Package: smfsb (via r-universe)

September 11, 2024

Type Package

Title Stochastic Modelling for Systems Biology

Version 1.5

Date 2024-01-11

Author Darren Wilkinson

Maintainer Darren Wilkinson <darrenjwilkinson@btinternet.com>

Description Code and data for modelling and simulation of stochastic kinetic biochemical network models. It contains the code and data associated with the second and third editions of the book Stochastic Modelling for Systems Biology, published by Chapman & Hall/CRC Press.

License LGPL-3

Depends R (>= 2.9.0), abind (>= 1.3), parallel (>= 3.2.0)

Suggests deSolve (>= 1.9)

NeedsCompilation yes

Date/Publication 2024-01-13 13:00:04 UTC

Repository https://darrenjw.r-universe.dev

RemoteUrl https://github.com/cran/smfsb

RemoteRef HEAD

RemoteSha 16df57d89e2a5bdb009f24494e3d8b1d1cbf7be9

Contents

`smfsb-package` *Stochastic Modelling for Systems Biology*

Description

This package contains code and data for modelling and simulation of stochastic kinetic biochemical network models. It contains the code and data associated with the second and third editions of the book Stochastic Modelling for Systems Biology, published by Chapman & Hall/CRC Press.

Author(s)

Maintainer: Darren Wilkinson <darrenjwilkinson@btinternet.com>

References

See https://darrenjw.github.io/work/smfsb/ or https://github.com/darrenjw/smfsb for further details.

---

abcRun — *Run a set of simulations initialised with parameters sampled from a given prior distribution, and compute statistics required for an ABC analaysis*

Description

Run a set of simulations initialised with parameters sampled from a given prior distribution, and compute statistics required for an ABC analaysis. Typically used to calculate "distances" of simulated synthetic data from observed data.

Usage

```
abcRun(n, rprior, rdist)
```

Arguments

n	An integer representing the number of simulations to run.
rprior	A function without arguments generating a single parameter (vector) from prior distribution.
rdist	A function taking a parameter (vector) as argument and returning the required statistic of interest. This will typically be computed by first using the parameter to run a forward model, then computing required summary statistics, then computing a distance. See the example for details.

Value

A list with elements 'param' and 'dist'. These will be returned as matrices or vectors depending on whether the parameters and distances are scalars or vectors.

See Also

pfMLLik, StepGillespie, abcSmc, simTs, stepLVc

Examples

```
data(LVdata)
rprior <- function() { exp(c(runif(1, -3, 3),runif(1,-8,-2),runif(1,-4,2))) }
rmodel <- function(th) { simTs(c(50,100), 0, 30, 2, stepLVc, th) }
sumStats <- identity
ssd = sumStats(LVperfect)
distance <- function(s) {
    diff = s - ssd
```

```
    sqrt(sum(diff*diff))
}
rdist <- function(th) { distance(sumStats(rmodel(th))) }
out = abcRun(10000, rprior, rdist)
q=quantile(out$dist, c(0.01, 0.05, 0.1))
print(q)
accepted = out$param[out$dist < q[1],]
print(summary(accepted))
print(summary(log(accepted)))
```

---

abcSmc *Run an ABC-SMC algorithm for infering the parameters of a forward model*

Description

Run an ABC-SMC algorithm for infering the parameters of a forward model. This sequential Monte Carlo algorithm often performs better than simple rejection-ABC in practice.

Usage

```
abcSmc(N, rprior, dprior, rdist, rperturb, dperturb, factor=10,
                  steps=15, verb=FALSE)
```

Arguments

N	An integer representing the number of simulations to pass on at each stage of the SMC algorithm. Note that the TOTAL number of forward simulations required by the algorithm will be (roughly) 'N*steps*factor'.
rprior	A function without arguments generating a single parameter (vector) from prior distribution.
dprior	A function with required argument a model parameter (such as generated by 'rprior') and optional parameter 'log' returing the (log) density of the parameter under the prior distribution.
rdist	A function taking a parameter (vector) as argument and returning a scalar "distance" representing a measure of how good the chosen parameter is. This will typically be computed by first using the parameter to run a forward model, then computing required summary statistics, then computing a distance. See the example for details.
rperturb	A function which takes a parameter as its argument and returns a perturbed parameter from an appropriate kernel.
dperturb	A function which takes a pair of parameters as its first two arguments (new first and old second), and has an optional argument 'log' for whether to return the log of the density associated with this perturbation kernel.

factor	At each step of the algorithm, 'N*factor' proposals are generated and the best 'N' of these are weighted and passed on to the next stage. Note that the effective sample size of the parameters passed on to the next step may be (much) smaller than 'N', since some of the particles may be assigned small (or zero) weight.
steps	The number of steps of the ABC-SMC algorithm. Typically, somewhere between 5 and 100 steps seems to be used in practice.
verb	Boolean indicating whether some progress should be printed to the console (the number of steps remaining).

Value

A matrix (or vector) with rows (or elements) representing samples from the approximate posterior distribution.

