Package 'fitR'

July 2, 2019

Title Tool box for fitting dynamic infectious disease models to time series.

Version 0.1

Description This package contains helper functions for model fitting and
inference

Depends R (>= 3.0.2),
deSolve,
adaptivetau

Imports ggplot2,
reshape2,
plyr,
tmvtnorm,
parallel,
doParallel,
lubridate,
coda,
stringr,
lattice,
dplyr,
truncnorm,
emdbook

License GPL-2

LazyData true

Roxygen list(wrap = FALSE)

RoxygenNote 6.1.1

R topics documented:

| burnAndThin | *Burn and thin MCMC chain* |

Description

Return a burned and thined trace of the chain.

Usage

```
burnAndThin(trace, burn = 0, thin = 0)
```

Arguments

trace	either a `data.frame` or a `list` of `data.frame` with all variables in column, as outputed by [mcmcMH](). Accept also an `mcmc` or `mcmc.list` object.
burn	proportion of the chain to burn.
thin	number of samples to discard per sample that is being kept

Value

an object with the same format as `trace` (`data.frame` or `list` of `data.frame` or `mcmc` object or `mcmc.list` object)

---

computeDIC *Compute the DIC*

Description

This function computes the Deviance Information Criterion (DIC) of a [fitmodel](#) from a MCMC sample.

Usage

```
computeDIC(trace, fitmodel, init.state, data, margLogLike = dTrajObs,
  ...)
```

Arguments

trace	either a data.frame or mcmc object. Must contain one column with the posterior log.likelihood.
fitmodel	a [fitmodel](#) object
init.state	named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.
data	data frame. Observation times and observed data. The time column must be named "time" and the observation column must be named "obs".
margLogLike	R-function to compute the marginal log-likelihood of theta.
...	further arguments to be passed to margLogLike

Value

a list of 5 elements:

- DIC value of the DIC
- theta_bar mean posterior of theta
- log_like_theta_bar log-likelihood of theta_bar
- D_theta_bar deviance of theta_bar
- p_D effective number of parameters

---

computeDistanceABC *Compute the distance between a model and data for ABC*

Description

Compute the distance (using distance.ABC) between the observed time series and a simulated time series of observations obtained by running the model with parameters theta.

Usage

```
computeDistanceABC(sum.stats, distanceABC, fitmodel, theta, init.state,
  data)
```

Arguments

`sum.stats`	a list of functions to calculate summary statistics. Each of these takes one argument (a trajectory with an "obs" column) and returns a number (the summary statistic given the trajectory)
`distanceABC`	a function that take three arguments: `sum.stats`, a list of summary statistics, `data.obs` (the data trajectory of observations) and `model.obs` (a model trajectory of observations), and returns the distance between the model run and the data in terms of the summary statistics
`fitmodel`	a fitmodel object
`theta`	named numeric vector. Values of the parameters. Names should match `fitmodel$theta.names`.
`init.state`	named numeric vector. Initial values of the state variables. Names should match `fitmodel$state.names`.
`data`	data frame. Observation times and observed data. The time column must be named ″`time`″ and the observation column must be named ″`obs`″.

Value

a sampled distance between

---

distanceOscillation   *Distance weighted by number of oscillations*

Description

This positive distance is the mean squared differences between x and the y, divided by the square of the number of times the x oscillates around the y (see note below for illustration).

Usage

```
distanceOscillation(x, y)
```

Arguments

`x, y`	numeric vectors of the same length.

Note

To illustrate this distance, suppose we observed a time series `y = c(1,3,5,7,5,3,1)` and we have two simulated time series `x1 = (3,5,7,9,7,5,3)` and `x2 = (3,5,3,5,7,5,3)`; x1 is consistently above y and x2 oscillates around y. While the squared differences are the same, we obtain $d(y, x1) = 4$ and $d(y, x2) = 1.3$.

---

dLogPosterior *Posterior distribution for a fitmodel*

Description

This function evaluates the posterior distribution at `theta` and returns the result in a suitable format for mcmcMH.

Usage

```
dLogPosterior(fitmodel, theta, init.state, data, margLogLike = dTrajObs,
  ...)
```

Arguments

fitmodel	a fitmodel object
theta	named numeric vector. Values of the parameters. Names should match `fitmodel$theta.names`.
init.state	named numeric vector. Initial values of the state variables. Names should match `fitmodel$state.names`.
data	data frame. Observation times and observed data. The time column must be named "time" and the observation column must be named "obs".
margLogLike	R-function to compute the marginal log-likelihood of `theta`.
...	further arguments to be passed to `margLogLike`

Value

a list of two elements

- `log.density` numeric, logged value of the posterior density evaluated at `theta`
- `trace` named vector with trace information (theta, log.prior, marg.log.like, log.posterior)

See Also

dTrajObs, margLogLikeSto

---

dTrajObs *Log-likelihood of a trajectory for a deterministic model*

Description

Compute the trajectory (log-)likelihood of `theta` for a deterministic model defined in a fitmodel object by summing the point log-likelihoods.

