Package 'EpiModel'

November 3, 2015

Version 1.2.2

Date 2015-11-02

Title Mathematical Modeling of Infectious Disease

Description Tools for simulating mathematical models of infectious disease. Epidemic model classes include deterministic compartmental models, stochastic individual contact models, and stochastic network models. Disease types include SI, SIR, and SIS epidemics with and without demography, with tools available for expansion to model complex epidemic processes.

Maintainer Samuel Jenness <sjenness@uw.edu>

License GPL-2

URL http://epimodel.org/

BugReports https://github.com/statnet/EpiModel/issues

Depends R (>= 3.1), deSolve (>= 1.11), networkDynamic (>= 0.7.1), tergm (>= 3.2.4)

Imports graphics, grDevices, stats, utils, doParallel, ergm (>= 3.2.4), foreach, network (>= 1.12.0), RColorBrewer, ape

Suggests knitr, ndtv (>= 0.7), shiny, testthat

VignetteBuilder knitr

RoxygenNote 5.0.0.9000

NeedsCompilation no

Author Samuel Jenness [cre, aut],
Steven M. Goodreau [aut],
Martina Morris [aut],
Emily Beylerian [ctb],
Li Wang [ctb],
Skye Bender-deMoll [ctb]

Repository CRAN

Date/Publication 2015-11-03 11:59:50

R topics documented:

EpiModel-package *Mathematical Modeling of Infectious Disease*

Description

Package:	EpiModel
Type:	Package
Version:	1.2.2
Date:	2015-11-02
License:	GPL (>= 2)
LazyLoad:	yes

Details

The EpiModel package provides functions for building, solving, and plotting mathematical models of infectious disease. The goals of the package are to provide basic tools for modeling in multiple frameworks for pedagogical purposes, and to support users in developing and expanding these tools using the package's utility functions for their own research.

Model Classes and Types

EpiModel currently provides functionality for three classes of epidemic models:

- **Deterministic Compartmental Models:** these continuous-time models are solved using ordinary differential equations. EpiModel allows for easy specification of sensitivity models to compare multiple runs of the same model with different parameter values.
- **Stochastic Individual Contact Models:** a novel class of microsimulation models were developed to mirror the deterministic models but add random variation in all components of the transmission dynamics system, from infection to recovery to vital dynamics (births and deaths).
- **Stochastic Network Models:** using the underlying statistical framework of dynamic exponential random graph models (ERGMs) recently developed in the **Statnet** suite of software in R, our network models simulate partnership formation and dissolution stochastically according to a user-defined statistical model, as well as disease spread on that network.

EpiModel supports three infectious disease types to be run across all of the three classes:

- **Susceptible-Infectious (SI):** a two-state disease in which there is life-long infection without recovery. HIV/AIDS is one example, although for this case it is more common to model infection stages as separate compartments (forthcoming in a future release).
- **Susceptible-Infectious-Recovered (SIR):** a three-stage disease in which one has life-long recovery with immunity after infection. Measles is one example, but modern models for the disease also require consideration of vaccination patterns in the population (forthcoming in a future release).

- **Susceptible-Infectious-Susceptible (SIS):** a two-stage disease in which one may transition back and forth from the susceptible to infected states throughout life. Examples include bacterial sexually transmitted diseases like gonorrhea.

Model Parameterization and Simulation

EpiModel uses three model setup functions for each model class to input the necessary parameters, initial conditions, and control settings:

- `param.dcm`, `param.icm`, and `param.net` are used to input epidemic parameters for each of the three model classes. Parameters include the rate of contacts or acts between actors, the transmission per contact, and recovery and demographic rates for models that include those transitions.
- `init.dcm`, `init.icm`, and `init.net` are used to input the initial conditions for each class. The main conditions are limited to the numbers or, if applicaple, the specific agents in the population who are infected or recovered at the simulation outset.
- `control.dcm`, `control.icm`, and `control.net` are used to specify the remaining control settings for each simulation. The core controls for built-in model types include the disease type, number of time steps, and number of simulations. Controls are also used to input new model functions (for DCMs) and new model modules (for ICMs and network models) to allow the user to simulate fully original epidemic models in EpiModel. See the extensive documention linked on the control function help pages for further details.

With the models parameterized, the functions for simulating the epidemic model are:

- `dcm` for deterministic compartmental epidemic models.
- `icm` for individual contact epidemic models.
- Network models are simulated in a three-step process:
  1. `netest` estimates the statistical models underlying partnership formation and dissolution. This function is a wrapper around the `ergm` and `stergm` functions in the `ergm` and `tergm` packages. The current framework for model simulation is one in which key target statistics guiding the formation formula from an egocentric sample of the population is passed along with an estimate of the average duration of edges in the network.
  2. `netdx` runs diagnostics on the dynamic model fit by simulating the base network over time to ensure the model fits the targets for formation and dissolution.
  3. `netsim` simulates the stochastic network epidemic models, with a given network model fit in `netest`. Here the function requires this model fit object along with the parameters, initial conditions, and control settings as defined above.

References

The main website is at `http://epimodel.org/`, and the source code is at `http://github.com/statnet/EpiModel`. Bug reports and feature requests may be filed there.

as.data.frame.dcm *Extract Model Data for Deterministic Compartmental Models*

Description

This function extracts a model run from an object of class dcm into a data frame using the generic as.data.frame function.

Usage

```
## S3 method for class 'dcm'
as.data.frame(x, row.names = NULL, optional = FALSE,
  run = 1, ...)
```

Arguments

x	An EpiModel object of class dcm.
row.names	See as.data.frame.default.
optional	See as.data.frame.default.
run	Run number for model; used for multiple-run sensitivity models.
...	See as.data.frame.default.

Details

Model output from a dcm simulation are available as a data frame with this helper function. The output data frame will include columns for time, the size of each compartment, the overall population size (the sum of compartment sizes), and the size of each flow.

Examples

```
## Example 1: One-group SIS model with varying act.rate
param <- param.dcm(inf.prob = 0.2, act.rate = seq(0.05, 0.5, 0.05),
                  rec.rate = 1/50)
init <- init.dcm(s.num = 500, i.num = 1)
control <- control.dcm(type = "SIS", nsteps = 500)
mod1 <- dcm(param, init, control)
head(as.data.frame(mod1, run = 1))
head(as.data.frame(mod1, run = 10))

## Example 2: Two-group SIR model with vital dynamics
param <- param.dcm(inf.prob = 0.2, inf.prob.g2 = 0.1,
                   act.rate = 3, balance = "g1",
                   rec.rate = 1/50, rec.rate.g2 = 1/50,
                   b.rate = 1/100, b.rate.g2 = NA,
                   ds.rate = 1/100, ds.rate.g2 = 1/100,
                   di.rate = 1/90, di.rate.g2 = 1/90,
                   dr.rate = 1/100, dr.rate.g2 = 1/100)
init <- init.dcm(s.num = 500, i.num = 1, r.num = 0,
```

```
                 s.num.g2 = 500, i.num.g2 = 1, r.num.g2 = 0)
control <- control.dcm(type = "SIR", nsteps = 500)
mod2 <- dcm(param, init, control)
head(as.data.frame(mod2))
tail(as.data.frame(mod2))
```

`as.data.frame.icm` *Extract Model Data for Stochastic Models*

Description

This function extracts model simulations for objects of classes `icm` and `netsim` into a data frame using the generic `as.data.frame` function.

Usage

```
## S3 method for class 'icm'
as.data.frame(x, row.names = NULL, optional = FALSE, sim,
  out = "mean", qval, ...)

## S3 method for class 'netsim'
as.data.frame(x, row.names = NULL, optional = FALSE, sim,
  out = "mean", ...)
```

Arguments

`x`	An `EpiModel` object of class `icm` or `netsim`.
`row.names`	See `as.data.frame.default`.
`optional`	See `as.data.frame.default`.
`sim`	Simulation number from model; used only if more than 1 simulation and `out="vals"`.
`out`	Data output to data frame: `"mean"` for row means across simulations, `"sd"` for row standard deviations across simulations, `"qnt"` for row quantiles at the level specified in `qval`, or `"vals"` for values from one specific simulation (with simulation number set with `sim` argument).
`qval`	Quantile value necessary when `out="qnt"`.
`...`	See `as.data.frame.default`.

Details

These methods work for both `icm` and `netsim` class models. The available output includes time-specific means, standard deviations, quantiles, and simulation values (compartment and flow sizes from one simulation) from these stochastic model classes. Means and standard deviations are calculated by taking the row summary across all simulations for each time step in the model output.

Examples

```
## Stochastic ICM SIS model with 5 simulations
param <- param.icm(inf.prob = 0.8, act.rate = 2, rec.rate = 0.1)
init <- init.icm(s.num = 500, i.num = 1)
control <- control.icm(type = "SIS", nsteps = 25,
                       nsims = 5, verbose = FALSE)
mod <- icm(param, init, control)

# Default output is mean across simulations
as.data.frame(mod)

# Standard deviations of simulations
as.data.frame(mod, out = "sd")

# Quantile values for interquartile interval
as.data.frame(mod, out = "qnt", qval = 0.25)
as.data.frame(mod, out = "qnt", qval = 0.75)

# Individual simulation runs, with default sim=1
as.data.frame(mod, out = "vals")
as.data.frame(mod, out = "vals", sim = 2)
```

---

as.network.transmat *Converts transmat infection tree into a network object*

Description

Converts the edges of the infection tree described in the transmat object into a network object, copying in appropriate edge attributes for 'at', 'infDur', 'transProb', 'actRate', and 'finalProb' and constructing a vertex attribute for 'at'.

Usage

```
## S3 method for class 'transmat'
as.network(x, ...)
```

Arguments

x	an object of class transmat to be converted into a network object
...	unused

`as.phylo.transmat` *Convert transmat infection tree into a phylo object*

Description

Converts the edgelist matrix in the transmat object into a phylo object by doing the required re-ordering and labeling.

Usage

```
## S3 method for class 'transmat'
as.phylo(x, collapse.singles = TRUE, ...)
```

Arguments

`x`	An object of class `"transmat"`, the output from `get_transmat`.
`collapse.singles`	logical, (default TRUE) should `collapse.singles` be called on the phylo object before it is returned? (many infection trees contain intermediate nodes that must be removed to be a proper phylo tree)
`...`	further arguments (unused)

Details

Converts the infection timing into elapsed time from parents' infections to be appropriate for the `edge.length` component. If the the tree does not have the appropriate structure to be a phylogenetic tree (chains of multiple vertices with no branches) the branches will be collapsed (depending on `collapse.singles`) and labeled with the latest vertex in the chain. Does not yet support infection trees with multiple sources.

Examples

```
set.seed(10)
nw <- network.initialize(n = 100, directed = FALSE)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)

est1 <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

param <- param.net(inf.prob = 0.5)
init <- init.net(i.num = 1)
control <- control.net(type = "SI", nsteps = 40, nsims = 1, verbose = FALSE,
                       use.pids = FALSE)

mod1 <- netsim(est1, param, init, control)
tm <- get_transmat(mod1)
tmPhylo <- as.phylo.transmat(tm)
```

```
plot(tmPhylo, show.node.label = TRUE, cex = 0.7)
```

`calc_eql` *Calculate Equilibrium for Infection Prevalence*

Description

Calculates the relative change in infection prevalence across a time series of an epidemic model to assess equilibrium.

Usage

```
calc_eql(x, numer = "i.num", denom = "num", nsteps, threshold = 0.001,
  digits = 4, invisible = FALSE)
```

Arguments

`x`	An `EpiModel` object of class `dcm`, `icm`, or `netsim`.
`numer`	Numerator for prevalence calculation.
`denom`	Denominator for prevalence calculation.
`nsteps`	Number of time steps at end of model simulation to calculate equilibrium as the absolute value of the difference between the minimum prevalence value and the maximum prevalence value over that time range.
`threshold`	Threshold value for determining equilibrium.
`digits`	Number of digits to round for table output.
`invisible`	If `TRUE`, function will suppress output to console and return summary statistics invisibly.

Details

This function calculates whether equilibrium in disease prevalence, or any other fraction of two compartments contained in an epidemic model, have reach equilibrium over a time series. Equilibrium is calculated as the absolute value of the difference of the maximum prevalence and minimum prevalence over a specified time series. That time series is specified as the final `nsteps` time steps of an epidemic model. A larger `nsteps` specification will therefore calculate differences over a longer time series.

Examples

```
# Calculate equilibrium for a DCM
param <- param.dcm(inf.prob = 0.2, inf.prob.g2 = 0.1, act.rate = 0.5,
                   balance = "g1", rec.rate = 1 / 50, rec.rate.g2 = 1 / 50,
                   b.rate = 1 / 100, b.rate.g2 = NA, ds.rate = 1 / 100,
                   ds.rate.g2 = 1 / 100, di.rate = 1 / 90,
                   di.rate.g2 = 1 / 90)
```

```
init <- init.dcm(s.num = 500, i.num = 1,
                 s.num.g2 = 500, i.num.g2 = 1)
control <- control.dcm(type = "SIS", nsteps = 500, verbose = FALSE)
x <- dcm(param, init, control)
plot(x)

# Different calculation options
calc_eql(x, nsteps = 100)
calc_eql(x, nsteps = 250)
calc_eql(x, nsteps = 100, numer = "i.num.g2", denom = "num.g2")
calc_eql(x, nsteps = 100, numer = "i.num.g2", denom = "num.g2",
         threshold = 0.00001)
```

`check_bip_degdist` *Check Degree Distribution for Bipartite Target Statistics*

Description

Checks for consistency in the implied network statistics of a bipartite network in which the mode size and mode-specific degree distributions are specified.

Usage

```
check_bip_degdist(num.m1, num.m2, deg.dist.m1, deg.dist.m2)
```

Arguments

`num.m1`	Number of nodes in mode 1.
`num.m2`	Number of nodes in mode 2.
`deg.dist.m1`	Vector with fractional degree distribution for mode 1.
`deg.dist.m2`	Vector with fractional degree distribution for mode 2.

Details

This function outputs the number of nodes of degree 0 to m, where m is the length of a fractional degree distribution vector, given that vector and the size of the mode. This utility is used to check for balance in implied degree given that fractional distribution within bipartite network simulations, in which the degree-constrained counts must be equal across modes.