See Also

pfMLLik, StepGillespie, abcRun, simTs, stepLVc

Examples

```
data(LVdata)
rprior <- function() { c(runif(1, -3, 3), runif(1, -8, -2), runif(1, -4, 2)) }
dprior <- function(x, ...) { dunif(x[1], -3, 3, ...) +
                dunif(x[2], -8, -2, ...) + dunif(x[3], -4, 2, ...) }
rmodel <- function(th) { simTs(c(50,100), 0, 30, 2, stepLVc, exp(th)) }
rperturb <- function(th){th + rnorm(3, 0, 0.5)}
dperturb <- function(thNew, thOld, ...){sum(dnorm(thNew, thOld, 0.5, ...))}
sumStats <- identity
ssd = sumStats(LVperfect)
distance <- function(s) {
    diff = s - ssd
    sqrt(sum(diff*diff))
}
rdist <- function(th) { distance(sumStats(rmodel(th))) }
out = abcSmc(5000, rprior, dprior, rdist, rperturb,
             dperturb, verb=TRUE, steps=6, factor=5)
print(summary(out))
```

---

as.timedData *Convert a time series object to a timed data matrix*

Description

This function converts a time series object to a timed data matrix, similar to that produced by simTimes. The main purpose is for passing data to the function pfMLLik, which expects data encoded in this format.

Usage

```
as.timedData(timeseries)
```

Arguments

timeseries	An R timeseries object, such as produced by the functions ts or simTs.

Value

An R matrix object with row names corresponding to observation times, similar to that produced by simTimes.

See Also

simTs, ts, simTimes, pfMLLik

Examples

```
truth=simTs(c(x1=50,x2=100),0,20,2,stepLVc)
simData=truth+rnorm(prod(dim(truth)),0,5)
timedData=as.timedData(simData)
print(timedData)
```

---

discretise *Discretise output from a discrete event simulation algorithm*

Description

This function discretise output from a discrete event simulation algorithm such as gillespie onto a regular time grid, and returns the results as an R ts object.

Usage

```
discretise(out, dt=1, start=0)
```

Arguments

out	A list containing discrete event simulation output in the form of that produced by gillespie.
dt	The time step required for the output of the discretisation process. Defaults to one time unit.
start	The start time for the output. Defaults to zero.

Value

An R ts object containing the discretised output.

See Also

simpleEuler, rdiff, gillespie, gillespied, ts

Examples

```
# load LV model
data(spnModels)
# simulate a realisation of the process and plot it
out = gillespie(LV,10000)
op=par(mfrow=c(2,2))
plot(stepfun(out$t,out$x[,1]),pch="")
plot(stepfun(out$t,out$x[,2]),pch="")
plot(out$x,type="l")

# use the "discretise" function to map it to an R "ts" object
plot(discretise(out,dt=0.01),plot.type="single",lty=c(1,2))
par(op)
```

`gillespie` *Simulate a sample path from a stochastic kinetic model described by a stochastic Petri net*

Description

This function simulates a single realisation from a discrete stochastic kinetic model described by a stochastic Petri net (SPN).

Usage

```
gillespie(N, n, ...)
```

Arguments

`N`	An R list with named components representing a stochastic Petri net (SPN). Should contain `N$M`, a vector representing the initial marking of the net, `N$Pre`, a matrix representing the LHS stoichiometries, `N$Post`, a matrix representing the RHS stoichiometries, and `N$h`, a function representing the rates of the reaction processes. `N$h` should have first argument `x`, a vector representing the current state of the system, and second argument `t`, a scalar representing the current simulation time (in the typical time-homogeneous case, `N$h` will ignore this argument). `N$h` may posses additional arguments, representing reaction rates, for example.
`n`	An integer representing the number of events to simulate, excluding the initial state, `N$M`.
`...`	Additional arguments (such as reactions rates) will be passed into the function `N$h`.

Value

A list with first component t, a vector of length n containing event times and second component x, a matrix with n+1 rows containing the state of the system. The ith row of x contains the state of the system prior to the ith event.

See Also

simpleEuler, rdiff, discretise, gillespied, StepGillespie

Examples

```
# load the LV model
data(spnModels)
# simulate a realisation of the process and plot it
out = gillespie(LV,10000)
op = par(mfrow=c(2,2))
plot(stepfun(out$t,out$x[,1]),pch="")
plot(stepfun(out$t,out$x[,2]),pch="")
plot(out$x,type="l")

# use the "discretise" function to map it to an R "ts" object
plot(discretise(out,dt=0.01),plot.type="single",lty=c(1,2))
par(op)
```

---

gillespied *Simulate a sample path from a stochastic kinetic model described by a stochastic Petri net*

Description

This function simulates a single realisation from a discrete stochastic kinetic model described by a stochastic Petri net and discretises the output onto a regular time grid.

Usage

```
gillespied(N, T=100, dt=1, ...)
```

Arguments

N	An R list with named components representing a stochastic Petri net (SPN). Should contain N$M, a vector representing the initial marking of the net, N$Pre, a matrix representing the LHS stoichiometries, N$Post, a matrix representing the RHS stoichiometries, and N$h, a function representing the rates of the reaction processes. N$h should have first argument x, a vector representing the current state of the system, and second argument t, a scalar representing the current simulation time (in the typical time-homogeneous case, N$h will ignore this argument). N$h may posses additional arguments, representing reaction rates, for example.

T	The required length of simulation time. Defaults to 100 time units.
dt	The grid size for the output. Note that this parameter simply determines the volume of output. It has no bearing on the correctness of the simulation algorithm. Defaults to one time unit.
...	Additional arguments will be passed into the function N$h.