Usage

```
dTrajObs(fitmodel, theta, init.state, data, log = FALSE)
```

Arguments

fitmodel	a [fitmodel](#) object
theta	named numeric vector. Values of the parameters. Names should match fitmodel$theta.names.
init.state	named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.
data	data frame. Observation times and observed data. The time column must be named "time" and the observation column must be named "obs".
log	logical (default: FALSE); whether the logarithm of the likelihood should be returned

Value

numeric value of the log-likelihood

See Also

[rTrajObs](#)

---

fitmodel   *Constructor of fitmodel object*

Description

A fitmodel object is a list that stores some variables and functions that will be useful to simulate and fit your model during the course.

Usage

```
fitmodel(name = NULL, state.names = NULL, theta.names = NULL,
  simulate = NULL, rPointObs = NULL, dprior = NULL,
  dPointObs = NULL)
```

Arguments

name	character. Name of the model (required).
state.names	character vector. Names of the state variables i.e. c("S","I","R") (required).
theta.names	character vector. Names of the parameters i.e. c("R0","infectious.period") (required).
simulate	R-function to simulate forward the model (required). This function takes 3 arguments:
	• theta named numeric vector. Values of the parameters. Names should match theta.names.
	• init.state named numeric vector. Initial values of the state variables. Names should match state.names.
	• times numeric vector. Time sequence for which the state of the model is wanted; the first value of times must be the initial time, i.e. the time of init.state.

and returns a `data.fame` containing the simulated trajectories that is the values of the state variables (1 per column) at each observation time (1 per row). The first column is `time`.

rPointObs   R-function that generates a (randomly sampled) observation point from a model point, using an observation model (optional). It thus acts as an inverse of `dPointObs` (see below). This function takes 2 arguments

- `model.point` named numeric vector. State of the model at a given point in time.
- `theta` named numeric vector. Values of the parameters. Names should match `theta.names`.

and returns an observation point

dprior   R-function that evaluates the prior density of the parameters at a given `theta` (optional). The function should take 2 arguments:

- `theta` named numeric vector. Values of the parameters. Names should match `theta.names`.
- `log` boolean. determines whether the logarithm of the prior density should be returned.

and returns the (logged, if requested) value of the prior density distribution.

dPointObs   R-function that evaluates the likelihood of one data point given the state of the model at the same time point. This function takes 4 arguments:

- `data.point` named numeric vector. Observation time and observed data point.
- `model.point` named numeric vector containing the state of the model at the observation time point.
- `theta` named numeric vector. Parameter values. Useful since parameters are usually needed to compute the likelihood (i.e. reporting rate).
- `log` boolean. determines whether the logarithm of the likelihood should be returned.

and returns the (log-)likelihood. (optional)

Value

a `fitmodel` object that is a `list` of 7 elements:

- `name` character, name of the model
- `state.names` vector, names of the state variables.
- `theta.names` vector, names of the parameters.
- `simulate` R-function to simulate forward the model; usage: `simulate(theta,init.state,times)`.
- `rPointObs` R-function to generate simulated observations; usage: `rPointObs(model.point, theta)`.
- `dprior` R-function to evaluate the log-prior of the parameter values; usage: `dprior(theta)`.
- `dPointObs` R-function to evaluate the log-likelihood of one data point; usage: `dPointObs(data.point, model.po`

See Also

[testFitmodel](#)

Examples

```
## create a simple deterministic SIR model with constant population size

SIR_name <- "SIR with constant population size"
SIR_state.names <- c("S","I","R")
SIR_theta.names <- c("R0","D_inf")

SIR_simulateDeterministic <- function(theta,init.state,times) {

    SIR_ode <- function(time, state, parameters) {

        ## parameters
        beta <- parameters[["R0"]] / parameters[["D_inf"]]
        nu <- 1 / parameters[["D_inf"]]

        ## states
        S <- state[["S"]]
        I <- state[["I"]]
        R <- state[["R"]]

        N <- S + I + R

        dS <- -beta * S * I/N
        dI <- beta * S * I/N - nu * I
        dR <- nu * I

        return(list(c(dS, dI, dR)))
    }

    trajectory <- data.frame(ode(y = init.state,
                                 times = times,
                                 func = SIR_ode,
                                 parms = theta,
                                 method = "ode45"))

    return(trajectory)
}

## function to compute log-prior
SIR_prior <- function(theta, log = FALSE) {

    ## uniform prior on R0: U[1,100]
    log.prior.R0 <- dunif(theta[["R0"]], min = 1, max = 100, log = TRUE)
    ## uniform prior on infectious period: U[0,30]
    log.prior.D <- dunif(theta[["D_inf"]], min = 0, max = 30, log = TRUE)

    log.sum <- log.prior.R0 + log.prior.D

    return(ifelse(log, log.sum, exp(log.sum)))
}

## function to compute the likelihood of one data point
SIR_pointLike <- function(data.point, model.point, theta, log = FALSE){

    ## the prevalence is observed through a Poisson process
    return(dpois(x = data.point[["obs"]],
```

```
                    lambda = model.point[["I"]],
                    log = log))
}

## function to generate observation from a model simulation
SIR_genObsPoint <- function(model.point, theta){

    ## the prevalence is observed through a Poisson process
    obs.point <- rpois(n = 1, lambda = model.point[["I"]])

    return(c(obs = obs.point))
}

## create deterministic SIR fitmodel
SIR <- fitmodel(
    name = SIR_name,
    state.names = SIR_state.names,
    theta.names = SIR_theta.names,
    simulate = SIR_simulateDeterministic,
    dprior = SIR_prior,
    rPointObs = SIR_genObsPoint,
    dPointObs = SIR_pointLike)

## test
## theta <- c(R0 = 3, D_inf = 2)
## init.state <- c(S = 999, I = 1, R = 0)
## data(epi)
```