See Also

For a detailed explanation of this function, see the tutorial: EpiModel Network Utility Functions.

Examples

```
# An imbalanced distribution
check_bip_degdist(num.m1 = 500, num.m2 = 500,
                  deg.dist.m2 = c(0.40, 0.55, 0.03, 0.02),
                  deg.dist.m1 = c(0.48, 0.41, 0.08, 0.03))

# A balanced distribution
check_bip_degdist(num.m1 = 500, num.m2 = 500,
                  deg.dist.m1 = c(0.40, 0.55, 0.04, 0.01),
                  deg.dist.m2 = c(0.48, 0.41, 0.08, 0.03))
```

---

color_tea *Creates a TEA Variable for Infection Status for* ndtv *Animations*

Description

Creates a new color-named temporally-extended attribute (TEA) variable in a networkDynamic object containing a disease status TEA in numeric format.

Usage

```
color_tea(nd, old.var = "testatus", old.sus = "s", old.inf = "i",
  old.rec = "r", new.var = "ndtvcol", new.sus, new.inf, new.rec,
  verbose = TRUE)
```

Arguments

nd	An object of class networkDynamic.
old.var	Old TEA variable name.
old.sus	Status value for susceptible in old TEA variable.
old.inf	Status value for infected in old TEA variable.
old.rec	Status value for recovered in old TEA variable.
new.var	New TEA variable name to be stored in networkDynamic object.
new.sus	Status value for susceptible in new TEA variable.
new.inf	Status value for infected in new TEA variable.
new.rec	Status value for recovered in new TEA variable.
verbose	Print progress to console.

Details

The ndtv package (http://cran.r-project.org/package=ndtv) produces animated visuals for dynamic networks with evolving edge structures and nodal attributes. Nodal attribute dynamics in ndtv movies require a temporally extended attribute (TEA) containing a standard R color for each node at each time step. By default, the EpiModel package uses TEAs to store disease status history in network model simulations run in netsim. But that status TEA is in numeric format (0, 1, 2). The color_tea function transforms those numeric values of that disease status TEA into a TEA with color values in order to visualize status changes in ndtv.

The convention in plot.netsim is to color the susceptible nodes as blue, infected nodes as red, and recovered nodes as green. Alternate colors may be specified using the new.sus, new.inf, and new.rec parameters, respectively.

Using the color_tea function with a netsim object requires that TEAs for disease status be used and that the networkDynamic object be saved in the output: both tea.status and save.network must be set to TRUE in control.net.

See Also

netsim and the ndtv package documentation.

---

comp_plot *Plot Compartment Diagram for Epidemic Models*

Description

Plots a compartment flow diagram for deterministic compartmental models, stochastic individual contact models, and stochastic network models.

Usage

```
comp_plot(x, at, digits, ...)

## S3 method for class 'dcm'
comp_plot(x, at = 1, digits = 3, run = 1, ...)

## S3 method for class 'icm'
comp_plot(x, at = 1, digits = 3, ...)

## S3 method for class 'netsim'
comp_plot(x, at = 1, digits = 3, ...)
```

Arguments

x	An EpiModel object of class dcm, icm, or netsim.
at	Time step for model statistics.
digits	Number of significant digits to print.

...	Additional arguments passed to plot (not currently used).
run	Model run number, for dcm class models with multiple runs (sensitivity analyses).

Details

The comp_plot function provides a visual summary of an epidemic model at a specific time step. The information contained in comp_plot is the same as in the summary functions for a model, but presented graphically as a compartment flow diagram.

For dcm class plots, specify the model run number if the model contains multiple runs, as in a sensitivity analysis. For icm and netsim class plots, the run argument is not used; the plots show the means and standard deviations across simulations at the specified time step.

These plots are currently limited to one-group and one-mode models for each of the three model classes. That functionality may be expanded in future software releases.

Examples

```
## Example 1: DCM SIR model with varying act.rate
param <- param.dcm(inf.prob = 0.2, act.rate = 5:7,
                   rec.rate = 1/3, b.rate = 1/90, ds.rate = 1/100,
                   di.rate = 1/35, dr.rate = 1/100)
init <- init.dcm(s.num = 1000, i.num = 1, r.num = 0)
control <- control.dcm(type = "SIR", nsteps = 25, verbose = FALSE)
mod1 <- dcm(param, init, control)
comp_plot(mod1, at = 25, run = 3)

## Example 2: ICM SIR model with 3 simulations
param <- param.icm(inf.prob = 0.2, act.rate = 3, rec.rate = 1/50,
                   b.rate = 1/100, ds.rate = 1/100,
                   di.rate = 1/90, dr.rate = 1/100)
init <- init.icm(s.num = 500, i.num = 1, r.num = 0)
control <- control.icm(type = "SIR", nsteps = 25,
                       nsims = 3, verbose = FALSE)
mod2 <- icm(param, init, control)
comp_plot(mod2, at = 25, digits = 1)
```

---

control.dcm *Control Settings for Deterministic Compartmental Models*

Description

Sets the controls for deterministic compartmental models simulated with dcm.

Usage

```
control.dcm(type, nsteps, dt = 1, odemethod = "rk4", dede = FALSE,
  new.mod = NULL, sens.param = TRUE, print.mod = FALSE, verbose = TRUE,
  ...)
```

Arguments

type	Disease type to be modeled, with the choice of "SI" for Susceptible-Infected diseases, "SIR" for Susceptible-Infected-Recovered diseases, and "SIS" for Susceptible-Infected-Susceptible diseases.
nsteps	Number of time steps to solve the model over. This must be a positive integer.
dt	Time unit for model solutions, with the default of 1. Model solutions for fractional time steps may be obtained by setting this to a number between 0 and 1.
odemethod	Ordinary differential equation (ODE) integration method, with the default of the "Runge-Kutta 4" method (see ode for other options).
dede	If TRUE, use the delayed differential equation solver, which allows for time-lagged variables.
new.mod	If not running a built-in model type, a function with a new model to be simulated (see details).
sens.param	If TRUE, evaluate arguments in parameters with length greater than 1 as sensitivity analyses, with one model run per value of the parameter. If FALSE, one model will be run with parameters of arbitrary length.
print.mod	If TRUE, print the model form to the console.
verbose	If TRUE, print model progress to the console.
...	additional control settings passed to model.

Details

control.dcm sets the required control settings for any deterministic compartmental models solved with the dcm function. Controls are required for both built-in model types and original models. For an overview of control settings for built-in DCM class models, consult the Basic DCMs tutorial. For all built-in models, the type argument is a necessary parameter and it has no default.

New Model Functions

The form of the model function for built-in models may be displayed with the print.mod argument set to TRUE. In this case, the model will not be run. These model forms may be used as templates to write original model functions.

These new models may be input and solved with dcm using the new.mod argument, which requires as input a model function. Details and examples are found in the Solving New DCMs tutorial.

See Also

Use param.dcm to specify model parameters and init.dcm to specify the initial conditions. Run the parameterized model with dcm.

control.icm *Control Settings for Stochastic Individual Contact Models*

Description

Sets the controls for stochastic individual contact models simulated with icm.

Usage

```
control.icm(type, nsteps, nsims = 1, rec.rand = TRUE, b.rand = TRUE,
  d.rand = TRUE, initialize.FUN = initialize.icm,
  infection.FUN = infection.icm, recovery.FUN = recovery.icm,
  deaths.FUN = deaths.icm, births.FUN = births.icm,
  get_prev.FUN = get_prev.icm, verbose = TRUE, verbose.int = 0,
  skip.check = FALSE, ...)
```

Arguments

type	Disease type to be modeled, with the choice of "SI" for Susceptible-Infected diseases, "SIR" for Susceptible-Infected-Recovered diseases, and "SIS" for Susceptible-Infected-Susceptible diseases.
nsteps	Number of time steps to solve the model over. This must be a positive integer.
nsims	Number of simulations to run.
rec.rand	If TRUE, use a stochastic recovery model, with the number of recovered at each time step a function of random draws from a binomial distribution with the probability equal to rec.rate. If FALSE, then a deterministic rounded count of the expectation implied by that rate.
b.rand	If TRUE, use a stochastic birth model, with the number of births at each time step a function of random draws from a binomial distribution with the probability equal to the governing birth rates. If FALSE, then a deterministic rounded count of the expectation implied by those rates.
d.rand	If TRUE, use a stochastic death model, with the number of deaths at each time step a function of random draws from a binomial distribution with the probability equal to the governing death rates. If FALSE, then a deterministic rounded count of the expectation implied by those rates.
initialize.FUN	Module to initialize the model at the outset, with the default function of initialize.icm.
infection.FUN	Module to simulate disease infection, with the default function of infection.icm.
recovery.FUN	Module to simulate disease recovery, with the default function of recovery.icm.
deaths.FUN	Module to simulate deaths or exits, with the default function of deaths.icm.
births.FUN	Module to simulate births or entries, with the default function of births.icm.
get_prev.FUN	Module to calculate disease prevalence at each time step, with the default function of get_prev.icm.
verbose	If TRUE, print model progress to the console.

verbose.int	Time step interval for printing progress to console, where 0 (the default) prints completion status of entire simulation and positive integer x prints progress after each x time steps.
skip.check	If TRUE, skips the error check for parameter values, initial conditions, and control settings before running the models. This is suggested only if encountering unnecessary errors when running new models.
...	Additional control settings passed to model.

Details

control.icm sets the required control settings for any stochastic individual contact model solved with the icm function. Controls are required for both built-in model types and when passing original process modules. For an overview of control settings for built-in ICM class models, consult the Basic ICMs tutorial. For all built-in models, the type argument is a necessary parameter and it has no default.

New Modules

Built-in ICM models use a set of module functions that specify how the individual agents in the population are subjected to infection, recovery, demographics, and other processes. Core modules are those listed in the .FUN arguments. For each module, there is a default function used in the simulation. The default infection module, for example, is contained in the infection.icm function.

For original models, one may substitute replacement module functions for any the default functions. New modules may be added to the workflow at each time step by passing a module function via the ... argument.

See Also

Use param.icm to specify model parameters and init.icm to specify the initial conditions. Run the parameterized model with icm.

---

control.net *Control Settings for Stochastic Network Models*

Description

Sets the controls for stochastic network models simulated with netsim.

Usage

```
control.net(type, nsteps, start = 1, nsims = 1, depend, rec.rand = TRUE,
  b.rand = TRUE, d.rand = TRUE, tea.status = TRUE, attr.rules, epi.by,
  use.pids = TRUE, pid.prefix, initialize.FUN = initialize.net,
  deaths.FUN = deaths.net, births.FUN = births.net,
  recovery.FUN = recovery.net, edges_correct.FUN = edges_correct,
  resim_nets.FUN = resim_nets, infection.FUN = infection.net,
  get_prev.FUN = get_prev.net, verbose.FUN = verbose.net,
```

```
module.order = NULL, set.control.stergm, save.nwstats = TRUE,
nwstats.formula = "formation", delete.nodes = FALSE,
save.transmat = TRUE, save.network = TRUE, save.other, verbose = TRUE,
verbose.int = 1, skip.check = FALSE, ...)
```

Arguments

type	Disease type to be modeled, with the choice of "SI" for Susceptible-Infected diseases, "SIR" for Susceptible-Infected-Recovered diseases, and "SIS" for Susceptible-Infected-Susceptible diseases.
nsteps	Number of time steps to simulate the model over. This must be a positive integer.
start	For dependent simulations, time point to start up simulation.
nsims	The total number of disease simulations.
depend	If TRUE, resimulate the network at each time step. This occurs by default with two varieties of dependent models: if there are any vital dynamic parameters in the model, or if the network model formation formula includes the "status" attribute.
rec.rand	If TRUE, use a stochastic recovery model, with the number of recovered at each time step a function of random draws from a binomial distribution with the probability equal to rec.rate. If FALSE, then a deterministic rounded count of the expectation implied by that rate.
b.rand	If TRUE, use a stochastic birth model, with the number of births at each time step a function of random draws from a binomial distribution with the probability equal to the governing birth rates. If FALSE, then a deterministic rounded count of the expectation implied by those rates.
d.rand	If TRUE, use a stochastic death model, with the number of deaths at each time step a function of random draws from a binomial distribution with the probability equal to the governing death rates. If FALSE, then a deterministic rounded count of the expectation implied by those rates.
tea.status	If TRUE, use a temporally extended attribute (TEA) to store disease status. A TEA is needed for plotting static networks at different time steps and for animating dynamic networks with evolving status. TEAs are computationally inefficient for large simulations and should be toggled off in those cases. This argument automatically set to FALSE if delete.nodes=TRUE.
attr.rules	A list containing the rules for setting the attributes of incoming nodes, with one list element per attribute to be set (see details below).
epi.by	A character vector of length 1 containing a nodal attribute for which subgroup epidemic prevalences should be calculated. This nodal attribute must be contained in the network model formation formula, otherwise it is ignored.
use.pids	If TRUE, use persistent ids for vertices; otherwise, numeric ids will be recycled in models with vital dynamics. For one-mode simulations, this will be a random hexidecimal value; for bipartite simulations, it will be based on pid.prefix.
pid.prefix	For bipartite network simulations with vital dynamics, a character vector of length 2 containing the prefixes, with the default of c("F", "M").
initialize.FUN	Module to initialize the model at time 1, with the default function of initialize.net.

`deaths.FUN` Module to simulate death or exit, with the default function of `deaths.net`.

`births.FUN` Module to simulate births or entries, with the default function of `births.net`.

`recovery.FUN` Module to simulate disease recovery, with the default function of `recovery.net`.

`edges_correct.FUN`
Module to adjust the edges coefficient in response to changes to the population size, with the default function of `edges_correct` that preserves mean degree.

`resim_nets.FUN` Module to resimulate the network at each time step, with the default function of `resim_nets`.

`infection.FUN` Module to simulate disease infection, with the default function of `infection.net`.

`get_prev.FUN` Module to calculate disease prevalence at each time step, with the default function of `get_prev.net`.

`verbose.FUN` Module to print simulation progress to screen, with the default function of `verbose.net`.

