

## End(Not run)
```

---

u0_SIR *Example data to initialize the 'SIR' model*

Description

Example data to initialize a population of 1600 nodes and demonstrate the `SIR` model.

Usage

```
u0_SIR()
```

Details

A `data.frame` with the number of individuals in the 'S', 'I' and 'R' compartments in 1600 nodes. Note that the 'I' and 'R' compartments are zero.

Value

A `data.frame`

Examples

```
## Not run:
## 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 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  = 0.01)

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

## Summarize trajectory
summary(result)

## End(Not run)
```

---

u0_SIS *Example data to initialize the 'SIS' model*

Description

Example data to initialize a population of 1600 nodes and demonstrate the `SIS` model.

Usage

```
u0_SIS()
```

Details

A `data.frame` with the number of individuals in the 'S', and 'I' compartments in 1600 nodes. Note that the 'I' compartment is zero.

Value

A `data.frame`

Examples

```
## Not run:
## 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 nodes and initialize
## it to run over 4*365 days. Add one infected individual
## to the first node.
u0 <- u0_SIS()
u0$I[1] <- 1
tspan <- seq(from = 1, to = 4*365, by = 1)
model <- SIS(u0     = u0,
             tspan  = tspan,
             events = events_SIS(),
             beta   = 0.16,
                 gamma  = 0.01)

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

## Summarize trajectory
summary(result)

## End(Not run)
```

u0_SISe   *Example data to initialize the 'SISe' model*

Description

Example data to initialize a population of 1600 nodes and demonstrate the SISe model.

Usage

```
u0_SISe()
```

Details

A data.frame with the number of individuals in the 'S' and 'I' compartments in 1600 nodes. Note that the 'I' compartment is zero.

Value

A data.frame

Examples

```
## Not run:
## 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 'SISe' model with 1600 nodes and initialize it to
## run over 4*365 days and record data at weekly time-points.

## Load the initial population and add ten infected individuals to
## the first node.
u0 <- u0_SISe()
u0$I[1] <- 10

## Define 'tspan' to run the simulation over 4*365 and record the
## state of the system at weekly time-points.
```

```
tspan <- seq(from = 1, to = 4*365, by = 7)

## Load scheduled events for the population of nodes with births,
## deaths and between-node movements of individuals.
events <- events_SISe()

## Create an 'SISe' model
model <- SISe(u0 = u0, tspan = tspan, events = events_SISe(),
              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 = 91, end_t2 = 182,
              end_t3 = 273, end_t4 = 365, epsilon = 0)

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

## Summarize trajectory
summary(result)

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

## End(Not run)
```

`u0_SISe3` *Example data to initialize the ‘SISe3’ model*

Description

Example data to initialize a population of 1600 nodes and demonstrate the `SISe3` model.

Usage

```
data(u0_SISe3)
```

Format

A `data.frame`

Details

A `data.frame` with the number of individuals in the ‘S_1’, ‘S_2’, ‘S_3’, ‘I_1’, ‘I_2’ and ‘I_3’ compartments in 1600 nodes. Note that the ‘I_1’, ‘I_2’ and ‘I_3’ compartments are zero.

Examples

```
## Not run:
## 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 nodes and initialize it to
## run over 4*365 days and record data at weekly time-points.

## Load the initial population and add ten infected individuals to
## I_1 in the first node.
u0 <- u0_SISe3
u0$I_1[1] <- 10

## Define 'tspan' to run the simulation over 4*365 and record the
## state of the system at weekly time-points.
tspan <- seq(from = 1, to = 4*365, by = 7)

## Load scheduled events for the population of nodes with births,
## deaths and between-node movements of individuals.
events <- events_SISe3

## Create a 'SISe3' model
model <- SISe3(u0 = u0, tspan = tspan, events = events,
               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)

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

## Summarize trajectory
summary(result)

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

## End(Not run)
```

`v0<-` *Update the initial continuous state v0 in each node*

Description

Update the initial continuous state v0 in each node

Usage

```
v0(model) <- value

## S4 replacement method for signature 'SimInf_model'
v0(model) <- value
```

Arguments

`model`	The model to update the initial continuous state `v0`.
`value`	the initial continuous state in each node. Must be a `data.frame` or an object that can be coerced to a `data.frame`. A named numeric vector will be coerced to a one-row `data.frame`. Each row is one node, and the number of rows in `v0` must match the number of nodes in `model`. Only the columns in `v0` with a name that matches a continuous state in `v0` in the `model` will be used

Examples

```
## Create an 'SISe' model with no infected individuals and no
## infectious pressure (phi = 0, epsilon = 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.
set.seed(22)
result <- run(model)
plot(result)

## Update the infectious pressure 'phi' in 'v0' and run
## the model again.
v0(model) <- data.frame(phi = 1)
result <- run(model)
plot(result)
```