FluTdC1971   *Time-series of the 1971 influenza epidemic in Tristan-da-Cunha*

Description

A dataset containing the daily incidence recorded during the 1971 influenza A/H3N2 two-wave epidemic on the island of Tristan-da-Cunha.

Format

A data frame with 59 rows and 3 variables

Details

- date calendar date of the record
- time day of record since beginning of epidemic
- Inc daily count incidence of influenza-like-illness

Source

http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=4511951&
retmode=ref&cmd=prlinks

margLogLikeSto   *Marginal log-likelihood for a stochastic model*

Description

Compute a Monte-Carlo estimate of the log-likelihood of theta for a stochastic model defined in a
fitmodel object, using particleFilter

Usage

```
margLogLikeSto(fitmodel, theta, init.state, data, n.particles,
  n.cores = 1)
```

Arguments

fitmodel	a fitmodel object
theta	named numeric vector. Values of the parameters. Names should match fitmodel$theta.names.
init.state	named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.
data	data frame. Observation times and observed data. The time column must be named "time" and the observation column must be named "obs".
n.particles	number of particles
n.cores	number of cores on which propogation of the particles is parallelised. By default no parallelisation (n.cores=1). If NULL, set to the value returned by detectCores.

Value

Monte-Carlo estimate of the marginal log-likelihood of theta

See Also

particleFilter

mcmcMH   *Metropolis-Hasting MCMC*

Description

Run n.iterations of a Metropolis-Hasting MCMC to sample from the target distribution using
a gaussian proposal kernel. Two optional optimizations are also implemented: truncated gaussian
proposal (to match the support of the target distribution, i.e. boundary of the parameters) and
adaptative gaussian proposal (to match the size and the shape of the target distribution).

Usage

```
mcmcMH(target, init.theta, proposal.sd = NULL, n.iterations,
  covmat = NULL, limits = list(lower = NULL, upper = NULL),
  adapt.size.start = NULL, adapt.size.cooling = 0.99,
  adapt.shape.start = NULL, adapt.shape.stop = NULL,
  print.info.every = n.iterations/100, verbose = FALSE,
  max.scaling.sd = 50)
```

Arguments

target	R-function that takes a single argument: theta (named numeric vector of parameter values) and returns a list of 2 elements:

- log.density the logged value of the target density, evaluated at theta.
- trace a named numeric vector of values to be printed in the trace data.frame returned by mcmcMH.

init.theta	named vector of initial parameter values to start the chain.
proposal.sd	vector of standard deviations. If this is given and covmat is not, a diagonal matrix will be built from this to use as covariance matrix of the multivariate Gaussian proposal distribution. By default, this is set to init.theta/10.
n.iterations	number of iterations to run the MCMC chain.
covmat	named numeric covariance matrix of the multivariate Gaussian proposal distribution. Must have named rows and columns with at least all estimated theta. If proposal.sd is given, this is ignored.
limits	limits for the - potentially truncated - multi-variate normal proposal distribution of the MCMC. Contains 2 elements:

- lower named numeric vector. Lower truncation points in each dimension of the Gaussian proposal distribution. By default they are set to -Inf.
- upper named numeric vector. Upper truncation points in each dimension of the Gaussian proposal distribution. By default they are set to Inf.

adapt.size.start	
	number of iterations to run before adapting the size of the proposal covariance matrix (see note below). Set to 0 (default) if size is not to be adapted.
adapt.size.cooling	
	cooling factor for the scaling factor of the covariance matrix during size adaptation (see note below).
adapt.shape.start	
	number of accepted jumps before adapting the shape of the proposal covariance matrix (see note below). Set to 0 (default) if shape is not to be adapted
print.info.every	
	frequency of information on the chain: acceptance rate and state of the chain. Default value to n.iterations/100. Set to NULL to avoid any info.
verbose	logical. If TRUE, information are printed.
max.scaling.sd	numeric. Maximum value for the scaling factor of the covariance matrix. Avoid too high values for the scaling factor, which might happen due to the exponential update scheme. In this case, the covariance matrix becomes too wide and the sampling from the truncated proposal kernel becomes highly inefficient

Value

a list with 3 elements:

- trace a data.frame. Each row contains a state of the chain (as returned by target, and an extra column for the log.density).
- acceptance.rate acceptance rate of the MCMC chain.
- covmat.proposal last covariance matrix used for proposals.