`module.order` A character vector of module names that lists modules the order in which they should be evaluated within each time step. If `NULL`, the modules will be evaluated as follows: first any new modules supplied through `...` in the order in which they are listed, then the built-in modules in their order of the function listing. The `initialize.FUN` will always be run first and the `verbose.FUN` always last.

`set.control.stergm`
Control arguments passed to simulate.stergm. See the help file for `netdx` for details and examples on specifying this parameter.

`save.nwstats` If `TRUE`, save network statistics in a data frame. The statistics to be saved are specified in the `nwstats.formula` argument.

`nwstats.formula`
A right-hand sided ERGM formula that includes network statistics of interest, with the default to the formation formula terms.

`delete.nodes` If `TRUE`, delete inactive nodes from the network after each time step, otherwise deactivate them but keep them in the network object. Deleting nodes increases computational efficiency in large network simulations.

`save.transmat` If `TRUE`, save a transmission matrix for each simulation. This object contains one row for each transmission event (see `discord_edgelist`).

`save.network` If `TRUE`, save a `networkDynamic` object containing full edge history for each simulation. If `delete.nodes` is set to `TRUE`, this will only contain a static network with the edge configuration at the final time step of each simulation.

`save.other` A vector of elements on the `dat` master data list to save out after each simulation. One example for built-in models is the attribute list, "attr", at the final time step.

`verbose` If `TRUE`, print model progress to the console.

`verbose.int` Time step interval for printing progress to console, where 0 prints completion status of entire simulation and positive integer `x` prints progress after each `x` time steps. The default is to print progress after each time step.

`skip.check` If `TRUE`, skips the error check for parameter values, initial conditions, and control settings before running the models. This is suggested only if encountering unnecessary errors when running new models.

`...` Additional control settings passed to model.

Details

`control.net` sets the required control settings for any network model solved with the `netsim` function. Controls are required for both built-in model types and when passing original process modules. For an overview of control settings for built-in network models, consult the Basic Network Models tutorial. For all built-in models, the `type` argument is a necessary parameter and it has no default.

The attr.rules Argument

The `attr.rules` parameter is used to specify the rules for how nodal attribute values for incoming nodes should be set. These rules are only necessary for models in which there are incoming nodes (i.e., births) and also there is a nodal attribute in the network model formation formula set in `netest`. There are three rules available for each attribute value:

- **"current":** new nodes will be assigned this attribute in proportion to the distribution of that attribute among existing nodes at that current time step.
- **"t1":** new nodes will be assigned this attribute in proportion to the distribution of that attribute among nodes at time 1 (that is, the proportions set in the original network for `netest`).
- **<Value>:** all new nodes will be assigned this specific value, with no variation.

For example, the rules list `attr.rules = list(race = "t1", sex = "current", status = "s")` specifies how the race, sex, and status attributes should be set for incoming nodes. By default, the rule is "current" for all attributes except status, in which case it is "s" (that is, all incoming nodes are susceptible).

New Modules

Built-in network models use a set of module functions that specify how the individual nodes in the network are subjected to infection, recovery, demographics, and other processes. Core modules are those listed in the `.FUN` arguments. For each module, there is a default function used in the simulation. The default infection module, for example, is contained in the `infection.net` function.

For original models, one may substitute replacement module functions for any the default functions. New modules may be added to the workflow at each time step by passing a module function via the `...` argument. Consult the New Network Models tutorial. One may remove existing modules, such as `births.FUN`, from the workflow by setting the parameter value for that argument to `NULL`.

See Also

Use `param.net` to specify model parameters and `init.net` to specify the initial conditions. Run the parameterized model with `netsim`.

dcm *Deterministic Compartmental Models*

Description

Solves deterministic compartmental epidemic models for infectious disease.

Usage

```
dcm(param, init, control)
```

Arguments

param	Model parameters, as an object of class param.dcm.
init	Initial conditions, as an object of class init.dcm.
control	Control settings, as an object of class control.dcm.

Details

The dcm function uses the ordinary differential equation solver in the deSolve package to model disease as a deterministic compartmental system. The parameterization for these models follows the standard approach in EpiModel, with epidemic parameters, initial conditions, and control settings. A description of solving DCMs with the dcm function may be found in the Basic DCMs tutorial.

The dcm function performs modeling of both built-in model types and original models with new structures. Built-in model types include one-group and two-group models with disease types for Susceptible-Infected (SI), Susceptible-Infected-Recovered (SIR), and Susceptible-Infected-Susceptible (SIS). New model types may be written and input into dcm following the steps outlined in the Solving New DCMs tutorial. Both built-in and original models require the param, init, and control inputs.

Value

A list of class dcm with the following elements:

- **param:** the epidemic parameters passed into the model through param, with additional parameters added as necessary.
- **control:** the control settings passed into the model through, control, with additional controls added as necessary.
- **epi:** a list of of data frames, one for each epidemiological output from the model. Outputs for built-in models always include the size of each compartment, as well as flows in, out, and between compartments.

References

Soetaert K, Petzoldt T, Setzer W. Solving Differential Equations in R: Package deSolve. Journal of Statistical Software. 2010; 33(9): 1-25. http://www.jstatsoft.org/v33/i09/.

See Also

Extract the model results with `as.data.frame.dcm`. Summarize the time-specific model results with `summary.dcm`. Plot the model results with `plot.dcm`. Plot a compartment flow diagram with `comp_plot`.

Examples

```
## Example 1: SI Model (One-Group)
# Set parameters
param <- param.dcm(inf.prob = 0.2, act.rate = 0.25)
init <- init.dcm(s.num = 500, i.num = 1)
control <- control.dcm(type = "SI", nsteps = 500)
mod1 <- dcm(param, init, control)
mod1
plot(mod1)

## Example 2: SIR Model with Vital Dynamics (One-Group)
param <- param.dcm(inf.prob = 0.2, act.rate = 5,
                   rec.rate = 1/3, b.rate = 1/90, ds.rate = 1/100,
                   di.rate = 1/35, dr.rate = 1/100)
init <- init.dcm(s.num = 500, i.num = 1, r.num = 0)
control <- control.dcm(type = "SIR", nsteps = 500)
mod2 <- dcm(param, init, control)
mod2
plot(mod2)

## Example 3: SIS Model with act.rate Sensitivity Parameter
param <- param.dcm(inf.prob = 0.2, act.rate = seq(0.1, 0.5, 0.1),
                   rec.rate = 1/50)
init <- init.dcm(s.num = 500, i.num = 1)
control <- control.dcm(type = "SIS", nsteps = 500)
mod3 <- dcm(param, init, control)
mod3
plot(mod3)

## Example 4: SI Model with Vital Dynamics (Two-Group)
param <- param.dcm(inf.prob = 0.4,  inf.prob.g2 = 0.1,
                   act.rate = 0.25, balance = "g1",
                   b.rate = 1/100, b.rate.g2 = NA,
                   ds.rate = 1/100, ds.rate.g2 = 1/100,
                   di.rate = 1/50, di.rate.g2 = 1/50)
init <- init.dcm(s.num = 500, i.num = 1,
                 s.num.g2 = 500, i.num.g2 = 0)
control <- control.dcm(type = "SI", nsteps = 500)
mod4 <- dcm(param, init, control)
mod4
plot(mod4)
```

`dissolution_coefs` *Dissolution Coefficients for Stochastic Network Models*

Description

Calculates dissolution coefficients, given a dissolution model and average edge duration, to pass as offsets to an ERGM/STERGM model fit in `netest`.

Usage

```
dissolution_coefs(dissolution, duration, d.rate = 0)
```

Arguments

`dissolution`	Right-hand sided STERGM dissolution formula (see `netest`). See below for list of supported dissolution models.
`duration`	A vector of mean edge durations in arbitrary time units.
`d.rate`	Death or exit rate from the population, as a single homogenous rate that applies to the entire population.

Details

This function performs two calculations for dissolution coefficients used in a network model estimated with `netest`:

1. **Transformation:** the mean duration of edges in a network are mathematically transformed to logit coefficients.
2. **Adjustment:** In a dynamic network simulation in an open population (in which there are deaths), it is further necessary to adjust these coefficients for dynamic simulations; this upward adjustment accounts for death as a competing risk to edge dissolution.

The current dissolution models supported by this function and in network model estimation in `netest` are as follows:

- `~offset(edges)`: a homogeneous dissolution model in which the edge duration is the same for all partnerships. This requires specifying one duration value.
- `~offset(edges) + offset(nodematch("<attr>"))`: a heterogeneous model in which the edge duration varies by whether the nodes in the dyad have similar values of a specified attribute. The duration vector should now contain two values: the first is the mean edge duration of non-matched dyads, and the second is the duration of the matched dyads.
- `~offset(edges) + offset(nodemix("<attr>"))`: a heterogenous model that extends the nodematch model to include non-binary attributes for homophily. The duration vector should first contain the base value, then the values for every other possible combination in the term.
- `~offset(edges) + offset(nodefactor("<attr>"))`: a heterogenous model in which the edge duration varies by a specified attribute. The duration vector should first contain the base value, then the values for every other value of that attribute in the term.

Value

A list of class `disscoef` with the following elements:

- **dissolution:** right-hand sided STERGM dissolution formula passed in the function call.
- **duration:** mean edge durations passed into the function.
- **coef.crude:** mean durations transformed into a logit coefficients.
- **coef.adj:** crude coefficients adjusted for the risk of death on edge persistence, if the `d.rate` argument is supplied.
- **d.rate:** the death rate.

See Also

The theory and details of this function are explained in detail in the EpiModel Network Utility Functions tutorial.

Examples

```
# Homogeneous dissolution model with no deaths
dissolution_coefs(dissolution = ~offset(edges), duration = 25)

# Homogeneous dissolution model with deaths
dissolution_coefs(dissolution = ~offset(edges), duration = 25,
                  d.rate = 0.001)

# Heterogeneous dissolution model in which same-race edges have
# shorter duration compared to mixed-race edges, with no deaths
dissolution_coefs(dissolution = ~offset(edges) + offset(nodematch("race")),
                  duration = c(20, 10))

# Heterogeneous dissolution model in which same-race edges have
# shorter duration compared to mixed-race edges, with deaths
dissolution_coefs(dissolution = ~offset(edges) + offset(nodematch("race")),
                  duration = c(20, 10), d.rate = 0.001)
```

`edgelist_censor` *Table of Edge Censoring*

Description

Outputs a table of the number and percent of edges that are left-censored, right-censored, both-censored, or uncensored for a `networkDynamic` object.

Usage

```
edgelist_censor(el)
```

Arguments

el	Timed edgelist with start and end times extracted from a networkDynamic object using the as.data.frame.networkDynamic function.

Details

Given a STERGM simulation over a specified number of time steps, the edges within that simulation may be left-censored (started before the first step), right-censored (continued after the last step), right and left-censored, or uncensored. The amount of censoring will increase when the average edge duration approaches the length of the simulation.

Examples

```
# Initialize and parameterize network model
nw <- network.initialize(n = 100, directed = FALSE)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)

# Model estimation
est <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

# Simulate the network and extract a timed edgelist
sim <- netdx(est, nsims = 1, nsteps = 100, verbose = FALSE)
el <- sim$edgelist[[1]]

# Calculate censoring
edgelist_censor(el)
```

---

epiweb *EpiModel Web*

Description

Runs a web browser-based GUI of deterministic compartmental models, stochastic individual contact models and basic network models.

Usage

```
epiweb(class, ...)
```

Arguments

class	Model class, with options of "dcm", "icm" and "net".
...	Additional arguments passed to shiny::runApp.

Details

epiweb runs a web-based GUI of one-group deterministic compartmental models, stochastic individual contact models, and stochastic network models with user input on model type, state sizes, and parameters. Model output may be plotted, summarized, and saved as raw data using the core EpiModel functionality for these model classes. These applications are built using the shiny package framework.

These apps are also hosted online at Rstudio's shinyapps site here:

- DCM App: 'https://statnet.shinyapps.io/epidcm/'
- ICM App: 'https://statnet.shinyapps.io/epiicm/'
- NET App: 'https://statnet.shinyapps.io/epinet/'

References

RStudio. shiny: Web Application Framework for R. R package version 0.12.2. 2015. http://www.rstudio.com/shiny/

See Also

dcm, icm, netsim

Examples

```
## Not run:
## Deterministic compartmental models
epiweb(class = "dcm")

## Stochastic individual contact models
epiweb(class = "icm")

## Stochastic network models
epiweb(class = "net")

## End(Not run)
```

---

get_network *Extract networkDynamic Object from Network Epidemic Model*

Description

Extracts the networkDynamic object from a network epidemic model simulated with netsim, with the option to collapse the extracted network at a specific time step.

Usage

```
get_network(x, sim = 1, network = 1, collapse = FALSE, at)
```

Arguments

x	An EpiModel object of class netsim.
sim	Simulation number of extracted network.
network	Network number, for simulations with multiple networks representing the population.
collapse	If TRUE, collapse the networkDynamic object to a static network object at a specified time step.
at	If collapse is used, the time step at which the extracted network should be collapsed.

Examples

```
## Not run:
## Simulate SI epidemic on bipartite Bernoulli random graph
nw <- network.initialize(n = 100, bipartite = 50, directed = FALSE)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)
est <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)
param <- param.net(inf.prob = 0.3, inf.prob.m2 = 0.15)
init <- init.net(i.num = 10, i.num.m2 = 10)
control <- control.net(type = "SI", nsteps = 10, nsims = 3, verbose = FALSE)
mod <- netsim(est, param, init, control)

## Extract the network from simulation 2
get_network(mod, sim = 2)

## Extract and collapse the network from simulation 1
get_network(mod, collapse = TRUE, at = 5)

## End(Not run)
```

---

get_nwstats *Extract Network Statistics from Network Epidemic Model*

Description

Extracts a data frame of network statistics from a network epidemic model.

Usage

```
get_nwstats(x, sim, network = 1)
```

Arguments

x	An EpiModel object of class netsim.
sim	A vector of simulation numbers of extracted network.
network	Network number, for simulations with multiple networks representing the population.