Value

An R ts object containing the simulated realisation of the process.

See Also

simpleEuler, rdiff, discretise, gillespie, StepGillespie

Examples

```
# load LV model
data(spnModels)
# simulate and plot a realisation
plot(gillespied(LV,T=100,dt=0.01))
```

---

imdeath *Simulate a sample path from the homogeneous immigration-death process*

Description

This function simulates a single realisation from a time-homogeneous immigration-death process.

Usage

```
imdeath(n=20,x0=0,lambda=1,mu=0.1)
```

Arguments

n	The number of states to be sampled from the process, not including the initial state, x0
x0	The initial state of the process, which defaults to zero.
lambda	The rate at which new individual immigrate into the population. Defaults to 1.
mu	The rate at which individuals within the population die, independently of all other individuals. Defaults to 0.1.

Value

An R stepfun object containing the sampled path of the process.

See Also

rcfmc, rdiff, stepfun, gillespie

Examples

```
plot(imdeath(50))
```

LVdata *Example simulated time courses from a stochastic Lotka–Volterra model*

Description

Collection of simulated time courses from a stochastic Lotka–Volterra model. LVperfect is direct output from a Gillespie simulation. LVprey is the prey component. LVnoise10 has Gaussian noise with standard deviation 10 added. LVnoise30 has Gaussian noise with standard deviation 30 added. LVpreyNoise10 is the prey component with 10 SD noise added. LVnoise3010 has Gaussian noise added. The noise added to the prey component has standard deviation 30 and the noise added to the predator component has standard deviation 10. LVnoise10scale10 has Gaussian noise with standard deviation 10 added, and is then rescaled by a factor of 10 to mimic a scenario of an uncalibrated measurement scale. LVirregular is direct output from a Gillespie simulator, but on an irregular time grid. LVirregularNoise10 is output on an irregular time grid with Gaussian noise of standard deviation 10 added.

Usage

```
data(LVdata)
```

Format

All datasets beginning LVirregular are R matrices such as output by simTimes, and the rest are R ts objects such as output by simTs.

---

mcmcSummary *Summarise and plot tabular MCMC output*

Description

This function summarises and plots tabular MCMC output such as that generated by the function normgibbs.

Usage

```
mcmcSummary(mat, rows = 4, lag.max=100, bins=30, show = TRUE, plot = TRUE, truth = NULL)
```

Arguments

mat	Matrix of MCMC output, where the columns represent variables and the rows represent iterations.
rows	Number of variables to plot per page on the graphics device.
lag.max	Maximum lag for the ACF plots.
bins	Approximate number of bins to use for the histograms.
show	If TRUE, will display numerical summaries on the R console.
plot	If TRUE, will plot graphical summaries on the default graphics device.
truth	Optional vector of "true values", one for each variable, for the case where an algorithm is being tested on synthetic data for known parameters. The plots will be annotated with a red line indicating the true value.

Value

An R summary object.

See Also

normgibbs, summary, acf

Examples

```
out=normgibbs(N=1000,n=15,a=3,b=11,cc=10,d=1/100,xbar=25,ssquared=20)
names(out)=c("mu","tau")
mcmcSummary(out,rows=2,bins=10,truth=c(25,1/20))
```

---

metrop *Run a simple Metropolis sampler with standard normal target and uniform innovations*

Description

This function runs a simple Metropolis sampler with standard normal target distribution and uniform innovations.

Usage

```
metrop(n, alpha)
```

Arguments

n	The number of iterations of the Metropolis sampler.
alpha	The tuning parameter of the sampler. The innovations of the sampler are of the form U(-alpha,alpha).

Value

An R vector containing the output of the sampler.

See Also

normgibbs

Examples

```
normvec=metrop(1000,1)
op=par(mfrow=c(2,1))
plot(ts(normvec))
hist(normvec,20)
par(op)
```

`metropolisHastings` *Run a Metropolis-Hastings MCMC algorithm for the parameters of a Bayesian posterior distribution*

Description

Run a Metropolis-Hastings MCMC algorithm for the parameters of a Bayesian posterior distribution. Note that the algorithm carries over the old likelihood from the previous iteration, making it suitable for problems with expensive likelihoods, and also for "exact approximate" pseudo-marginal or particle marginal MH algorithms.

Usage

```
metropolisHastings(init, logLik, rprop, dprop=function(new, old, ...){1},
                   dprior=function(x, ...){1}, iters=10000, thin=10,
                   verb=TRUE, debug=FALSE)
```

Arguments

`init`	An parameter vector with which to initialise the MCMC algorithm.
`logLik`	A function which takes a parameter (such as `init`) as its only required argument and returns the log-likelihood of the data. Note that it is fine for this to return the log of an unbiased estimate of the likelihood, in which case the algorithm will be an "exact approximate" pseudo-marginal MH algorithm.
`rprop`	A function which takes a parameter as its only required argument and returns a single sample from a proposal distribution.
`dprop`	A function which takes a new and old parameter as its first two required arguments and returns the (log) density of the new value conditional on the old. It should accept an optional parameter `log`, and must return the log-density when `log` is `TRUE`. Defaults to a flat function which causes this term to drop out of the acceptance probability. It is fine to use the default for _any_ _symmetric_ proposal, since the term will also drop out for any symmetric proposal.

dprior	A function which take a parameter as its only required argument and returns the (log) density of the parameter value under the prior. It should accept an optional parameter log, and must return the log-density when log is TRUE. Defaults to a flat function which causes this term to drop out of the acceptance probability. People often use a flat prior when they are trying to be "uninformative" or "objective", but this is slightly naive. In particular, what is "flat" is clearly dependent on the parametrisation of the model.
iters	The number of MCMC iterations required (_after_ thinning).
thin	The required thinning factor. eg. only store every thin iterations.
verb	Boolean indicating whether some progress information should be printed to the console. Defaults to TRUE.
debug	Boolean indicating whether debugging information is required. Prints information about each iteration to console, to, eg., debug a crashing sampler.