Note

The size of the proposal covariance matrix is adapted using the following formulae:

$$\Sigma_{n+1} = \sigma_n * \Sigma_n$$

with $\sigma_n = \sigma_{n-1} * exp(\alpha^n * (acc - 0.234))$, where α is equal to `adapt.size.cooling` and acc is the acceptance rate of the chain.

The shape of the proposal covariance matrix is adapted using the following formulae:

$$\Sigma_{n+1} = 2.38^2/d * \Sigma_n$$

with Σ_n the empirical covariance matrix and d is the number of estimated parameters in the model.

References

Roberts GO, Rosenthal JS. Examples of adaptive MCMC. Journal of Computational and Graphical Statistics. Taylor & Francis; 2009;18(2):349-67.

---

measles	*Time-series of a measles outbreak*

Description

A dataset containing the weekly incidence recorded during a recent outbreak of measles in Europe

Format

A data frame with 37 rows and 2 variables

Details

- `time` week of the record
- `Inc` weekly recorded incidence of measles

---

particleFilter	*Run a particle filter for fitmodel object*

Description

The particle filter returns an estimate of the marginal log-likelihood $L = p(y(t_{1:T})|\theta)$ as well as the set of filtered trajectories and their respective weights at the last observation time $\omega(t_T) = p(y(t_T)|\theta)$.

Usage

```
particleFilter(fitmodel, theta, init.state, data, n.particles,
  progress = FALSE, n.cores = 1)
```

Arguments

fitmodel	a [fitmodel](#) object
theta	named numeric vector. Values of the parameters. Names should match fitmodel$theta.names.
init.state	named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.
data	data frame. Observation times and observed data. The time column must be named "time" and the observation column must be named "obs".
n.particles	number of particles
progress	if TRUE progression of the filter is displayed in the console.
n.cores	number of cores on which propogation of the particles is parallelised. By default no parallelisation (n.cores=1). If NULL, set to the value returned by [detectCores](#).

Value

A list of 3 elements:

- dPointObs the marginal log-likelihood of the theta.
- traj a list of size n.particles with all filtered trajectories.
- traj.weight a vector of size n.particles with the normalised weight of the filtered trajectories.

Note

An unbiased state sample $x(t_{0:T})$ $p(X(t_{0:T})|\theta, y(t_{0:T}))$ can be obtained by sampling the set of trajectories traj with probability traj.weight.

See Also

plotSMC

---

plotESSBurn	*Plot Effective Sample Size (ESS) against burn-in*

Description

Takes an mcmc trace and tests the ESS at different values of burn-in

Usage

```
plotESSBurn(trace, longest.burn.in = ifelse(is.data.frame(trace) |
  is.mcmc(trace), nrow(trace), nrow(trace[[1]]))/2,
  step.size = round(longest.burn.in/50))
```

Arguments

trace	either a data.frame or a list of data.frame with all variables in column, as outputed by [mcmcMH](#). Accept also mcmc or mcmc.list objects.
longest.burn.in	
	The longest burn-in to test. Defaults to half the length of the trace
step.size	The size of the steps of burn-in to test. Defaults to 1/50th of longest.burn.in

Value

a plot of the ESS against burn.in

---

plotFit   *Plot fit of model to data*

Description

This function simulates the model under theta, generates observation and plot them against the data. Since simulation and observation processes can be stochastic, n.replicates can be plotted.

Usage

```
plotFit(fitmodel, theta, init.state, data, n.replicates = 1,
  summary = TRUE, alpha = min(1, 10/n.replicates), all.vars = FALSE,
  non.extinct = NULL, observation = TRUE, plot = TRUE)
```

Arguments

fitmodel	a fitmodel object
theta	named numeric vector. Values of the parameters. Names should match fitmodel$theta.names.
init.state	named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.
data	data frame. Observation times and observed data. The time column must be named "time" and the observation column must be named "obs".
n.replicates	numeric, number of replicated simulations.
summary	logical. If TRUE, the mean, median as well as the 50th and 95th percentile of the trajectories are plotted (default). If FALSE, all individual trajectories are plotted (transparency can be set with alpha).
alpha	transparency of the trajectories (between 0 and 1).
all.vars	logical, if FALSE only the observations are plotted. Otherwise, all state variables are plotted.
non.extinct	character vector. Names of the infected states which must be non-zero so the epidemic is still ongoing. When the names of these states are provided, the extinction probability is plotted by computing the proportion of faded-out epidemics over time. An epidemic has faded-out when all the infected states (whose names are provided) are equal to 0. This is only relevant for stochastic models. In addition, if summary == TRUE, the summaries of the trajectories conditioned on non-extinction are shown. Default to NULL.
observation	logical, if TRUE simulated observation are generated by rTrajObs.
plot	if TRUE the plot is displayed, and returned otherwise.