Examples

```
## Simulate SI epidemic on bipartite Bernoulli random graph
nw <- network.initialize(n = 100, bipartite = 50, directed = FALSE)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)
est <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)
param <- param.net(inf.prob = 0.3, inf.prob.m2 = 0.15)
init <- init.net(i.num = 10, i.num.m2 = 10)
control <- control.net(type = "SI", nsteps = 10, nsims = 3,
                       nwstats.formula = ~edges + meandeg + degree(0:5),
                       verbose = FALSE)
mod <- netsim(est, param, init, control)

## Extract the network statistics from simulation 2
get_nwstats(mod)
get_nwstats(mod, sim = c(1,3))
```

---

get_sims *Extract Network Simulations*

Description

Subsets the entire netsim object to a subset of simulations, essentially functioning like a reverse of merge.

Usage

```
get_sims(x, sims, var = "i.num")
```

Arguments

x	An object of class netsim.
sims	A vector of simulation numbers to retain in the output object, or "mean" which selects the one simulation with value of the variable specified in var closest to the mean of var across all simulations.
var	Variable to use when sims = "mean" for selecting the average simulation from the set.

icm *Stochastic Individual Contact Models*

Description

Simulates stochastic individual contact epidemic models for infectious disease.

Usage

```
icm(param, init, control)
```

Arguments

`param`	Model parameters, as an object of class `param.icm`.
`init`	Initial conditions, as an object of class `init.icm`.
`control`	Control settings, as an object of class `control.icm`.

Details

Individual contact models are intended to be the stochastic microsimulation analogs to deterministic compartmental models. ICMs simulate disease spread on individual agents in discrete time as a function of processes with stochastic variation. The stochasticity is inherent in all transition processes: infection, recovery, and demographics. A detailed description of these models may be found in the Basic ICMs tutorial.

The `icm` function performs modeling of both the built-in model types and original models. Built-in model types include one-group and two-group models with disease types for Susceptible-Infected (SI), Susceptible-Infected-Recovered (SIR), and Susceptible-Infected-Susceptible (SIS). Original models may be built by writing new process modules that either take the place of existing modules (for example, disease recovery), or supplement the set of existing processes with a new one contained in an original module.

Value

A list of class `icm` with the following elements:

- **param:** the epidemic parameters passed into the model through `param`, with additional parameters added as necessary.
- **control:** the control settings passed into the model through, `control`, with additional controls added as necessary.
- **epi:** a list of of data frames, one for each epidemiological output from the model. Outputs for built-in models always include the size of each compartment, as well as flows in, out, and between compartments.

See Also

Extract the model results with `as.data.frame.icm`. Summarize the time-specific model results with `summary.icm`. Plot the model results with `plot.icm`. Plot a compartment flow diagram with `comp_plot`.

Examples

```
## Not run:
## Example 1: SI Model
param <- param.icm(inf.prob = 0.2, act.rate = 0.25)
init <- init.icm(s.num = 500, i.num = 1)
control <- control.icm(type = "SI", nsteps = 500, nsims = 10)
mod1 <- icm(param, init, control)
mod1
plot(mod1)

## Example 2: SIR Model
param <- param.icm(inf.prob = 0.2, act.rate = 0.25, rec.rate = 1/50)
init <- init.icm(s.num = 500, i.num = 1, r.num = 0)
control <- control.icm(type = "SIR", nsteps = 500, nsims = 10)
mod2 <- icm(param, init, control)
mod2
plot(mod2)

## Example 3: SIS Model
param <- param.icm(inf.prob = 0.2, act.rate = 0.25, rec.rate = 1/50)
init <- init.icm(s.num = 500, i.num = 1)
control <- control.icm(type = "SIS", nsteps = 500, nsims = 10)
mod3 <- icm(param, init, control)
mod3
plot(mod3)

## Example 4: SI Model with Vital Dynamics (Two-Group)
param <- param.icm(inf.prob = 0.4,  inf.prob.g2 = 0.1,
                   act.rate = 0.25, balance = "g1",
                   b.rate = 1/100, b.rate.g2 = NA,
                   ds.rate = 1/100, ds.rate.g2 = 1/100,
                   di.rate = 1/50, di.rate.g2 = 1/50)
init <- init.icm(s.num = 500, i.num = 1,
                 s.num.g2 = 500, i.num.g2 = 0)
control <- control.icm(type = "SI", nsteps = 500, nsims = 10)
mod4 <- icm(param, init, control)
mod4
plot(mod4)

## End(Not run)
```

`init.dcm` *Initial Conditions for Deterministic Compartmental Models*

Description

Sets the initial conditions for deterministic compartmental models simulated with `dcm`.

Usage

```
init.dcm(s.num, i.num, r.num, s.num.g2, i.num.g2, r.num.g2, ...)
```

Arguments

s.num	Number of initial susceptible. For two-group models, this is the number of initial group 1 susceptible.
i.num	Number of initial infected. For two-group models, this is the number of initial group 1 infected.
r.num	Number of initial recovered. For two-group models, this is the number of initial group 1 recovered. This parameter is only used for the SIR model type.
s.num.g2	Number of initial susceptible in group 2. This parameter is only used for two-group models.
i.num.g2	Number of initial infected in group 2. This parameter is only used for two-group models.
r.num.g2	Number of initial recovered in group 2. This parameter is only used for two-group SIR models.
...	Additional initial conditions passed to model.

Details

The initial conditions for a model solved with dcm should be input into the init.dcm function. This function handles initial conditions for both built-in model types and original models. For an overview of initial conditions for built-in DCM class models, consult the Basic DCMs tutorial.

Original models may use the parameter names listed as arguments here, a new set of names, or a combination of both. With new models, initial conditions must be input in the same order that the solved derivatives from the model are output. More details on this requirement are outlined in the Solving New DCMs tutorial.

See Also

Use param.dcm to specify model parameters and control.dcm to specify the control settings. Run the parameterized model with dcm.

---

init.icm *Initial Conditions for Stochastic Individual Contact Models*

Description

Sets the initial conditions for stochastic individual contact models simulated with icm.

Usage

```
init.icm(s.num, i.num, r.num, s.num.g2, i.num.g2, r.num.g2,
  status.rand = FALSE, ...)
```

Arguments

s.num	Number of initial susceptible. For two-group models, this is the number of initial group 1 susceptible.
i.num	Number of initial infected. For two-group models, this is the number of initial group 1 infected.
r.num	Number of initial recovered. For two-group models, this is the number of initial group 1 recovered. This parameter is only used for the SIR model type.
s.num.g2	Number of initial susceptible in group 2. This parameter is only used for two-group models.
i.num.g2	Number of initial infected in group 2. This parameter is only used for two-group models.
r.num.g2	Number of initial recovered in group 2. This parameter is only used for two-group SIR models.
status.rand	If TRUE, sets infection based on random binomial draws from the distribution implied by the number susceptible, infected, and recovered in each group.
...	Additional initial conditions passed to model.

Details

The initial conditions for a model solved with icm should be input into the init.icm function. This function handles initial conditions for both built-in models and original models using new modules. For an overview of initial conditions for built-in ICM class models, consult the Basic ICMs tutorial.

See Also

Use param.icm to specify model parameters and control.icm to specify the control settings. Run the parameterized model with icm.

---

init.net *Initial Conditions for Stochastic Network Models*

Description

Sets the initial conditions for stochastic network models simulated with netsim.

Usage

```
init.net(i.num, r.num, i.num.m2, r.num.m2, status.vector, status.rand = FALSE,
  ...)
```

Arguments

i.num	Number of initial infected. For bipartite models, this is the number of initial mode 1 infected.
r.num	Number of initial recovered. For bipartite models, this is the number of initial mode 1 recovered. This parameter is only used for the SIR model type.
i.num.m2	Number of initial infected in mode 2. This parameter is only used for bipartite models.
r.num.m2	Number of initial recovered in mode 2. This parameter is only used for bipartite SIR models.
status.vector	A vector of length equal to the size of the input network, containing the status of each node. Setting status here overrides any inputs passed in the .num arguments and also overrides status.rand=TRUE.
status.rand	If TRUE and not using status.vector, sets infection based on random binomial draws from the distribution implied by the number infected and recovered in each mode.
...	Additional initial conditions passed to model.

Details

The initial conditions for a model solved with netsim should be input into the init.net function. This function handles initial conditions for both built-in models and new modules. For an overview of specifying initial conditions across a variety of built-in network models, consult the Basic Network Models tutorial.

See Also

Use param.net to specify model parameters and control.net to specify the control settings. Run the parameterized model with netsim.

---

is.transmat *Extract Transmissions Matrix from Network Epidemic Model*

Description

Extracts the matrix of transmission data for each transmission event that occured within a network epidemic model.

Usage

```
is.transmat(x)

get_transmat(x, sim = 1)
```

Arguments

x	An EpiModel object of class netsim.
sim	Simulation number of extracted network.

Value

A data frame with the following collumns

- **at:** the time step at which the transmission occurred.
- **sus:** the ID number of the susceptible (newly infected) node.
- **inf:** the ID number of the infecting node.
- **infDur:** the duration of the infecting node's disease at the time of the transmission.
- **transProb:** the probability of transmission per act.
- **actRate:** the rate of acts per unit time.
- **finalProb:** the final transmission probability for the transmission event.

Examples

```
## Simulate SI epidemic on bipartite Bernoulli random graph
nw <- network.initialize(n = 100, bipartite = 50, directed = FALSE)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)
est <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)
param <- param.net(inf.prob = 0.3, inf.prob.m2 = 0.15)
init <- init.net(i.num = 10, i.num.m2 = 10)
control <- control.net(type = "SI", nsteps = 10, nsims = 3, verbose = FALSE)
mod <- netsim(est, param, init, control)

## Extract the transmission matrix from simulation 2
get_transmat(mod, sim = 2)
```

---

merge.icm *Merge Data across Stochastic Individual Contact Model Simulations*

Description

Merges epidemiological data from two independent simulations of stochastic individual contact models from icm.

Usage

```
## S3 method for class 'icm'
merge(x, y, ...)
```

Arguments

x	An EpiModel object of class icm.
y	Another EpiModel object of class icm, with the identical model parameterization as x.
...	Additional merge arguments (not used).

Details

This merge function combines the results of two independent simulations of icm class models, simulated under separate function calls. The model parameterization between the two calls must be exactly the same, except for the number of simulations in each call. This allows for manual parallelization of model simulations.

This merge function does not work the same as the default merge in allowing for a combined object where the structure differs between the input elements. Instead, the function checks that objects are identical in model parameterization in every respect (except number of simulations) and binds the results.

Examples

```
param <- param.icm(inf.prob = 0.2, act.rate = 0.8)
init <- init.icm(s.num = 1000, i.num = 100)
control <- control.icm(type = "SI", nsteps = 10,
                       nsims = 3, verbose = FALSE)
x <- icm(param, init, control)

control <- control.icm(type = "SI", nsteps = 10,
                       nsims = 1, verbose = FALSE)
y <- icm(param, init, control)

z <- merge(x, y)
x$epi
y$epi
z$epi
```

---

merge.netsim *Merge Model Simulations Across netsim Objects*

Description

Merges epidemiological data from two independent simulations of stochastic network models from netsim.

Usage

```
## S3 method for class 'netsim'
merge(x, y, keep.transmat = TRUE, keep.network = TRUE,
  keep.nwstats = TRUE, keep.other = TRUE, param.error = TRUE, ...)
```

Arguments

x	An EpiModel object of class netsim.
y	Another EpiModel object of class netsim, with the identical model parameterization as x.
keep.transmat	If TRUE, keep the transmission matrices from the original x and y elements.
keep.network	If TRUE, keep the networkDynamic objects from the original x and y elements.
keep.nwstats	If TRUE, keep the network statistics (as set by the nwstats.formula parameter in control.netsim) from the original x and y elements.
keep.other	If TRUE, keep the other simulation elements (as set by the save.other parameter in control.netsim) from the original x and y elements.
param.error	If TRUE, if x and y have different params (in param.net) or controls (passed in control.net) an error will prevent the merge. Use FALSE to override that check.
...	Additional merge arguments (not currently used).

Details

This merge function combines the results of two independent simulations of netsim class models, simulated under separate function calls. The model parameterization between the two calls must be exactly the same, except for the number of simulations in each call. This allows for manual parallelization of model simulations.

This merge function does not work the same as the default merge in allowing for a combined object where the structure differs between the input elements. Instead, the function checks that objects are identical in model parameterization in every respect (except number of simulations) and binds the results.

Examples

```
# Network model
nw <- network.initialize(n = 100, directed = FALSE)
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 10)
est <- netest(nw, formation = ~edges, target.stats = 25,
              coef.diss = coef.diss, verbose = FALSE)

# Epidemic models
param <- param.net(inf.prob = 1)
init <- init.net(i.num = 1)
control <- control.net(type = "SI", nsteps = 20, nsims = 2,
                       save.nwstats = TRUE,
                       nwstats.formula = ~edges + degree(0),
                       verbose = FALSE)
x <- netsim(est, param, init, control)
y <- netsim(est, param, init, control)

# Merging
z <- merge(x, y)
x$epi
y$epi
```

```
z$epi
```

`modules.icm` *Modules for Stochastic Individual Contact Models*

Description

Stochastic individual contact models of infectious disease simulate epidemics in which contacts between individuals are instantaneous events in discrete time. They are intended to be the stochastic microsimulation analogs to deterministic compartmental models.

The `icm` function handles both the simulation tasks. Within this function are a series of modules that initialize the simulation, and then simulate new infections, recoveries, and vital dynamics at each time step. A module also handles the basic bookkeeping calculations for disease prevalence.

Writing original ICMs will require modifying the existing modules or adding new modules to the workflow in `icm`. The existing modules may be used as a template for replacement or new modules.

This help page presents a brief overview of the module functions in the order in which they are used within `icm`, in order to help guide users in writing their own module functions. These module functions are not shown on the help index since they are not called directly by the end-user. To understand these functions in more detail, review the separate help pages listed below.

Initialization Module

This function sets up the agent attributes like disease status on the network at the starting time step of disease simulation, t_1. For multiple-simulation function calls, these are reset at the beginning of each simulation.

- `initialize.icm`: sets which agents are initially infected, either through the initial conditions passed in `init.icm`.

Disease Status Modification Modules

The main disease simulation occurs at each time step given the current state of the population at that step. Infection of agents is simulated as a function of disease parameters and population composition. Recovery of agents is likewise simulated with respect to infected nodes. These functions also analyze the flows for summary measures such as disease incidence.