Value

A matrix with rows representing samples from the posterior distribution.

See Also

pfMLLik, StepGillespie, abcRun, simTs, stepLVc, metrop

Examples

```
## First simulate some synthetic data
data = rnorm(250,5,2)
## Now use MH to recover the parameters
llik = function(x) { sum(dnorm(data,x[1],x[2],log=TRUE)) }
prop = function(x) { rnorm(2,x,0.1) }
prior = function(x, log=TRUE) {
    l = dnorm(x[1],0,100,log=TRUE) + dgamma(x[2],1,0.0001,log=TRUE)
    if (log) l else exp(l)
}
out = metropolisHastings(c(mu=1,sig=1), llik, prop,
                         dprior=prior, verb=FALSE)
out = out[1000:10000,]
mcmcSummary(out, truth=c(5,2), rows=2, plot=FALSE)
```

---

mytable *Simple example data frame*

Description

Trivial example of a very small data frame. Used as part of the R tutorial.

Usage

```
data(mytable)
```

Format

A very small example data frame.

`normgibbs` *A simple Gibbs sampler for Bayesian inference for the mean and precision of a normal random sample*

Description

This function runs a simple Gibbs sampler for the Bayesian posterior distribution of the mean and precision given a normal random sample.

Usage

```
normgibbs(N, n, a, b, cc, d, xbar, ssquared)
```

Arguments

`N`	The number of iterations of the Gibbs sampler.
`n`	The sample size of the normal random sample.
`a`	The shape parameter of the gamma prior on the sample precision.
`b`	The scale parameter of the gamma prior on the sample precision.
`cc`	The mean of the normal prior on the sample mean.
`d`	The precision of the normal prior on the sample mean.
`xbar`	The sample mean of the data. eg. `mean(x)` for a vector `x`.
`ssquared`	The sample variance of the data. eg. `var(x)` for a vector `x`.

Value

An R matrix object containing the samples of the Gibbs sampler.

See Also

`rcfmc`, `metrop`, `mcmcSummary`

Examples

```
postmat=normgibbs(N=1100,n=15,a=3,b=11,cc=10,d=1/100,xbar=25,ssquared=20)
postmat=postmat[101:1100,]
op=par(mfrow=c(3,3))
plot(postmat)
plot(postmat,type="l")
plot.new()
plot(ts(postmat[,1]))
plot(ts(postmat[,2]))
plot(ts(sqrt(1/postmat[,2])))
```

```
hist(postmat[,1],30)
hist(postmat[,2],30)
hist(sqrt(1/postmat[,2]),30)
par(op)
```

`pfMLLik` *Create a function for computing the log of an unbiased estimate of marginal likelihood of a time course data set*

Description

Create a function for computing the log of an unbiased estimate of marginal likelihood of a time course data set using a simple bootstrap particle filter. This version uses the "log-sum-exp trick" for avoiding numerical underflow of weights. See `pfMLLik1` for a version which doesn't.

Usage

```
pfMLLik(n,simx0,t0,stepFun,dataLik,data)
```

Arguments

`n`	An integer representing the number of particles to use in the particle filter.
`simx0`	A function with interface `simx0(n,t0,...)`, where `n` is the number of rows of a matrix and `t0` is a time at which to simulate from an initial distribution for the state of the particle filter. The return value should be a matrix whose rows are random samples from this distribution. The function therefore represents a prior distribution on the initial state of the Markov process.
`t0`	The time corresponding to the starting point of the Markov process. Can be no bigger than the smallest observation time.
`stepFun`	A function for advancing the state of the Markov process, such as returned by `StepGillespie`.
`dataLik`	A function with interface `dataLik(x,t,y,log=TRUE,...)`, where `x` and `t` represent the true state and time of the process, and `y` is the observed data. The return value should be the (log of the) likelihood of the observation. The function therefore represents the observation model.
`data`	A timed data matrix representing the observations, such as produced by `simTimes` or `as.timedData`.

Value

An R function with interface `(...)` which evaluates to the log of the particle filter's unbiased estimate of the marginal likelihood of the data.