Value

if plot == FALSE, a list of 2 elements is returned:

- simulations data.frame of n.replicates simulated observations.
- plot the plot of the fit.

---

plotHPDregion2D   *2D highest posterior density region*

Description

Given a sample from a multivariate posterior distribution, plot the bivariate region of highest marginal posterior density (HPD) for two variables with defined levels.

Usage

```
plotHPDregion2D(trace, vars, prob = c(0.95, 0.75, 0.5, 0.25, 0.1),
  xlab = NULL, ylab = NULL, plot = TRUE)
```

Arguments

trace	either a data.frame or mcmc object.
vars	which variables to plot: numeric or character vector
prob	probability level
xlab	x axis label
ylab	y axis label
plot	if TRUE the plot is displayed, and returned otherwise.

Note

HPD levels are computed using the function HPDregionplot from the package emdbook.

---

plotPosteriorDensity   *Plot MCMC posterior densities*

Description

Plot the posterior density.

Usage

```
plotPosteriorDensity(trace, prior = NULL, colour = NULL, plot = TRUE)
```

Arguments

trace	either a data.frame or a list of data.frame with all variables in column, as outputed by mcmcMH. Accept also an mcmc, a mcmc.list object or a list of mcmc.list.
prior	a data.frame containing the prior density. It must have the three following columns: - theta names of the parameters - x value of the parameters - density density of the prior at x
colour	named vector of two characters and containing colour names for posterior and prior distributions. Vector names must be posterior and prior.
plot	if TRUE the plot is displayed, and returned otherwise.

See Also

burnAndThin

---

plotPosteriorFit   *Plot MCMC posterior fit*

Description

Plot posterior distribution of observation generated under model's posterior parameter distribution.

Usage

```
plotPosteriorFit(trace, fitmodel, init.state, data,
  posterior.summary = c("sample", "median", "mean", "max"),
  summary = TRUE, sample.size = 100, non.extinct = NULL,
  alpha = min(1, 10/sample.size), plot = TRUE, all.vars = FALSE,
  init.date = NULL)
```

Arguments

trace	a data.frame with one column per estimated parameter, as returned by [burnAndThin](#)
fitmodel	a [fitmodel](#) object
init.state	named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.
data	data frame. Observation times and observed data. The time column must be named "time" and the observation column must be named "obs".
posterior.summary	
	character. Set to "sample" to plot trajectories from a sample of the posterior (default). Set to "median", "mean" or "max" to plot trajectories corresponding to the median, mean and maximum of the posterior density.
summary	logical, if TRUE trajectories are summarised by their mean, median, 50% and 95% quantile distributions. Otheriwse, the trajectories are ploted.
sample.size	number of theta sampled from posterior distribution (if posterior.summary == "sample"). Otherwise, number of replicated simulations.
non.extinct	character vector. Names of the infected states which must be non-zero so the epidemic is still ongoing. When the names of these states are provided, the extinction probability is plotted by computing the proportion of faded-out epidemics over time. An epidemic has faded-out when all the infected states (whose names are provided) are equal to 0. This is only relevant for stochastic models. In addition, if summary == TRUE, the summaries of the trajectories conditioned on non-extinction are shown. Default to NULL.
alpha	transparency of the trajectories (between 0 and 1).
plot	if TRUE the plot is displayed, and returned otherwise.
all.vars	logical, if FALSE only the observations are plotted. Otherwise, all state variables are plotted.
init.date	character. Date of the first point of the time series (default to NULL). If provided, the x-axis will be in calendar format. NB: currently only works if the unit of time is the day.

Value

If `plot == FALSE`, a list of 2 elements is returned:

- `theta` the `theta`(s) used for plotting (either a `vector` or a `data.frame`)

- `traj` a `data.frame` with the trajectories (and observations) sampled from the posterior distribution.

- `plot` the plot of the fit displayed.

---

plotSMC   *Plot result of SMC*

Description

Plot the observation generated by the filtered trajectories together with the data.

Usage

```
plotSMC(smc, fitmodel, theta, data = NULL, summary = TRUE, alpha = 1,
  all.vars = FALSE, plot = TRUE)
```

Arguments

smc	output of [particleFilter](#)
fitmodel	a [fitmodel](#) object
theta	named numeric vector. Values of the parameters. Names should match `fitmodel$theta.names`.
data	data frame. Observation times and observed data. The time column must be named as given by `time.column`, whereas the name of the data column should match one of `fitmodel$state.names`.
summary	logical. If `TRUE`, the mean, median as well as the 50th and 95th percentile of the trajectories are plotted (default). If `FALSE`, all individual trajectories are plotted (transparency can be set with `alpha`).
alpha	transparency of the trajectories (between 0 and 1).
all.vars	logical, if `FALSE` only the observations are plotted. Otherwise, all state variables are plotted.
plot	if `TRUE` the plot is displayed, and returned otherwise.

See Also

particleFilter

---

plotTrace   *Plot MCMC trace*

Description

Plot the traces of all estimated variables.