- `infection.icm`: randomly draws an edgelist given the parameters, subsets the list for discordant pairs, and simulates transmission on those discordant pairs through a series of draws from a binomial distribution.
- `recovery.icm`: simulates recovery from infection either to a lifelong immune state (for SIR models) or back to the susceptible state (for SIS models), as a function of the recovery rate specified in the `rec.rate` parameter. The recovery rate may vary for two-group models.

Demographic Modules

Vital dynamics such as birth and death processes are simulated at each time step to update entries into and exits from the population. These are used in open-population ICMs.

- deaths.icm: randomly simulates death or exits for agents given the death rate specified in the disease-state and group-specific death parameters in param.icm. This involves deactivating agents from the population, but their historical data is preserved in the simulation.
- births.icm: randomly simulates new births into the population given the current population size and the birth rate parameters. This involves adding new agents into the population.

Bookkeeping Module

Simulations require bookkeeping at each time step to calculate the summary epidemiological statistics used in the model output analysis.

- get_prev.icm: calculates the number in each disease state (susceptible, infected, recovered) at each time step for those active agents in the population.

---

modules.net *Modules for Stochastic Network Models*

Description

Stochastic network models of infectious disease in EpiModel require statistical modeling of networks, simulation of those networks forward through time, and simulation of epidemic dynamics on top of those evolving networks. The netsim function handles both the network and epidemic simulation tasks. Within this function are a series of modules that initialize the simulation, and then simulate new infections, recoveries, and demographics on the network. Modules also handle the resimulation of the network and some bookkeeping calculations for disease prevalence.

Writing original network models that expand upon our built-in model set will require modifying the existing modules or adding new modules to the workflow in netsim. The existing modules may be used as a template for replacement or new modules.

This help page provides an orientation to these module functions, in the order in which they are used within netsim, to help guide users in writing their own functions. These module functions are not shown on the help index since they are not called directly by the end-user. To understand these functions in more detail, review the separate help pages listed below.

Initialization Module

This function sets up the nodal attributes like disease status on the network at the starting time step of disease simulation, t_1. For multiple-simulation function calls, these are reset at the beginning of each individual simulation.

- initialize.net: sets up the master data structure used in the simulation, initializes which nodes are infected (via the initial conditions passed in init.net), and simulates a first time step of the networks given the network model fit from netest.

Disease Status Modification Modules

The main disease simulation occurs at each time step given the current state of the network at that step. Infection of nodes is simulated as a function of attributes of the nodes and the edges. Recovery of nodes is likewise simulated as a function of nodal attributes of those infected nodes. These functions also calculate summary flow measures such as disease incidence.

- `infection.net`: simulates disease transmission given an edgelist of discordant partnerships by calculating the relevant transmission and act rates for each edge, and then updating the nodal attributes and summary statistics.
- `recovery.net`: simulates recovery from infection either to a lifelong immune state (for SIR models) or back to the susceptible state (for SIS models), as a function of the recovery rate parameters specified `param.net`.

Demographic Modules

Demographics such as birth and death processes are simulated at each time step to update entries into and exits from the network. These are used in dependent network models, in which the network is resimulated at each time step to account for the nodal changes affecting the edges.

- `deaths.net`: randomly simulates death for nodes given their disease status (susceptible, infected, recovered), and their mode-specific death rates specified in `param.net`. Deaths involve deactivating nodes, which are then deleted from the network if `delete.nodes=TRUE` is set in `control.net`.
- `births.net`: randomly simulates new births into the network given the current population size and the birth rate specified in the `b.rate` parameters. This involves adding new nodes into the network.

Network Resimulation Module

In dependent network models, the network object is resimulated at each time step to account for changes in the size of the network (changed through entries and exits), and the disease status of the nodes.

- `edges_correct`: adjusts the edges coefficient of a network model to account for changes in the population size due to entries and exits. The default behavior is to preserve the mean degree (average number of edges per person) in response to change population sizes.
- `resim_nets`: resimulates the network object one time step forward given the set of formation and dissolution coefficients estimated in `netest`. This function also deletes the inactive nodes if the `delete.nodes` control is set to `TRUE`.

Bookkeeping Module

Network simulations require bookkeeping at each time step to calculate the summary epidemiological statistics used in the model output analysis.

- `get_prev.net`: calculates the number in each disease state (susceptible, infected, recovered) at each time step for those active nodes in the network. If the `epi.by` control is used, it calculates these statistics by a set of nodal attributes.
- `verbose.net`: summarizes the current state of the simulation and prints this to the console.

netdx *Dynamic Network Model Diagnostics*

Description

Runs dynamic diagnostics on an ERGM/STERGM estimated through netest

Usage

```
netdx(x, nsims = 1, dynamic = TRUE, nsteps, nwstats.formula = "formation",
  set.control.ergm, set.control.stergm, keep.tedgelist = FALSE,
  verbose = TRUE, ncores = 1)
```

Arguments

x	An EpiModel object of class netest.
nsims	Number of simulations to run.
dynamic	If TRUE, runs dynamic diagnostics. If FALSE and the netest object was fit with the Edges Dissolution approximation method, simulates from the static ERGM fit.
nsteps	Number of time steps per simulation (dynamic simulations only).
nwstats.formula	A right-hand sided ERGM formula with the network statistics of interest. The default is the formation formula of the network model contained in x.
set.control.ergm	Control arguments passed to simulate.ergm (see details).
set.control.stergm	Control arguments passed to simulate.stergm (see details).
keep.tedgelist	If TRUE, keep the timed edgelist generated from the dynamic simulations, for further analysis on edge durations.
verbose	Print progress to the console.
ncores	Number of processor cores to run multiple simulations on, using the foreach and doParallel implementations.

Details

The netdx function handles dynamic network diagnostics for network models fit with the netest function. Given the fitted model, netdx simulates a specified number of dynamic networks for a specified number of time steps per simulation. The network statistics in nwstats.formula are saved for each time step. Summary statistics for the formation model terms, as well as dissolution model and relational duration statistics, are then calculated for access when printing or plotting the netdx object.

Control Arguments

Models fit with the full STERGM method in `netest` (setting `edapprox` argument to `FALSE`) require only a call to `simulate.stergm`. Control parameters for those simulations may be set using `set.control.stergm` in `netdx`. The parameters should be input through the `control.simulate.stergm()` function, with the available parameters listed in the `control.simulate.stergm` help page in the `tergm` package.

Models fit with the ERGM method with the edges dissolution approximation (setting `edapprox` to `TRUE`) require a call first to `simulate.ergm` for simulating an initial network and second to `simulate.network` for simulating that static network forward through time. Control parameters may be set for both processes in `netdx`. For the first, the parameters should be input through the `control.simulate.ergm()` function, with the available parameters listed in the `control.simulate.ergm` help page in the `ergm` package. For the second, parameters should be input through the `control.simulate.network()` function, with the available parameters listed in the `control.simulate.network` help page in the `tergm` package. An example is shown below.

See Also

Plot these model diagnostics with `plot.netdx`.

Examples

```
## Not run:
# Network initialization and model parameterization
nw <- network.initialize(100, directed = FALSE)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 25)

# Estimate the model
est <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

# Static diagnostics on the ERGM fit
dx1 <- netdx(est, nsims = 1e4, dynamic = FALSE,
             nwstats.formula = ~edges + meandeg + concurrent)
dx1
plot(dx1, method = "b", stats = c("edges", "concurrent"))

# Dynamic diagnostics on the STERGM approximation
dx2 <- netdx(est, nsims = 5, nsteps = 500,
             nwstats.formula = ~edges + meandeg + concurrent,
             set.control.ergm = control.simulate.ergm(MCMC.burnin = 1e6))
dx2
plot(dx2, stats = c("edges", "meandeg"), plots.joined = FALSE)
plot(dx2, type = "duration")
plot(dx2, type = "dissolution", method = "b", col = "bisque")

## End(Not run)
```

netest *Dynamic Network Model Estimation*

Description

Estimates statistical network models using the exponential random graph modeling (ERGM) framework with extensions for dynamic/temporal models (STERGM).

Usage

```
netest(nw, formation, target.stats, coef.diss, constraints, coef.form = NULL,
  edapprox = TRUE, output = "fit", set.control.ergm, set.control.stergm,
  nonconv.error = FALSE, verbose = FALSE)
```

Arguments

nw	An object of class network.
formation	Right-hand sided STERGM formation formula in the form ~edges + ..., where ... are additional network statistics.
target.stats	Vector of target statistics for the formation model, with one number for each network statistic in the model.
coef.diss	An object of class disscoef output from the dissolution_coefs function.
constraints	Right-hand sided formula specifying constraints for the modeled network, in the form ~..., where ... are constraint terms. By default, no constraints are set.
coef.form	Vector of coefficients for the offset terms in the formation formula.
edapprox	If TRUE, use the indirect edges dissolution approximation method for the dynamic model fit, otherwise use the more time-intensive full STERGM estimation (see details).
output	If using the edges dissolution approximation method, "sim" simulates a static network from the fitted network model, for storage efficiency purposes.
set.control.ergm	Control arguments passed to simulate.ergm (see details).
set.control.stergm	Control arguments passed to simulate.stergm (see details).
nonconv.error	If TRUE, function will error if model did not converge after the specified number of iterations. This may be useful in batch mode while fitting many models and there is no desire to return to the nonconverged model object.
verbose	Print model fitting progress to console.

Details

netest is a wrapper function for the ergm and stergm functions that estimate static and dynamic network models, respectively. Network model estimation is the first step in simulating a stochastic network epidemic model in EpiModel. The output from netest is a necessary input for running the epidemic simulations in netsim. With a fitted network model, one should always first proceed to model diagnostics, available through the netdx function, to check model fit. A detailed description of fitting these models, along with examples, may be found in the Basic Network Models tutorial.

Edges Dissolution Approximation

The edges dissolution approximation method is described in Carnegie et al. This approximation requires that the dissolution coefficients are known, that the formation model is being fit to cross-sectional data conditional on those dissolution coefficients, and that the terms in the dissolution model are a subset of those in the formation model. Under certain additional conditions, the formation coefficients of a STERGM model are approximately equal to the coefficients of that same model fit to the observed cross-sectional data as an ERGM, minus the corresponding coefficients in the dissolution model. The approximation thus estimates this ERGM (which is typically much faster than estimating a STERGM) and subtracts the dissolution coefficients.

The conditions under which this approximation best hold are when there are few relational changes from one time step to another; i.e. when either average relational durations are long, or density is low, or both. Conveniently, these are the same conditions under which STERGM estimation is slowest. Note that the same approximation is also used to obtain starting values for the STERGM estimate when the latter is being conducted. The estimation does not allow for calculation of standard errors, p-values, or likelihood for the formation model; thus, this approach is of most use when the main goal of estimation is to drive dynamic network simulations rather than to conduct inference on the formation model. The user is strongly encouraged to examine the behavior of the resulting simulations to confirm that the approximation is adequate for their purposes. For an example, see the vignette for the package tergm.

Control Arguments

The ergm and stergm functions allow control settings for the model fitting process. When fitting a STERGM directly (setting edapprox to FALSE) control parameters may be passed to the stergm function with the set.control.stergm argument in netest. The controls should be input through the control.stergm() function, with the available parameters listed in the control.stergm help page in the tergm package.

When fitting a STERGM indirectly (setting edapprox to TRUE) control settings may be passed to the ergm function using set.control.ergm in netest. The controls should be input through the control.ergm() function, with the available parameters listed in the control.ergm help page in the ergm package. An example is below.

References

Krivitsky PN, Handcock MS. "A separable model for dynamic networks." JRSS(B). 2014; 76.1:29-46.

Carnegie NB, Krivitsky PN, Hunter DR, Goodreau SM. An approximation method for improving dynamic network model fitting. Journal of Computational and Graphical Statistics. 2014; In press.

See Also

Use netdx to diagnose the fitted network model, and netsim to simulate epidemic spread over a simulated dynamic network consistent with the model fit.

Examples

```
# Initialize a network of 100 nodes
nw <- network.initialize(n = 100, directed = FALSE)

# Set formation formula
formation <- ~edges + concurrent

# Set target statistics for formation
target.stats <- c(50, 25)

# Obtain the offset coefficients
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 10)

# Estimate the STERGM using the edges dissolution approximation
est <- netest(nw, formation, target.stats, coef.diss,
              set.control.ergm = control.ergm(MCMC.burnin = 1e5,
                                              MCMC.interval = 1000))
est

# To estimate the STERGM directly, use edapprox = FALSE
# est2 <- netest(nw, formation, target.stats, coef.diss, edapprox = FALSE)
```

---

netsim *Stochastic Network Models*

Description

Simulates stochastic network epidemic models for infectious disease.

Usage

```
netsim(x, param, init, control)
```

Arguments

x	Fitted network model object, as an object of class netest. Alternatively, if restarting a previous simulation, may be an object of class netsim.
param	Model parameters, as an object of class param.net.
init	Initial conditions, as an object of class init.net.
control	Control settings, as an object of class control.net.

Details

Stochastic network models move beyond stochastic individual contact models by explicitly modeling phenomena within and across edges (pairs of nodes that remain connected) over time. This enables edges to have duration, allowing for repeated transmission-related acts within the same dyad, specification of edge formation and dissolution rates, control over the temporal sequencing of multiple edges, and specification of network-level features. A detailed description of these models, along with examples, is found in the the Basic Network Models tutorial.

The `netsim` function performs modeling of both the built-in model types and original models. Built-in model types include one-mode and bipartite models with disease types for Susceptible-Infected (SI), Susceptible-Infected-Recovered (SIR), and Susceptible-Infected-Susceptible (SIS).

Original models may be parameterized by writing new process modules that either take the place of existing modules (for example, disease recovery), or supplement the set of existing processes with a new one contained in an new module. This functionality is documented in the Solving New Network Models tutorial. The list of modules within `netsim` available for modification is listed in `modules.net`.