See Also

`pfMLLik1`, `StepGillespie`, `as.timedData`, `simTimes`, `stepLVc`

Examples

```
noiseSD=5
# first simulate some data
truth=simTs(c(x1=50,x2=100),0,20,2,stepLVc)
data=truth+rnorm(prod(dim(truth)),0,noiseSD)
data=as.timedData(data)
# measurement error model
dataLik <- function(x,t,y,log=TRUE,...)
{
    ll=sum(dnorm(y,x,noiseSD,log=TRUE))
    if (log)
        return(ll)
    else
        return(exp(ll))
}
# now define a sampler for the prior on the initial state
simx0 <- function(N,t0,...)
{
    mat=cbind(rpois(N,50),rpois(N,100))
    colnames(mat)=c("x1","x2")
    mat
}
mLLik=pfMLLik(1000,simx0,0,stepLVc,dataLik,data)
print(mLLik())
print(mLLik(th=c(th1 = 1, th2 = 0.005, th3 = 0.6)))
print(mLLik(th=c(th1 = 1, th2 = 0.005, th3 = 0.5)))
```

`pfMLLik1` *Create a function for computing the log of an unbiased estimate of marginal likelihood of a time course data set*

Description

Create a function for computing the log of an unbiased estimate of marginal likelihood of a time course data set using a simple bootstrap particle filter. This version does not use the "log-sum-exp trick" for avoiding numerical underflow. See `pfMLLik` for a version which does.

Usage

```
pfMLLik1(n,simx0,t0,stepFun,dataLik,data)
```

Arguments

`n`	An integer representing the number of particles to use in the particle filter.
`simx0`	A function with interface `simx0(n,t0,...)`, where `n` is the number of rows of a matrix and `t0` is a time at which to simulate from an initial distribution for the state of the particle filter. The return value should be a matrix whose rows are random samples from this distribution. The function therefore represents a prior distribution on the initial state of the Markov process.

t0	The time corresponding to the starting point of the Markov process. Can be no bigger than the smallest observation time.
stepFun	A function for advancing the state of the Markov process, such as returned by StepGillespie.
dataLik	A function with interface dataLik(x,t,y,log=TRUE,...), where x and t represent the true state and time of the process, and y is the observed data. The return value should be the (log of the) likelihood of the observation. The function therefore represents the observation model.
data	A timed data matrix representing the observations, such as produced by simTimes or as.timedData.

Value

An R function with interface (...) which evaluates to the log of the particle filter's unbiased estimate of the marginal likelihood of the data.

See Also

pfMLLik, StepGillespie, as.timedData, simTimes, stepLVc

Examples

```
noiseSD=5
# first simulate some data
truth=simTs(c(x1=50,x2=100),0,20,2,stepLVc)
data=truth+rnorm(prod(dim(truth)),0,noiseSD)
data=as.timedData(data)
# measurement error model
dataLik <- function(x,t,y,log=TRUE,...)
{
    ll=sum(dnorm(y,x,noiseSD,log=TRUE))
    if (log)
        return(ll)
    else
        return(exp(ll))
}
# now define a sampler for the prior on the initial state
simx0 <- function(N,t0,...)
{
    mat=cbind(rpois(N,50),rpois(N,100))
    colnames(mat)=c("x1","x2")
    mat
}
mLLik=pfMLLik1(1000,simx0,0,stepLVc,dataLik,data)
print(mLLik())
print(mLLik(th=c(th1 = 1, th2 = 0.005, th3 = 0.6)))
print(mLLik(th=c(th1 = 1, th2 = 0.005, th3 = 0.5)))
```

rcfmc — *Simulate a continuous time finite state space Markov chain*

Description

This function simulates a single realisation from a continuous time Markov chain having a finite state space based on a given transition rate matrix.

Usage

```
rcfmc(n,Q,pi0)
```

Arguments

`n`	The number of states to be sampled from the Markov chain, including the initial state, which will be sampled using `pi0`.
`Q`	The transition rate matrix of the Markov chain, where each off-diagonal element `Q[i,j]` represents the rate of transition from state `i` to state `j`. This matrix is assumed to be square, having rows summing to zero.
`pi0`	A vector representing the probability distribution of the initial state of the Markov chain. If this vector is of length `r`, then the transition matrix `P` is assumed to be `r x r`. The elements of this vector are assumed to be non-negative and sum to one, though in fact, they will be normalised by the sampling procedure.

Value

An R `stepfun` object containing the sampled path of the process.

See Also

`rfmc`, `stepfun`

Examples

```
plot(rcfmc(20,matrix(c(-0.5,0.5,1,-1),ncol=2,byrow=TRUE),c(1,0)))
```

rdiff — *Simulate a sample path from a univariate diffusion process*

Description

This function simulates a single realisation from a time-homogeneous univariate diffusion process.

Usage

```
rdiff(afun, bfun, x0 = 0, t = 50, dt = 0.01, ...)
```

Arguments

afun	A scalar-valued function representing the infinitesimal mean (drift) of the diffusion process. The first argument of afun is the current state of the process.
bfun	A scalar-valued function representing the infinitesimal standard deviation of the process. The first argument of bfun is the current state of the process.
x0	The initial state of the diffusion process.
t	The length of the time interval over which the diffusion process is to be simulated. Defaults to 50 time units.
dt	The step size to be used both for the time step of the Euler integration method and the recording interval for the output. It would probably be better to have separate parameters for these two things (see StepSDE and simTs). Defaults to 0.01 time units.
...	Additional arguments will be passed into afun and bfun.

Value

An R ts object containing the sampled path of the process.