Usage

```
plotTrace(trace, estimated.only = FALSE)
```

Arguments

trace   a data.frame with one column per estimated parameter, as returned by burnAndThin

estimated.only   logical, if TRUE only estimated parameters are displayed.

See Also

burnAndThin

---

plotTraj   *Plot model trajectories*

Description

This function use faceting to plot all trajectories in a data frame. Convenient to see results of several simulations, or data. Also, if data is present, then an additional plot is displayed with data and potentially observation generated.

Usage

```
plotTraj(traj = NULL, state.names = NULL, data = NULL,
  time.column = "time", lines.data = FALSE, summary = TRUE,
  replicate.column = "replicate", non.extinct = NULL, alpha = 1,
  plot = TRUE, colour = "red", init.date = NULL, same = FALSE)
```

Arguments

traj   data.frame, output of fitmodel$simulate or simulateModelReplicates.

state.names   character vector. Names of the state variables to plot. Names must match
                fitmodel$state.names. If NULL (default) all state variables are plotted.

data   data frame. Observation times and observed data. The time column must be
                named as given by time.column, whereas the name of the data column should
                match one of fitmodel$state.names.

time.column   character vector. The column in the data that indicates time

lines.data   logical. If TRUE, the data will be plotted as lines

summary   logical. If TRUE, the mean, median as well as the 50th and 95th percentile of the
                trajectories are plotted (default). If FALSE, all individual trajectories are plotted
                (transparency can be set with alpha).

replicate.column

        character Vector. The column in the data that indicates the replicate (if muliple replicates are to be plotted, i.e. if summary is FALSE

non.extinct   character vector. Names of the infected states which must be non-zero so the epidemic is still ongoing. When the names of these states are provided, the extinction probability is plotted by computing the proportion of faded-out epidemics over time. An epidemic has faded-out when all the infected states (whose names are provided) are equal to 0. This is only relevant for stochastic models. In addition, if summary == TRUE, the summaries of the trajectories conditioned on non-extinction are shown. Default to NULL.

alpha   transparency of the trajectories (between 0 and 1).

plot   if TRUE the plot is displayed, and returned otherwise.

colour   character vector. If a character, will use that colour to plot trajectories. If "all", use all available colous. If NULL, don't set the colour.

init.date   character. Date of the first point of the time series (default to NULL). If provided, the x-axis will be in calendar format. NB: currently only works if the unit of time is the day.

same   logical (default: FALSE); if TRUE, trajectories will be plotted in the same panel.

See Also

[simulateModelReplicates](#)

---

rTrajObs   *Generate an observation trajectory for a fitmodel*

Description

This function simulates a model defined in a [fitmodel](#) object and generates observations at each time point. It returns the trajectory with an additions obs column.

Usage

```
rTrajObs(fitmodel, theta, init.state, times)
```

Arguments

fitmodel   a [fitmodel](#) object

theta   named numeric vector. Values of the parameters. Names should match fitmodel$theta.names.

init.state   named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.

times   the times at which to generate observations

Value

data.frame

See Also

[dTrajObs](#)

SEIT2L_deter   *The deterministic SEIT2L model with constant population size*

Description

The deterministic SEIT2L model with constant population size, uniform prior and Poisson observation with reporting rate.

Format

A `fitmodel` object, that is a list with the following elements:

Details

- `name` character.
- `state.names` character vector.
- `theta.names` character vector.
- `simulate` R-function.
- `rPointObs` R-function.
- `dprior` R-function.
- `dPointObs` R-function.

Look at the documentation of `fitmodel` for more details about each of these elements. You can look at the code of the R-functions by typing SEIT2L_deter$simulate for instance. There are some comments included.

SEIT2L_stoch   *The stochastic SEIT2L model with constant population size*

Description

The stochastic SEIT2L model with constant population size, uniform prior and Poisson observation with reporting rate.

Format

A `fitmodel` object, that is a list with the following elements:

Details

- `name` character.
- `state.names` character vector.
- `theta.names` character vector.
- `simulate` R-function.
- `rPointObs` R-function.
- `dprior` R-function.
- `dPointObs` R-function.

Look at the documentation of `fitmodel` for more details about each of these elements. You can look at the code of the R-functions by typing SEIT2L_stoch$simulate for instance. There are some comments included.

SEIT4L_deter *The deterministic SEIT4L model with constant population size*

Description

The deterministic SEIT4L model with constant population size, uniform prior and Poisson observation with reporting rate.

Format

A [fitmodel](#) object, that is a list with the following elements:

Details

- name character.
- state.names character vector.
- theta.names character vector.
- simulate R-function.
- rPointObs R-function.
- dprior R-function.
- dPointObs R-function.

Look at the documentation of [fitmodel](#) for more details about each of these elements. You can look at the code of the R-functions by typing SEIT4L_deter$simulate for instance. There are some comments included.

SEIT4L_stoch *The stochastic SEIT4L model with constant population size*

Description

The stochastic SEIT4L model with constant population size, uniform prior and Poisson observation with reporting rate.