Value

A list of class `netsim` with the following elements:

- **param:** the epidemic parameters passed into the model through `param`, with additional parameters added as necessary.
- **control:** the control settings passed into the model through, `control`, with additional controls added as necessary.
- **epi:** a list of of data frames, one for each epidemiological output from the model. Outputs for built-in models always include the size of each compartment, as well as flows in, out, and between compartments.
- **stats:** a list containing two sublists, `nwstats` for any network statistics saved in the simulation, and `transmat` for the transmission matrix saved in the simulation. See `control.net` and the Tutorial for further details.
- **network:** a list of `networkDynamic` objects (or `network` objects if `delete.nodes` was set to `TRUE`), one for each model simulation.

References

Goodreau SM, Carnegie NB, et al. What drives the US and Peruvian HIV epidemics in men who have sex with men (MSM)? PloS One. 2012; 7(11): e50522.

See Also

Extract the model results with `as.data.frame.netsim`. Summarize the time-specific model results with `summary.netsim`. Plot the model results with `plot.netsim`.

Examples

```
## Not run:
## Example 1: Independent SI Model
```

```
# Network model estimation
nw <- network.initialize(n = 100, bipartite = 50, directed = FALSE)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)
est1 <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

# Epidemic model
param <- param.net(inf.prob = 0.3, inf.prob.m2 = 0.15)
init <- init.net(i.num = 10, i.num.m2 = 10)
control <- control.net(type = "SI", nsteps = 100, nsims = 5, verbose.int = 0)
mod1 <- netsim(est1, param, init, control)

# Print, plot, and summarize the results
mod1
plot(mod1)
summary(mod1, at = 50)

## Example 2: Dependent SIR Model
# Recalculate dissolution coefficient with death rate
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20,
                               d.rate = 0.0021)

# Reestimate the model with new coefficient
est2 <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

# Reset parameters to include demographic rates
param <- param.net(inf.prob = 0.3, inf.prob.m2 = 0.15,
                   rec.rate = 0.02, rec.rate.m2 = 0.02,
                   b.rate = 0.002, b.rate.m2 = NA,
                   ds.rate = 0.001, ds.rate.m2 = 0.001,
                   di.rate = 0.001, di.rate.m2 = 0.001,
                   dr.rate = 0.001, dr.rate.m2 = 0.001)
init <- init.net(i.num = 10, i.num.m2 = 10,
                 r.num = 0, r.num.m2 = 0)
control <- control.net(type = "SIR", nsteps = 100, nsims = 5)

# Simulate the model with new network fit
mod2 <- netsim(est2, param, init, control)

# Print, plot, and summarize the results
mod2
plot(mod2)
summary(mod2, at = 100)

## End(Not run)
```

---

param.dcm *Epidemic Parameters for Deterministic Compartmental Models*

Description

Sets the epidemic parameters for deterministic compartmental models simulated with dcm.

Usage

```
param.dcm(inf.prob, inter.eff, inter.start, act.rate, rec.rate, b.rate, ds.rate,
  di.rate, dr.rate, inf.prob.g2, act.rate.g2, rec.rate.g2, b.rate.g2,
  ds.rate.g2, di.rate.g2, dr.rate.g2, balance, ...)
```

Arguments

inf.prob	Probability of infection per transmissible act between a susceptible and an infected person. In two-group models, this is the probability of infection for the group 1 members.
inter.eff	Efficacy of an intervention which affects the per-act probability of infection. Efficacy is defined as 1 - the relative hazard of infection given exposure to the intervention, compared to no exposure.
inter.start	Time step at which the intervention starts, between 1 and the number of time steps specified in the model. This will default to 1 if the inter.eff is defined but this parameter is not.
act.rate	Average number of transmissible acts per person per unit time. For two-group models, this is the number of acts per group 1 persons per unit time; a balance between the acts in groups 1 and 2 is necessary, and set using the balance parameter (see details).
rec.rate	Average rate of recovery with immunity (in SIR models) or re-susceptibility (in SIS models). The recovery rate is the reciprocal of the disease duration. For two-group models, this is the recovery rate for group 1 persons only. This parameter is only used for SIR and SIS models.
b.rate	Birth or entry rate. For one-group models, the birth rate is the rate of new births per person per unit time. For two-group models, the birth rate may be parameterized as a rate per group 1 person time (with group 1 persons representing females), and with the b.rate.g2 rate set as described below.
ds.rate	Death or exit rate for susceptible. For two-group models, it is the rate for the group 1 susceptible only.
di.rate	Death or exit rate for infected. For two-group models, it is the rate for the group 1 infected only.
dr.rate	Death or exit rate for recovered. For two-group models, it is the rate for the group 1 recovered only. This parameter is only used for SIR models.
inf.prob.g2	Probability of infection per transmissible act between a susceptible group 2 person and an infected group 1 person. It is the probability of infection to group 2 members.
act.rate.g2	Average number of transmissible acts per group 2 person per unit time; a balance between the acts in groups 1 and 2 is necessary, and set using the balance parameter (see details).

`rec.rate.g2` Average rate of recovery with immunity (in `SIR` models) or re-susceptibility (in `SIS` models) for group 2 persons. This parameter is only used for two-group `SIR` and `SIS` models.

`b.rate.g2` Birth or entry rate for group 2. This may either be specified numerically as the rate of new births per group 2 persons per unit time, or as `NA` in which case the group 1 rate, `b.rate`, governs the group 2 rate. The latter is used when, for example, the first group is conceptualized as female, and the female population size determines the birth rate. Such births are evenly allocated between the two groups.

`ds.rate.g2` Death or exit rate for group 2 susceptible.

`di.rate.g2` Death or exit rate for group 2 infected.

`dr.rate.g2` Death or exit rate for group 2 recovered. This parameter is only used for `SIR` model types.

`balance` For two-group models, balance the `act.rate` to the rate set for group 1 (with `balance="g1"`) or group 2 (with `balance="g2"`). See details.

`...` Additional arguments passed to model.

Details

`param.dcm` sets the epidemic parameters for deterministic compartmental models solved with the `dcm` function. The models may use the built-in types, for which these parameters are used, or original model specifications for which these parameters may be used (but not necessarily). A detailed description of DCM parameterization for built-in models is found in the Basic DCMs tutorial.

For built-in models, the model specification will be selected as a function of the model parameters entered here and the control settings in `control.dcm`. One-group and two-group models are available, where the former assumes a homogenous mixing in the population and the latter assumes a purely heterogenous mixing between two distinct partitions in the population (e.g., men and women). Specifying any group two parameters (those with a `.g2`) implies the simulation of a two-group model. All the parameters for a desired model type must be specified, even if they are zero.

Act Balancing

In two-group models, a balance between the number of acts for group 1 members and those for group 2 members must be maintained. With purely heterogenous mixing, the product of one group size and act rate must equal the product of the other group size and act rate: $N_1\alpha_1 = N_2\alpha_2$, where N_i is the group size and α_i the group-specific act rates at time t. The `balance` parameter here specifies which group's act rate should control the others with respect to balancing. See the Basic DCMs tutorial for further details.

Sensitivity Analyses

`dcm` has been designed to easily run DCM sensitivity analyses, where a series of models varying one or more of the model parameters is run. This is possible by setting any parameter as a vector of length greater than one. See both the example below and the Basic DCMs tutorial.

New Model Types

To build original model specifications outside of the built-in models, start by consulting the Solving New DCMs with EpiModel tutorial. Briefly, an original model may use either the existing model parameters named here, an original set of parameters, or a combination of both. The `...` argument allows the user to pass an arbitrary set of new model parmeters into `param.dcm`. Whereas there are strict checks for built-in models that the model parameters are valid, parameter validity is the user's responsibility with these original models.

See Also

Use `init.dcm` to specify the initial conditions and `control.dcm` to specify the control settings. Run the parameterized model with `dcm`.

`param.icm` *Epidemic Parameters for Stochastic Individual Contact Models*

Description

Sets the epidemic parameters for stochastic individual contact models simulated with `icm`.

Usage

```
param.icm(inf.prob, inter.eff, inter.start, act.rate, rec.rate, b.rate, ds.rate,
  di.rate, dr.rate, inf.prob.g2, act.rate.g2, rec.rate.g2, b.rate.g2,
  ds.rate.g2, di.rate.g2, dr.rate.g2, balance, ...)
```

Arguments

`inf.prob`	Probability of infection per transmissible act between a susceptible and an infected person. In two-group models, this is the probability of infection for the group 1 members.
`inter.eff`	Efficacy of an intervention which affects the per-act probability of infection. Efficacy is defined as 1 - the relative hazard of infection given exposure to the intervention, compared to no exposure.
`inter.start`	Time step at which the intervention starts, between 1 and the number of time steps specified in the model. This will default to 1 if the inter.eff is defined but this parameter is not.
`act.rate`	Average number of transmissible acts per person per unit time. For two-group models, this is the number of acts per group 1 persons per unit time; a balance between the acts in groups 1 and 2 is necessary, and set using the `balance` parameter (see details).
`rec.rate`	Average rate of recovery with immunity (in `SIR` models) or re-susceptibility (in `SIS` models). The recovery rate is the reciprocal of the disease duration. For two-group models, this is the recovery rate for group 1 persons only. This parameter is only used for `SIR` and `SIS` models.

b.rate	Birth or entry rate. For one-group models, the birth rate is the rate of new births per person per unit time. For two-group models, the birth rate may be parameterized as a rate per group 1 person time (with group 1 persons representing females), and with the b.rate.g2 rate set as described below.
ds.rate	Death or exit rate for susceptible. For two-group models, it is the rate for the group 1 susceptible only.
di.rate	Death or exit rate for infected. For two-group models, it is the rate for the group 1 infected only.
dr.rate	Death or exit rate for recovered. For two-group models, it is the rate for the group 1 recovered only. This parameter is only used for SIR models.
inf.prob.g2	Probability of infection per transmissible act between a susceptible group 2 person and an infected group 1 person. It is the probability of infection to group 2 members.
act.rate.g2	Average number of transmissible acts per group 2 person per unit time; a balance between the acts in groups 1 and 2 is necessary, and set using the balance parameter (see details).
rec.rate.g2	Average rate of recovery with immunity (in SIR models) or re-susceptibility (in SIS models) for group 2 persons. This parameter is only used for two-group SIR and SIS models.
b.rate.g2	Birth or entry rate for group 2. This may either be specified numerically as the rate of new births per group 2 persons per unit time, or as NA in which case the group 1 rate, b.rate, governs the group 2 rate. The latter is used when, for example, the first group is conceptualized as female, and the female population size determines the birth rate. Such births are evenly allocated between the two groups.
ds.rate.g2	Death or exit rate for group 2 susceptible.
di.rate.g2	Death or exit rate for group 2 infected.
dr.rate.g2	Death or exit rate for group 2 recovered. This parameter is only used for SIR model types.
balance	For two-group models, balance the act.rate to the rate set for group 1 (with balance="g1") or group 2 (with balance="g2"). See details.
...	Additional arguments passed to model.

Details

param.icm sets the epidemic parameters for the stochastic individual contact models simulated with the icm function. Models may use the built-in types, for which these parameters are used, or new process modules which may use these parameters (but not necessarily). A detailed description of ICM parameterization for built-in models is found in the Basic ICMs tutorial.

For built-in models, the model specification will be chosen as a result of the model parameters entered here and the control settings in control.icm. One-group and two-group models are available, where the former assumes a homogenous mixing in the population whereas the latter assumes a purely heterogenous mixing between two distinct partitions in the population (e.g., men and women). Specifying any group two parameters (those with a .g2) implies the simulation of a two-group model. All the parameters for a desired model type must be specified, even if they are zero.

Act Balancing

In two-group models, a balance between the number of acts for group 1 members and those for group 2 members must be maintained. With purely heterogenous mixing, the product of one group size and act rate must equal the product of the other group size and act rate: $N_1\alpha_1 = N_2\alpha_2$, where N_i is the group size and α_i the group-specific act rates at time t. The `balance` parameter here specifies which group's act rate should control the others with respect to balancing. See the Basic DCMs tutorial.

New Modules

To build original models outside of the built-in models, new process modules may be constructed to replace the existing modules or to supplement the existing set. These are passed into the control settings in `control.icm`. New modules may use either the existing model parameters named here, an original set of parameters, or a combination of both. The `...` allows the user to pass an arbitrary set of original model parmeters into `param.icm`. Whereas there are strict checks with default modules for parameter validity, these checks are the user's responsibility with new modules.

See Also

Use `init.icm` to specify the initial conditions and `control.icm` to specify the control settings. Run the parameterized model with `icm`.

`param.net` *Epidemic Parameters for Stochastic Network Models*

Description

Sets the epidemic parameters for stochastic network models simulated with `netsim`.

Usage

```
param.net(inf.prob, inter.eff, inter.start, act.rate, rec.rate, b.rate, ds.rate,
  di.rate, dr.rate, inf.prob.m2, rec.rate.m2, b.rate.m2, ds.rate.m2, di.rate.m2,
  dr.rate.m2, ...)
```

Arguments

`inf.prob`	Probability of infection per transmissible act between a susceptible and an infected person. In bipartite models, this is the probability of infection to the mode 1 nodes. This may also be a vector of probabilities, with each element corresponding to the probability in that time step of infection (see Time-Varying Parameters below).
`inter.eff`	Efficacy of an intervention which affects the per-act probability of infection. Efficacy is defined as 1 - the relative hazard of infection given exposure to the intervention, compared to no exposure.

inter.start	Time step at which the intervention starts, between 1 and the number of time steps specified in the model. This will default to 1 if the inter.eff is defined but this parameter is not.
act.rate	Average number of transmissible acts *per partnership* per unit time (see act.rate Parameter below). This may also be a vector of rates, with each element corresponding to the rate in in that time step of infection (see Time-Varying Parameters below).
rec.rate	Average rate of recovery with immunity (in SIR models) or re-susceptibility (in SIS models). The recovery rate is the reciprocal of the disease duration. For bipartite models, this is the recovery rate for mode 1 persons only. This parameter is only used for SIR and SIS models. This may also be a vector of rates, with each element corresponding to the rate in that time step of infection (see Time-Varying Parameters below).
b.rate	Birth or entry rate. For one-mode models, the birth rate is the rate of new births per person per unit time. For bipartite models, the birth rate may be parameterized as a rate per mode 1 person time (with mode 1 persons representing females), and with the b.rate.g2 rate set as described below.
ds.rate	Death or exit rate for susceptible. For bipartite models, it is the rate for the mode 1 susceptible only.
di.rate	Death or exit rate for infected. For bipartite models, it is the rate for the mode 1 infected only.
dr.rate	Death or exit rate for recovered. For bipartite models, it is the rate for the mode 1 recovered only. This parameter is only used for SIR models.
inf.prob.m2	Probability of transmission given a transmissible act between a susceptible mode 2 person and an infected mode 1 person. It is the probability of transmission to mode 2 members.
rec.rate.m2	Average rate of recovery with immunity (in SIR models) or re-susceptibility (in SIS models) for mode 2 persons. This parameter is only used for bipartite SIR and SIS models.
b.rate.m2	Birth or entry rate for mode 2. This may either be specified numerically as the rate of new births per mode 2 persons per unit time, or as NA in which case the mode 1 rate, b.rate, governs the mode 2 rate. The latter is used when, for example, the first mode is conceptualized as female, and the female population size determines the birth rate. Such births are evenly allocated between the two modes.
ds.rate.m2	Death or exit rate for mode 2 susceptible.
di.rate.m2	Death or exit rate for mode 2 infected.
dr.rate.m2	Death or exit rate for mode 2 recovered. This parameter is only used for SIR model types.
...	Additional arguments passed to model.