See Also

rcfmc, ts, StepSDE, simTs

Examples

```
# simulate a diffusion approximation to an immigration-death process
# infinitesimal mean
afun<-function(x,lambda,mu)
{
        lambda-mu*x
}
# infinitesimal standard deviation
bfun<-function(x,lambda,mu)
{
        sqrt(lambda+mu*x)
}
# plot a sample path
plot(rdiff(afun,bfun,lambda=1,mu=0.1,t=30))
```

---

rfmc *Simulate a finite state space Markov chain*

Description

This function simulates a single realisation from a discrete time Markov chain having a finite state space based on a given transition matrix.

Usage

```
rfmc(n,P,pi0)
```

Arguments

n	The number of states to be sampled from the Markov chain, including the initial state, which will be sampled using pi0.
P	The transition matrix of the Markov chain. This is assumed to be a stochastic matrix, having non-negative elements and rows summing to one, though in fact, the rows will in any case be normalised by the sampling procedure.
pi0	A vector representing the probability distribution of the initial state of the Markov chain. If this vector is of length r, then the transition matrix P is assumed to be r x r. The elements of this vector are assumed to be non-negative and sum to one, though in fact, they will be normalised by the sampling procedure.

Value

An R ts object containing the sampled values from the Markov chain.

See Also

rcfmc, ts

Examples

```
# example for sampling a finite Markov chain
P = matrix(c(0.9,0.1,0.2,0.8),ncol=2,byrow=TRUE)
pi0 = c(0.5,0.5)
samplepath = rfmc(200,P,pi0)
plot(samplepath)
summary(samplepath)
table(samplepath)
table(samplepath)/length(samplepath) # empirical distribution
# now compute the exact stationary distribution...
e = eigen(t(P))$vectors[,1]
e/sum(e)
```

---

simpleEuler *Simulate a sample path from an ODE model*

Description

This function integrates an Ordinary Differential Equation (ODE) model using a simple first order Euler method. The function is pedagogic and not intended for serious use. See the deSolve package for better, more robust ODE solvers.

Usage

```
simpleEuler(t=50, dt=0.001, fun, ic, ...)
```

Arguments

t	The length of the time interval over which the ODE model is to be integrated. Defaults to 50 time units.
dt	The step size to be used both for the time step of the Euler integration method and the recording interval for the output. It would probably be better to have separate parameters for these two things (see StepEuler and simTs). Defaults to 0.01 time units.
fun	A vector-valued function representing the right hand side of the ODE model. The first argument is a vector representing the current state of the model, x. The second argument of fun is the current simulation time, t. In the case of a homogeneous ODE model, this argument will be unused within the function. The function may have additional arguments, representing model parameters. The output of fun should be a vector of the same dimension as x.
ic	The initial conditions for the ODE model. This should be a vector of the same dimensions as the output from fun, and the second argument of fun.
...	Additional arguments will be passed into fun.

Value

An R ts object containing the sampled path of the model.

See Also

rdiff, ts, StepEuler, simTs

Examples

```
# simple Lotka-Volterra example
lv <- function(x,t,k=c(k1=1,k2=0.1,k3=0.1))
{
        with(as.list(c(x,k)),{
                c( k1*x1 - k2*x1*x2 ,
                      k2*x1*x2 - k3*x2 )
        })
}
plot(simpleEuler(t=100,fun=lv,ic=c(x1=4,x2=10)),plot.type="single",lty=1:2)

# now an example which instead uses deSolve...
require(deSolve)
times = seq(0,50,by=0.01)
k = c(k1=1,k2=0.1,k3=0.1)
lvlist = function(t,x,k)
        list(lv(x,t,k))
plot(ode(y=c(x1=4,x2=10),times=times,func=lvlist,parms=k))
```

simSample — *Simulate a many realisations of a model at a given fixed time in the future given an initial time and state, using a function (closure) for advancing the state of the model*

Description

This function simulates many realisations of a model at a given fixed time in the future given an initial time and state, using a function (closure) for advancing the state of the model , such as created by StepGillespie or StepSDE.

Usage

```
simSample(n=100,x0,t0=0,deltat,stepFun,...)
```

Arguments

n	The number of samples required. Defaults to 100.
x0	The initial state of the process at time t0.
t0	The initial time to be associated with the initial state x0. Defaults to 0.
deltat	The amount of time in the future of t0 at which samples of the system state are required.
stepFun	A function (closure) for advancing the state of the process, such as produced by StepGillespie or StepEulerSPN.
...	Additional arguments will be passed to stepFun.

Value

An R matrix whose rows represent the simulated states of the process at time t0+deltat.

See Also

StepSDE, StepGillespie, simTimes, simTs

Examples

```
out3 = simSample(100,c(x1=50,x2=100),0,20,stepLVc)
hist(out3[,"x2"])
```

simTimes *Simulate a model at a specified set of times, using a function (closure) for advancing the state of the model*

Description

This function simulates a single realisation from a Markovian model and records the state at a specified set of times using a function (closure) for advancing the state of the model, such as created by StepGillespie or StepEulerSPN.

Usage

```
simTimes(x0,t0=0,times,stepFun,...)
```

Arguments

x0	The initial state of the process at time t0.
t0	The initial time to be associated with the initial state x0. Defaults to 0.
times	A vector of times at which the state of the process is required. It is assumed that the times are in increasing order, and that the first time is at least as big as t0.
stepFun	A function (closure) for advancing the state of the process, such as produced by StepGillespie or StepEulerSPN.
...	Additional arguments will be passed to stepFun.