Format

A [fitmodel](#) object, that is a list with the following elements:

Details

- name character.
- state.names character vector.
- theta.names character vector.
- simulate R-function.
- rPointObs R-function.
- dprior R-function.
- dPointObs R-function.

Look at the documentation of [fitmodel](#) for more details about each of these elements. You can look at the code of the R-functions by typing SEIT4L_stoch$simulate for instance. There are some comments included.

SEITL_deter   *The deterministic SEITL model with constant population size*

Description

The deterministic SEITL model with constant population size, uniform prior and Poisson observation with reporting rate.

Format

A fitmodel object, that is a list with the following elements:

Details

- name character.
- state.names character vector.
- theta.names character vector.
- simulate R-function.
- rPointObs R-function.
- dprior R-function.
- dPointObs R-function.

Look at the documentation of fitmodel for more details about each of these elements. You can look at the code of the R-functions by typing SEITL_deter$simulate for instance. There are some comments included.

SEITL_stoch   *The stochastic SEITL model with constant population size*

Description

The stochastic SEITL model with constant population size, uniform prior and Poisson observation with reporting rate.

Format

A fitmodel object, that is a list with the following elements:

Details

- name character.
- state.names character vector.
- theta.names character vector.
- simulate R-function.
- rPointObs R-function.
- dprior R-function.
- dPointObs R-function.

Look at the documentation of fitmodel for more details about each of these elements. You can look at the code of the R-functions by typing SEITL_stoch$simulate for instance. There are some comments included.

simulateFinalStateAtExtinction

Simulate model until extinction

Description

Return final state at extinction

Usage

```
simulateFinalStateAtExtinction(fitmodel, theta, init.state,
  extinct = NULL, time.init = 0, time.step = 100, n = 100,
  observation = FALSE, n.cores = 1)
```

Arguments

fitmodel	a [fitmodel](#) object
theta	named numeric vector. Values of the parameters. Names should match fitmodel$theta.names.
init.state	named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.
extinct	character vetor. Simulations stop when all these state are extinct.
time.init	numeric. Start time of simulation.
time.step	numeric. Time step at which extinction is checked
n	number of replicated simulations.
observation	logical, if TRUE simulated observation are generated by rTrajObs.
n.cores	number of cores on which propogation of the particles is parallelised. By default no parallelisation (n.cores=1). If NULL, set to the value returned by [detectCores](#).

simulateModelReplicates

Simulate several replicate of the model

Description

Simulate several replicate of a fitmodel using its function simulate

Usage

```
simulateModelReplicates(fitmodel, theta, init.state, times, n,
  observation = FALSE)
```

Arguments

fitmodel	a [fitmodel](#) object
theta	named numeric vector. Values of the parameters. Names should match fitmodel$theta.names.
init.state	named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.
times	vector of times at which you want to observe the states of the model.
n	number of replicated simulations.
observation	logical, if TRUE simulated observation are generated by rTrajObs.

Value

a data.frame of dimension [nxlength(times)]x[length(init.state)+2] with column names equal to c("replicate","time",names(init.state)).

---

simulateModelStochastic

Simulate forward a stochastic model

Description

This function uses the function `ssa.adaptivetau` to simulate the model and returns the trajectories in a valid format for the class `fitmodel`.

Usage

```
simulateModelStochastic(theta, init.state, times, transitions, rateFunc)
```

Arguments

theta	named vector of model parameters.
init.state	named vector of initial state of the model.
times	time sequence for which state of the model is wanted; the first value of times must be the initial time.
transitions	One of two possible data types:

- A list with length equal to the number of transitions. Each element of the list should be a vector specifying a transition (i.e., which state(s) change and by how much). Each entry in the vector needs a name (specifying which state variable to change, either by name or index) and a value (specifying the amount by which this variable will change).
- A two-dimensional matrix of integers specifying how each state variable (rows) should be changed for a given transition (columns). Generally this will be a sparse matrix of primarily 1s and -1s, which can make this structure inefficient.

See the example below for details as well as `ssa.maketrans` or the vignette accompanying this package.

rateFunc	R function that returns instantaneous transition rates for each transition in the form a real-valued one-dimensional vector with length equal to the number of transitions. The order of these rates must match the order in which transitions were specified in the transitions parameter above. This function must accept the following arguments:

- vector of current values for all state variables (in order used in the init.values argument above)
- parameters as supplied in argument to ssa.adaptivetau
- single real number giving the current time (all simulations start at t=0)

Value

a data.frame of dimension length(times)x(length(init.state)+1) with column names equal to c("time",names(init.state)).

SIR   *A simple deterministic SIR model with constant population size*

Description

A simple deterministic SIR model with constant population size, uniform prior and Poisson observation.

Format

A [fitmodel] object, that is a list with the following elements:

Details

- name character.
- state.names character vector.
- theta.names character vector.
- simulate R-function.
- rPointObs R-function.
- dprior R-function.
- dPointObs R-function.

Look at the documentation of [fitmodel] for more details about each of these elements. You can look at the code of the R-functions by typing SIR$simulate for instance. There are some comments included.