Details

param.net sets the epidemic parameters for the stochastic network models simulated with the netsim function. Models may use the built-in types, for which these parameters are used, or new

process modules which may use these parameters (but not necessarily). A detailed description of network model parameterization for built-in models is found in the Basic Network Models tutorial.

For built-in models, the model specification will be chosen as a result of the model parameters entered here and the control settings in `control.net`. One-mode and two-mode models are available, where the the latter assumes a heterogenous mixing between two distinct partitions in the population (e.g., men and women). Specifying any bipartite parameters (those with a `.m2`) implies the simulation of a bipartite model. All the parameters for a desired model type must be specified, even if they are zero.

The act.rate Parameter

A key difference between these network models and DCM/ICM classes is the treatment of transmission events. With DCM and ICM, contacts or partnerships are mathematically instantaneous events: they have no duration in time, and thus no changes may occur within them over time. In contrast, network models allow for partnership durations defined by the dynamic network model, summarized in the model dissolution coefficients calculated in `dissolution_coefs`. Therefore, the `act.rate` parameter has a different interpretation here, where it is the number of transmissible acts *per partnership* per unit time.

Time-Varying Parameters

The `inf.prob`, `act.rate`, `rec.rate` arguments (and their `.m2` companions) may be specified as time-varying parameters by passing in a vector of probabilities or rates, respectively. The value in each position on the vector then corresponds to the probability or rate at that discrete time step for the infected partner. For example, an `inf.prob` of `c(0.5, 0.5, 0.1)` would simulate a 0.5 transmission probability for the first two time steps of a person's infection, followed by a 0.1 for the third time step. If the infected person has not recovered or exited the population by the fourth time step, the third element in the vector will carry forward until one of those occurs or the simulation ends. For further examples, see the NME tutorial, Additional Modeling Topics.

New Modules

To build original models outside of the built-in models, new process modules may be constructed to replace the existing modules or to supplement the existing set. These are passed into the control settings in `control.net`. New modules may use either the existing model parameters named here, an original set of parameters, or a combination of both. The `...` allows the user to pass an arbitrary set of original model parmeters into `param.net`. Whereas there are strict checks with default modules for parameter validity, these checks are the user's responsibility with new modules.

See Also

Use `init.net` to specify the initial conditions and `control.net` to specify the control settings. Run the parameterized model with `netsim`.

plot.dcm *Plot Data from a Deterministic Compartmental Epidemic Model*

Description

Plots epidemiological data from a deterministic compartment epidemic model solved with dcm.

Usage

```
## S3 method for class 'dcm'
plot(x, y, popfrac, run, col, lwd, lty, alpha = 0.9, leg,
  leg.name, leg.cex = 0.8, axs = "r", add = FALSE, ...)
```

Arguments

x	An EpiModel object of class dcm.
y	Output compartments or flows from dcm object to plot.
popfrac	If TRUE, plot prevalence of values rather than numbers (see details).
run	Run number to plot, for models with multiple runs (default is run 1).
col	Color for lines, either specified as a single color in a standard R color format, or alternatively as a color palette from RColorBrewer (see details).
lwd	Line width for output lines.
lty	Line type for output lines.
alpha	Transparency level for lines, where 0 = transparent and 1 = opaque (see transco).
leg	Type of legend to plot. Values are "n" for no legend, "full" for full legend, and "lim" for limited legend (see details).
leg.name	Character string to use for legend, with the default determined automatically based on the y input.
leg.cex	Legend scale size.
axs	Plot axis type (see par for details), with default of "r".
add	If TRUE, new plot window is not called and lines are added to existing plot window.
...	Additional arguments to pass to main plot window (see plot.default).

Details

This function plots epidemiological outcomes from a deterministic compartmental model solved with dcm. Depending on the number of model runs (sensitivity analyses) and number of groups, the default plot is the fractional proportion of each compartment in the model over time. The specific compartments or flows to plot may be set using the y parameter, and in multiple run models the specific run may also be specified.

The popfrac Argument

Compartment prevalences are the size of a compartment over some denominator. To plot the raw numbers from any compartment, use `popfrac=FALSE`; this is the default for any plots of flows. The `popfrac` parameter calculates and plots the denominators of all specified compartments using these rules: 1) for one-group models, the prevalence of any compartment is the compartment size divided by the total population size; 2) for two-group models, the prevalence of any compartment is the compartment size divided by the group size.

Color Palettes

Since `dcm` supports multiple run sensitivity models, plotting the results of such models uses a complex color scheme for distinguishing runs. This is accomplished using the `RColorBrewer` color palettes, in which includes a range of linked colors using named palettes. For `plot.dcm`, one may either specify a brewer color palette listed in `brewer.pal.info`, or alternatively a vector of standard R colors (named, hexidecimal, or positive integers; see `col2rgb`).

Plot Legends

There are three automatic legend types available, and the legend is added by default for plots. To turn off the legend, use `leg="n"`. To plot a legend with values for every line in a sensitivity analysis, use `leg="full"`. With models with many runs, this may be visually overwhelming. In those cases, use `leg="lim"` to plot a legend limited to the highest and lowest of the varying parameter in the model. In cases where the default legend names are not helpful, one may override those names with the `leg.name` argument.

See Also

`dcm`, `brewer.pal.info`

Examples

```
# Deterministic SIR model with varying act rate
param <- param.dcm(inf.prob = 0.2, act.rate = 1:10,
                   rec.rate = 1/3, b.rate = 0.011, ds.rate = 0.01,
                   di.rate = 0.03, dr.rate = 0.01)
init <- init.dcm(s.num = 1000, i.num = 1, r.num = 0)
control <- control.dcm(type = "SIR", nsteps = 100, dt = 0.25)
mod <- dcm(param, init, control)

# Plot disease prevalence by default
plot(mod)

# Plot prevalence of susceptibles
plot(mod, y = "s.num", col = "Greys")

# Plot number of susceptibles
plot(mod, y = "s.num", popfrac = FALSE, col = "Greys")

# Plot multiple runs of multiple compartments together
plot(mod, y = c("s.num", "i.num"),
```

```
     run = 5, xlim = c(0, 50))
plot(mod, y = c("s.num", "i.num"),
     run = 10, lty = 2, leg = "n", add = TRUE)
```

---

plot.icm *Plot Data from a Stochastic Individual Contact Epidemic Model*

Description

Plots epidemiological data from a stochastic individual contact model simulated with icm.

Usage

```
## S3 method for class 'icm'
plot(x, y, popfrac, sim.lines = FALSE, sims, sim.col, sim.lwd,
  sim.alpha, mean.line = TRUE, mean.smooth = TRUE, mean.col, mean.lwd = 2,
  mean.lty = 1, qnts = 0.5, qnts.col, qnts.alpha, qnts.smooth = TRUE, leg,
  leg.cex = 0.8, axs = "r", add = FALSE, ...)
```

Arguments

x	An EpiModel model object of class netsim.
y	Output compartments or flows from icm object to plot.
popfrac	If TRUE, plot prevalence of values rather than numbers (see details).
sim.lines	If TRUE, plot individual simulation lines. Default is to plot lines for one-group models but not for two-group models.
sims	If type="epi" or "formation", a vector of simulation numbers to plot. If type="network", a single simulation number for network plot, or else "min" to plot the simulation number with the lowest disease prevalence, "max" for the simulation with the highest disease prevalence, or "mean" for the simulation with the prevalance closest to the mean across simulations at the specified time step.
sim.col	Vector of any standard R color format for simulation lines.
sim.lwd	Line width for simulation lines.
sim.alpha	Transparency level for simulation lines, where 0 = transparent and 1 = opaque (see transco).
mean.line	If TRUE, plot mean of simulations across time.
mean.smooth	If TRUE, use a lowess smoother on the mean line.
mean.col	Vector of any standard R color format for mean lines.
mean.lwd	Line width for mean lines.
mean.lty	Line type for mean lines.
qnts	If numeric, plot polygon of simulation quantiles based on the range implied by the argument (see details). If FALSE, suppress polygon from plot.

qnts.col	Vector of any standard R color format for polygons.
qnts.alpha	Transparency level for quantile polygons, where 0 = transparent and 1 = opaque (see transco).
qnts.smooth	If TRUE, use a lowess smoother on quantile polygons.
leg	If TRUE, plot default legend.
leg.cex	Legend scale size.
axs	Plot axis type (see par for details), with default to "r".
add	If TRUE, new plot window is not called and lines are added to existing plot window.
...	additional arguments to pass.

See Also

icm

Examples

```
## Not run:
## Example 1: Plotting multiple compartment values from SIR model
param <- param.icm(inf.prob = 0.5, act.rate = 0.5, rec.rate = 0.02)
init <- init.icm(s.num = 500, i.num = 1, r.num = 0)
control <- control.icm(type = "SIR", nsteps = 100,
                       nsims = 3, verbose = FALSE)
mod <- icm(param, init, control)
plot(mod)

## Example 2: Plot only infected with specific output from SI model
param <- param.icm(inf.prob = 0.25, act.rate = 0.25)
init <- init.icm(s.num = 500, i.num = 10)
control <- control.icm(type = "SI", nsteps = 100,
                       nsims = 3, verbose = FALSE)
mod2 <- icm(param, init, control)

# Plot prevalence
plot(mod2, y = "i.num", mean.line = FALSE)

# Plot incidence
plot(mod2, y = "si.flow", mean.smooth = TRUE)
plot(mod2, y = "si.flow", qnts.smooth = FALSE, qnts = 1)

## End(Not run)
```

plot.netdx *Plot Dynamic Network Model Diagnostics*

Description

Plots dynamic network model diagnostics calculated in netdx.

Usage

```
## S3 method for class 'netdx'
plot(x, type = "formation", method = "l", sims, stats,
  sim.lines, sim.col, sim.lwd, mean.line = TRUE, mean.smooth = TRUE,
  mean.col, mean.lwd = 2, mean.lty = 1, qnts = 0.5, qnts.col, qnts.alpha,
  qnts.smooth = TRUE, targ.line = TRUE, targ.col, targ.lwd = 2,
  targ.lty = 2, plots.joined, leg, ...)
```

Arguments

x	An EpiModel object of class netdx.
type	Plot type, with options of "formation" for network model formation statistics, "duration" for dissolution model statistics for average edge duration, or "dissolution" for dissolution model statistics for proportion of ties dissolved per time step.
method	Plot method, with options of "l" for line plots and "b" for boxplots.
sims	If type="epi" or "formation", a vector of simulation numbers to plot. If type="network", a single simulation number for network plot, or else "min" to plot the simulation number with the lowest disease prevalence, "max" for the simulation with the highest disease prevalence, or "mean" for the simulation with the prevalance closest to the mean across simulations at the specified time step.
stats	Network statistics to plot, among those specified in the call to netdx, with the default to plot all statistics contained in the object.
sim.lines	If TRUE, plot individual simulation lines. Default is to plot lines for one-group models but not for two-group models.
sim.col	Vector of any standard R color format for simulation lines.
sim.lwd	Line width for simulation lines.
mean.line	If TRUE, plot mean of simulations across time.
mean.smooth	If TRUE, use a lowess smoother on the mean line.
mean.col	Vector of any standard R color format for mean lines.
mean.lwd	Line width for mean lines.
mean.lty	Line type for mean lines.
qnts	If numeric, plot polygon of simulation quantiles based on the range implied by the argument (see details). If FALSE, suppress polygon from plot.

qnts.col	Vector of any standard R color format for polygons.
qnts.alpha	Transparency level for quantile polygons, where 0 = transparent and 1 = opaque (see transco).
qnts.smooth	If TRUE, use a lowess smoother on quantile polygons.
targ.line	If TRUE, plot target or expected value line for the statistic of interest.
targ.col	Vector of standard R colors for target statistic lines, with default colors based on RColorBrewer color palettes.
targ.lwd	Line width for the line showing the target statistic values.
targ.lty	Line type for the line showing the target statistic values.
plots.joined	If TRUE and type="formation", combine all target statistics in one plot, versus one plot per target statistic if FALSE.
leg	If TRUE, plot default legend.
...	additional arguments to pass.

Details

The plot function for netdx objects will generate plots of two types of model diagnostic statistics that run as part of the diagnostic tools within that function. The formation plot shows the summary statistics requested in nwstats.formula, where the default includes those statistics in the network model formation formula specified in the original call to netest.

The duration plot shows the average age of existing edges at each time step, up until the maximum time step requested. This is calculated with the edgelist_meanage function. The age is used as an estimator of the average duration of edges in the equilibrium state.

The dissolution plot shows the proportion of the extant ties that are dissolved at each time step, up until the maximum time step requested. Typically the proportion of ties that are dissolved is the reciprocal of the mean relational duration. This plot thus contains similar information to that in the duration plot, but should reach its expected value more quickly, since it is not subject to censoring.

The plots.joined argument will control whether the statistics in the formation plot are joined in one plot or plotted separately. The default is based on the number of network statistics requested. The layout of the separate plots within the larger plot window is also based on the number of statistics.