Value

An R matrix where each row represents the state of the process at one of the required times. The row names contain the sampled times.

See Also

StepEulerSPN, StepGillespie, simTs, simSample, as.timedData, pfMLLik

Examples

```
# load the LV model
data(spnModels)
# create a stepping function
stepLV = StepGillespie(LV)
# simulate a realisation using simTimes
times = seq(0,100,by=0.1)
plot(ts(simTimes(c(x1=50,x2=100),0,times,stepLV),start=0,deltat=0.1),plot.type="single",lty=1:2)
# simulate a realisation at irregular times
times = c(0,10,20,50,100)
out2 = simTimes(c(x1=50,x2=100),0,times,stepLV)
print(out2)
```

simTs *Simulate a model on a regular grid of times, using a function (closure) for advancing the state of the model*

Description

This function simulates single realisation of a model on a regular grid of times using a function (closure) for advancing the state of the model, such as created by StepGillespie or StepEulerSPN.

Usage

```
simTs(x0,t0=0,tt=100,dt=0.1,stepFun,...)
```

Arguments

x0	The initial state of the process at time t0.
t0	The initial time to be associated with the initial state x0. Defaults to 0.
tt	The terminal time of the simulation.
dt	The time step of the output. Note that this time step relates only to the recorded output, and has no bearing on the accuracy of the simulation process.
stepFun	A function (closure) for advancing the state of the process, such as produced by StepGillespie or StepEulerSPN.
...	Additional arguments will be passed to stepFun.

Value

An R ts object representing the simulated process.

See Also

StepEulerSPN, StepGillespie, StepSDE, simTimes, simSample, as.timedData

Examples

```
# load the LV model
data(spnModels)
# create a stepping function
stepLV = StepGillespie(LV)
# simulate a realisation of the process and plot it
out = simTs(c(x1=50,x2=100),0,100,0.1,stepLV)
plot(out)
plot(out,plot.type="single",lty=1:2)
```

simTs1D *Simulate a model on a regular grid of times, using a function (closure) for advancing the state of the model*

Description

This function simulates single realisation of a model on a 1D regular spatial grid and regular grid of times using a function (closure) for advancing the state of the model, such as created by StepGillespie1D.

Usage

```
simTs1D(x0,t0=0,tt=100,dt=0.1,stepFun,verb=FALSE,...)
```

Arguments

x0	The initial state of the process at time t0, a matrix with rows corresponding to reacting species and columns corresponding to spatial location.
t0	The initial time to be associated with the initial state x0. Defaults to 0.
tt	The terminal time of the simulation.
dt	The time step of the output. Note that this time step relates only to the recorded output, and has no bearing on the accuracy of the simulation process.
stepFun	A function (closure) for advancing the state of the process, such as produced by StepGillespie1D.
verb	Output progress to the console (this function can be very slow).
...	Additional arguments will be passed to stepFun.

Value

An R 3d array representing the simulated process. The dimensions are species, space, and time.

See Also

StepGillespie1D, simTs

Examples

```
data(spnModels)
N=20; T=30
x0=matrix(0,nrow=2,ncol=N)
rownames(x0)=c("x1","x2")
x0[,round(N/2)]=LV$M
stepLV1D = StepGillespie1D(LV,c(0.6,0.6))
xx = simTs1D(x0,0,T,0.2,stepLV1D,verb=TRUE)
op=par(mfrow=c(1,2))
image(xx[1,,],main="Prey",xlab="Space",ylab="Time")
```

```
image(xx[2,,],main="Predator",xlab="Space",ylab="Time")
par(op)
```

`simTs2D` *Simulate a model on a regular grid of times, using a function (closure) for advancing the state of the model*

Description

This function simulates single realisation of a model on a 2D regular spatial grid and regular grid of times using a function (closure) for advancing the state of the model, such as created by `StepGillespie2D`.

Usage

```
simTs2D(x0,t0=0,tt=100,dt=0.1,stepFun,verb=FALSE,...)
```

Arguments

`x0`	The initial state of the process at time `t0`, a 3d array with dimensions corresponding to reacting species and two spatial dimensions.
`t0`	The initial time to be associated with the initial state `x0`. Defaults to 0.
`tt`	The terminal time of the simulation.
`dt`	The time step of the output. Note that this time step relates only to the recorded output, and has no bearing on the accuracy of the simulation process.
`stepFun`	A function (closure) for advancing the state of the process, such as produced by `StepGillespie2D`.
`verb`	Output progress to the console and graphics window (this function can be very slow).
`...`	Additional arguments will be passed to `stepFun`.

Value

An R 4d array representing the simulated process. The dimensions are species, 2 space, and time.

See Also

`StepGillespie2D`, `simTs1D`

Examples

```
data(spnModels)
m=20; n=30; T=15
x0=array(0,c(2,m,n))
dimnames(x0)[[1]]=c("x1","x2")
x0[,round(m/2),round(n/2)]=LV$M
stepLV2D = StepGillespie2D(LV,c(0.6,0.6))
xx = simTs2D(x0,0,T,0.2,stepLV2D,verb=TRUE)
N = dim(xx)[4]
op=par(mfrow=c(1,2))
image(xx[1,,,N],main="Prey",xlab="Space",ylab="Time")
image(xx[2,,,N],main="Predator",xlab="Space",ylab="Time")
par(op)
```

`spnModels` *Example SPN models*

Description

Collection of example stochastic Petri net (SPN) models. Includes `LV`, a Lotka–Volterra model, `ID`, an immigration–death process, `BD`, a birth–death process, `SIR`, a simple SIR model, `SEIR`, an SEIR epidemic model, `Dimer`, a simple dimerisation kinetics model, and `MM`, a Michaelis–Menten enzyme kinetic model.