SIR_exp   *A simple deterministic SIR model with constant population size and parameters on the exponential scale*

Description

A simple deterministic SIR model with constant population size, uniform prior and Poisson observation. The parameters are transformed using an exponential transformation.

Format

A [fitmodel] object, that is a list with the following elements:

Details

- name character.
- state.names character vector.
- theta.names character vector.
- simulate R-function.
- rPointObs R-function.
- dprior R-function.

- dPointObs R-function.

Look at the documentation of [fitmodel](#) for more details about each of these elements. You can look at the code of the R-functions by typing SIR_exp$simulate for instance. There are some comments included.

SIR_reporting   *A simple deterministic SIR model with constant population size and reporting rate*

Description

A simple deterministic SIR model with constant population size, uniform prior and Poisson observation with reporting rate.

Format

A [fitmodel](#) object, that is a list with the following elements:

Details

- name character.
- state.names character vector.
- theta.names character vector.
- simulate R-function.
- rPointObs R-function.
- dprior R-function.
- dPointObs R-function.

Look at the documentation of [fitmodel](#) for more details about each of these elements. You can look at the code of the R-functions by typing SIR_reporting$simulate for instance. There are some comments included.

SIR_stoch   *A simple stochastic SIR model with constant population size*

Description

A simple stochastic SIR model with constant population size, uniform prior and Poisson observation.

Format

A [fitmodel](#) object, that is a list with the following elements:

Details

- name character.
- state.names character vector.
- theta.names character vector.
- simulate R-function.
- rPointObs R-function.
- dprior R-function.
- dPointObs R-function.

Look at the documentation of [fitmodel](#) for more details about each of these elements. You can look at the code of the R-functions by typing SIR_stoch$simulate for instance. There are some comments included.

---

testFitmodel	*Test a fitmodel*

Description

This function performs a serie of checks on the [fitmodel](#) provided by the user in order to make sure that it will be compatible both with the functions coded during the course and the functions available in the fitR package. The latters can be used as a correction.

Usage

```
testFitmodel(fitmodel, theta, init.state, data = NULL, verbose = TRUE)
```

Arguments

fitmodel	a [fitmodel](#) object
theta	named numeric vector. Values of the parameters. Names should match fitmodel$theta.names.
init.state	named numeric vector. Initial values of the state variables. Names should match fitmodel$state.names.
data	data frame. Observation times and observed data. The time column must be named "time" and the observation column must be named "obs".
verbose	if TRUE, print details of the test performed to check validity of the arguments

See Also

[fitmodel](#)

Examples

```
## create a simple deterministic SIR model with constant population size

SIR_name <- "SIR with constant population size"
SIR_state.names <- c("S","I","R")
SIR_theta.names <- c("R0","D_inf")

SIR_simulateDeterministic <- function(theta,init.state,times) {
```

```r
    SIR_ode <- function(time, state, parameters) {

        ## parameters
        beta <- parameters[["R0"]] / parameters[["D_inf"]]
        nu <- 1 / parameters[["D_inf"]]

        ## states
        S <- state[["S"]]
        I <- state[["I"]]
        R <- state[["R"]]

        N <- S + I + R

        dS <- -beta * S * I/N
        dI <- beta * S * I/N - nu * I
        dR <- nu * I

        return(list(c(dS, dI, dR)))
    }

    trajectory <- data.frame(ode(y = init.state,
                                 times = times,
                                 func = SIR_ode,
                                 parms = theta,
                                 method = "ode45"))

    return(trajectory)
}

## function to compute log-prior
SIR_prior <- function(theta, log = FALSE) {

    ## uniform prior on R0: U[1,100]
    log.prior.R0 <- dunif(theta[["R0"]], min = 1, max = 100, log = TRUE)
    ## uniform prior on infectious period: U[0,30]
    log.prior.D <- dunif(theta[["D_inf"]], min = 0, max = 30, log = TRUE)

    log.sum <- log.prior.R0 + log.prior.D

    return(ifelse(log, log.sum, exp(log.sum)))
}

## function to compute the likelihood of one data point
SIR_pointLike <- function(data.point, model.point, theta, log = FALSE){

    ## the prevalence is observed through a Poisson process
    return(dpois(x = data.point[["obs"]],
                 lambda = model.point[["I"]],
                 log = log))
}

## function to generate observation from a model simulation
SIR_genObsPoint <- function(model.point, theta){

    ## the prevalence is observed through a Poisson process
    obs.point <- rpois(n = 1, lambda = model.point[["I"]])
```

```
        return(c(obs = obs.point))
    }

    ## create deterministic SIR fitmodel
    SIR <- fitmodel(
        name = SIR_name,
        state.names = SIR_state.names,
        theta.names = SIR_theta.names,
        simulate = SIR_simulateDeterministic,
        dprior = SIR_prior,
        rPointObs = SIR_genObsPoint,
        dPointObs = SIR_pointLike)

    ## test
    ## theta <- c(R0 = 3, D_inf = 2)
    ## init.state <- c(S = 999, I = 1, R = 0)
    ## data(epi)
```

Index