See Also

netdx

Examples

```
## Not run:
# Network initialization and model parameterization
nw <- network.initialize(100, directed = FALSE)
nw <- set.vertex.attribute(nw, "sex", rbinom(100, 1, 0.5))
formation <- ~edges + nodematch("sex")
target.stats <- c(50, 40)
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 50)
```

```
# Estimate the model
est <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

# Static diagnostics
dx1 <- netdx(est, nsims = 1e4, dynamic = FALSE,
             nwstats.formula = ~edges + meandeg + concurrent +
                               nodefactor("sex", base = 0) +
                               nodematch("sex"))
dx1

# Only formation diagnostics are available to plot
plot(dx1, stats = "edges")
plot(dx1, stats = c("edges", "concurrent"))
plot(dx1, stats = "edges", method = "b", col = "seagreen3")
plot(dx1, stats = c("nodefactor.sex.0", "nodefactor.sex.1"),
     method = "b", col = transco(2:3, 0.5))

# Dynamic diagnostics
dx2 <- netdx(est, nsims = 10, nsteps = 500,
             nwstats.formula = ~edges + meandeg + concurrent +
                               nodefactor("sex", base = 0) +
                               nodematch("sex"))
dx2

# Formation statistics plots, joined and separate
plot(dx2)
plot(dx2, type = "formation", plots.joined = TRUE)
plot(dx2, type = "formation", sim = 1, plots.joined = TRUE,
     qnts = FALSE, sim.lines = TRUE, mean.line = FALSE)
plot(dx2, type = "formation", plots.joined = FALSE,
     stats = c("edges", "concurrent"))
plot(dx2, type = "formation", stats = "nodefactor.sex.0",
     sim = 1, sim.lwd = 5, sim.col = "darkmagenta")

plot(dx2, method = "b", col = "bisque")
plot(dx2, method = "b", stats = "meandeg", col = "dodgerblue")

# Duration statistics plot
plot(dx2, type = "duration", mean.col = "black")
plot(dx2, type = "duration", sim = 10, mean.line = FALSE, sim.line = TRUE,
     sim.col = "steelblue", sim.lwd = 3, targ.lty = 1, targ.lwd = 0.5)

# Dissolution statistics plot
plot(dx2, type = "dissolution", mean.col = "black")
plot(dx2, type = "dissolution", method = "b", col = "pink1")

## End(Not run)
```

`plot.netsim` *Plot Data from a Stochastic Network Epidemic Model*

Description

Plots epidemiological and network data from a stochastic network model simulated with `netsim`.

Usage

```
## S3 method for class 'netsim'
plot(x, type = "epi", y, popfrac, sim.lines = FALSE, sims,
  sim.col, sim.lwd, sim.alpha, mean.line = TRUE, mean.smooth = TRUE,
  mean.col, mean.lwd = 2, mean.lty = 1, qnts = 0.5, qnts.col, qnts.alpha,
  qnts.smooth = TRUE, leg, leg.cex = 0.8, axs = "r", add = FALSE,
  network = 1, at = 1, col.status = FALSE, shp.bip = NULL, stats,
  targ.line = TRUE, targ.col, targ.lwd = 2, targ.lty = 2, plots.joined,
  ...)
```

Arguments

`x`	An `EpiModel` model object of class `netsim`.
`type`	Type of plot: `"epi"` for epidemic model results, `"network"` for a static network plot (`plot.network`), or `"formation"` for network formation statistics.
`y`	Output compartments or flows from `icm` object to plot.
`popfrac`	If `TRUE`, plot prevalence of values rather than numbers (see details).
`sim.lines`	If `TRUE`, plot individual simulation lines. Default is to plot lines for one-group models but not for two-group models.
`sims`	If `type="epi"` or `"formation"`, a vector of simulation numbers to plot. If `type="network"`, a single simulation number for network plot, or else `"min"` to plot the simulation number with the lowest disease prevalence, `"max"` for the simulation with the highest disease prevalence, or `"mean"` for the simulation with the prevalance closest to the mean across simulations at the specified time step.
`sim.col`	Vector of any standard R color format for simulation lines.
`sim.lwd`	Line width for simulation lines.
`sim.alpha`	Transparency level for simulation lines, where 0 = transparent and 1 = opaque (see `transco`).
`mean.line`	If `TRUE`, plot mean of simulations across time.
`mean.smooth`	If `TRUE`, use a lowess smoother on the mean line.
`mean.col`	Vector of any standard R color format for mean lines.
`mean.lwd`	Line width for mean lines.
`mean.lty`	Line type for mean lines.
`qnts`	If numeric, plot polygon of simulation quantiles based on the range implied by the argument (see details). If `FALSE`, suppress polygon from plot.
`qnts.col`	Vector of any standard R color format for polygons.
`qnts.alpha`	Transparency level for quantile polygons, where 0 = transparent and 1 = opaque (see `transco`).

qnts.smooth	If TRUE, use a lowess smoother on quantile polygons.
leg	If TRUE, plot default legend.
leg.cex	Legend scale size.
axs	Plot axis type (see par for details), with default to "r".
add	If TRUE, new plot window is not called and lines are added to existing plot window.
network	Network number, for simulations with multiple networks representing the population.
at	If type="network", time step for network graph.
col.status	If TRUE and type="network", automatic disease status colors (blue = susceptible, red = infected, , green = recovered).
shp.bip	If type="network" and a bipartite simulation, shapes for the mode 2 vertices, with acceptable inputs of "triangle" and "square". Mode 1 vertices will be circles.
stats	If type="formation", network statistics to plot, among those specified in nwstats.formula of control.net, with the default to plot all statistics.
targ.line	If TRUE, plot target or expected value line for the statistic of interest.
targ.col	Vector of standard R colors for target statistic lines, with default colors based on RColorBrewer color palettes.
targ.lwd	Line width for the line showing the target statistic values.
targ.lty	Line type for the line showing the target statistic values.
plots.joined	If TRUE and type="formation", combine all target statistics in one plot, versus one plot per target statistic if FALSE.
...	additional arguments to pass.

Details

This plot function can produce three types of plots with a stochastic network model simulated through netsim:

1. type="epi": epidemic model results (e.g., disease prevalence and incidence) may be plotted.
2. type="network": a static network plot will be generated. A static network plot of a dynamic network is a cross-sectional extraction of that dynamic network at a specific time point. This plotting function wraps the plot.network function in the network package. Consult the help page for plot.network for all the plotting parameters. In addition, four plotting parameters specific to netsim plots are available: sim, at, col.status, and shp.bip.
3. type="formation": summary network statistics related to the network model formation are plotted. These plots are similar to the formation plots for netdx objects. When running a netsim simulation, one must specify there that save.nwstats=TRUE; the plot here will then show the network statistics requested explicitly in nwstats.formula, or will use the formation formula set in netest otherwise.

When type="epi", this plotting function will extract the epidemiological output from a model object of class netsim and plot the time series data of disease prevalence and other results. The summary statistics that the function calculates and plots are individual simulation lines, means of the individual simulation lines, and quantiles of those individual simulation lines. The mean line, toggled on with mean.line=TRUE is calculated as the row mean across simulations at each time step.

Compartment prevalences are the size of a compartment over some denominator. To plot the raw numbers from any compartment, use popfrac=FALSE; this is the default for any plots of flows. The popfrac parameter calculates and plots the denominators of all specified compartments using these rules: 1) for one-group models, the prevalence of any compartment is the compartment size divided by the total population size; 2) for two-group models, the prevalence of any compartment is the compartment size divided by the group population size.

The quantiles show the range of outcome values within a certain specified quantile range. By default, the interquartile range is shown: that is the middle 50% of the data. This is specified by qnts=0.5. To show the middle 95% of the data, specify qnts=0.95. To toggle off the polygons where they are plotted by default, specify qnts=FALSE.

See Also

plot.network

Examples

```
## Not run:
## Independent SI Model
# Initialize network and set network model parameters
nw <- network.initialize(n = 100, bipartite = 50, directed = FALSE)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)

# Estimate the network model
est <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

# Simulate the epidemic model
param <- param.net(inf.prob = 0.3, inf.prob.m2 = 0.15)
init <- init.net(i.num = 10, i.num.m2 = 10)
control <- control.net(type = "SI", nsteps = 100, nsims = 5,
                       verbose = FALSE, save.nwstats = TRUE,
                       nwstats.formula = ~edges + meandeg + concurrent)
mod <- netsim(est, param, init, control)

# Plot epidemic trajectory (default type)
plot(mod, type = "epi")
plot(mod, type = "epi", popfrac = FALSE)
plot(mod, type = "epi", y = "si.flow", qnts = 1)

# Plot static networks
par(mar = c(0,0,0,0))
plot(mod, type = "network")
```

```
# Automatic coloring of infected nodes as red
par(mfrow = c(1, 2), mar = c(0, 0, 2, 0))
plot(mod, type = "network", main = "Min Prev | Time 50",
     col.status = TRUE, at = 50, sims = "min")
plot(mod, type = "network", main = "Max Prev | Time 50",
     col.status = TRUE, at = 50, sims = "max")

# Automatic shape by mode number (circle = mode 1)
par(mar = c(0,0,0,0))
plot(mod, type = "network", at = 50, col.status = TRUE, shp.bip = "square")
plot(mod, type = "network", at = 50, col.status = TRUE, shp.bip = "triangle")

# Plot formation statistics
par(mfrow = c(1,1), mar = c(3,3,1,1), mgp = c(2,1,0))
plot(mod, type = "formation")
plot(mod, type = "formation", plots.joined = FALSE)
plot(mod, type = "formation", sim = 2:4)
plot(mod, type = "formation", plots.joined = FALSE,
     stats = c("edges", "concurrent"))
plot(mod, type = "formation", stats = "meandeg",
     sim.lwd = 2, sim.col = "seagreen")

## End(Not run)
```

`plot.transmat` *Plot transmat infection tree in one of several styles*

Description

Plots the infection tree described in a `transmat` object in one of several styles: phylogentic tree, an un-rooted network, a hierarchical tree, or a transmissionTimeline.

Usage

```
## S3 method for class 'transmat'
plot(x, style = c("phylo", "network",
  "transmissionTimeline"), ...)
```

Arguments

`x`	a `transmat` object to be plotted
`style`	character name of plot style. One of "phylo", "network", or "transmissionTimeline"
`...`	additional plot arguments to be passed to lower-level plot functions (plot.network, etc)

Details

The phylo plot requires the ape package. The ndtv::transmissionTimeline requires that the ndtv package is installed. All of the options are essentially wrappers to other plot calls with some appropriate preset arguments.

See Also

plot.network,plot.phylo

---

summary.dcm *Summary Model Statistics*

Description

Extracts and prints model statistics solved with dcm.

Usage

```
## S3 method for class 'dcm'
summary(object, at, run = 1, digits = 3, ...)
```

Arguments

object	An EpiModel object of class dcm.
at	Time step for model statistics.
run	Model run number, for dcm class models with multiple runs (sensitivity analyses).
digits	Number of significant digits to print.
...	Additional summary function arguments (not used).

Details

Summary statistics for the main epidemiological outcomes (state and transition size and prevalence) from an dcm model. Time-specific summary measures are provided, so it is necessary to input a time of interest. For multiple-run models (sensitivity analyses), input a model run number. See examples below.

See Also

dcm

Examples

```
## Deterministic SIR model with varying act.rate
param <- param.dcm(inf.prob = 0.2, act.rate = 2:4, rec.rate = 1/3,
                   b.rate = 0.011, ds.rate = 0.01,
                   di.rate = 0.03, dr.rate = 0.01)
init <- init.dcm(s.num = 1000, i.num = 1, r.num = 0)
control <- control.dcm(type = "SIR", nsteps = 50)
mod <- dcm(param, init, control)
summary(mod, at = 25, run = 1)
summary(mod, at = 25, run = 3)
summary(mod, at = 26, run = 3)
```

---

summary.icm *Summary Model Statistics*

Description

Extracts and prints model statistics simulated with `icm`.

Usage

```
## S3 method for class 'icm'
summary(object, at, digits = 3, ...)
```

Arguments

`object`	An `EpiModel` object of class `icm`.
`at`	Time step for model statistics.
`digits`	Number of significant digits to print.
`...`	Additional summary function arguments.

Details

Summary statistics for the main epidemiological outcomes (state and transition size and prevalence) from an `icm` model. Time-specific summary measures are provided, so it is necessary to input a time of interest.

See Also

`icm`

Examples

```
## Stochastic ICM SI model with 3 simulations
param <- param.icm(inf.prob = 0.2, act.rate = 1)
init <- init.icm(s.num = 500, i.num = 1)
control <- control.icm(type = "SI", nsteps = 50,
                       nsims = 5, verbose = FALSE)
mod <- icm(param, init, control)
summary(mod, at = 25)
summary(mod, at = 50)
```

`summary.netsim` *Summary Model Statistics*

Description

Extracts and prints model statistics simulated with `netsim`.

Usage

```
## S3 method for class 'netsim'
summary(object, at, digits = 3, ...)
```

Arguments

`object`	An `EpiModel` object of class `netsim`.
`at`	Time step for model statistics.
`digits`	Number of significant digits to print.
`...`	Additional summary function arguments.

Details

Summary statistics for the main epidemiological outcomes (state and transition size and prevalence) from an `netsim` model. Time-specific summary measures are provided, so it is necessary to input a time of interest.

See Also

`netsim`

Examples

```
## Not run:
## Independent SI Model
# Initialize network and set network model parameters
nw <- network.initialize(n = 100, bipartite = 50, directed = FALSE)
formation <- ~edges
target.stats <- 50
```

```
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)

# Estimate the ERGM models (see help for netest)
# Skipping model diagnostics for this, but one should always run these
est1 <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

# Parameters, initial conditions, and controls for model
param <- param.net(inf.prob = 0.3, inf.prob.m2 = 0.15)
init <- init.net(i.num = 10, i.num.m2 = 10)
control <- control.net(type = "SI", nsteps = 100, nsims = 5, verbose.int = 0)

# Run the model simulation
mod <- netsim(est1, param, init, control)

summary(mod, at = 1)
summary(mod, at = 50)
summary(mod, at = 100)

## End(Not run)
```

Index