Usage

```
data(spnModels)
```

Format

Each model is a list, with components `Pre`, `Post`, and `h`. Some models also include an initial state, `M`. See `gillespie` and `StepGillespie` for further details, and examples of use.

`StepCLE` *Create a function for advancing the state of an SPN by using a simple Euler-Maruyama integration method for the approximating CLE*

Description

This function creates a function for advancing the state of an SPN model using a simple Euler-Maruyama integration method for the approximating chemical Langevin equation (CLE). The resulting function (closure) can be used in conjunction with other functions (such as `simTs`) for simulating realisations of SPN models.

Usage

```
StepCLE(N,dt=0.01)
```

Arguments

N	An R list with named components representing a stochastic Petri net. Should contain N$Pre, a matrix representing the LHS stoichiometries, N$Post, a matrix representing the RHS stoichiometries, and N$h, a function representing the rates of the reaction processes. N$h should have first argument x, a vector representing the current state of the system, and second argument t, a scalar representing the current simulation time (in the typical time-homogeneous case, N$h will ignore this argument). N$h may posses additional arguments, representing reaction rates, for example. N does not need to contain an initial marking, N$M. N$M will be ignored by most functions which use the resulting function closure.
dt	Time step to be used by the Euler-Maruyama integration method. Defaults to 0.01.

Value

An R function which can be used to advance the state of the SPN model N by using an Euler-Maruyama method on the approximating CLE with step size dt. The function closure has interface function(x0,t0,deltat,...), where x0 and t0 represent the initial state and time, and deltat represents the amount of time by which the process should be advanced. The function closure returns a vector representing the simulated state of the system at the new time.

See Also

StepGillespie, StepEulerSPN, StepSDE, simTs, simSample

Examples

```
# load the LV model
data(spnModels)
# create a stepping function
stepLV = StepCLE(LV)
# step the function
print(stepLV(c(x1=50,x2=100),0,1))
# integrate the process and plot it
out = simTs(c(x1=50,x2=100),0,20,0.1,stepLV)
plot(out)
plot(out,plot.type="single",lty=1:2)
```

StepCLE1D — *Create a function for advancing the state of an SPN by using a simple Euler-Maruyama discretisation of the CLE on a 1D regular grid*

Description

This function creates a function for advancing the state of an SPN model using a simple Euler-Maruyama discretisation of the CLE on a 1D regular grid. The resulting function (closure) can be used in conjunction with other functions (such as simTs1D) for simulating realisations of SPN models in space and time.

Usage

```
StepCLE1D(N,d,dt=0.01)
```

Arguments

N	An R list with named components representing a stochastic Petri net (SPN). Should contain N$Pre, a matrix representing the LHS stoichiometries, N$Post, a matrix representing the RHS stoichiometries, and N$h, a function representing the rates of the reaction processes. N$h should have first argument x, a vector representing the current state of the system, and second argument t, a scalar representing the current simulation time (in the typical time-homogeneous case, N$h will ignore this argument). N$h may possess additional arguments, representing reaction rates, for example. N does not need to contain an initial marking, N$M. N$M will be ignored by most functions which use the resulting function closure.
d	A vector of diffusion coefficients - one coefficient for each reacting species, in order. The coefficient is the reaction rate for a reaction for a molecule moving into an adjacent compartment. The hazard for a given molecule leaving the compartment is therefore twice this value (as it can leave to the left or the right).
dt	Time step for the Euler-Maruyama discretisation.

Value

An R function which can be used to advance the state of the SPN model N by using a simple Euler-Maruyama algorithm. The function closure has interface function(x0,t0,deltat,...), where x0 is a matrix with rows corresponding to species and columns corresponding to voxels, representing the initial condition, t0 represent the initial state and time, and deltat represents the amount of time by which the process should be advanced. The function closure returns a matrix representing the simulated state of the system at the new time.

See Also

StepGillespie1D,StepCLE, simTs1D, StepCLE2D

Examples

```
N=200
T=40
data(spnModels)
x0=matrix(0,nrow=2,ncol=N)
rownames(x0)=c("x1","x2")
x0[,round(N/2)]=LV$M
stepLV1D = StepCLE1D(LV,c(0.6,0.6),dt=0.05)
xx = simTs1D(x0,0,T,0.2,stepLV1D)
op=par(mfrow=c(1,2))
image(xx[1,,],main="Prey",xlab="Space",ylab="Time")
image(xx[2,,],main="Predator",xlab="Space",ylab="Time")
par(op)
```

StepCLE2D *Create a function for advancing the state of an SPN by using a simple Euler-Maruyama discretisation of the CLE on a 2D regular grid*

Description

This function creates a function for advancing the state of an SPN model using a simple Euler-Maruyama discretisation of the CLE on a 2D regular grid. The resulting function (closure) can be used in conjunction with other functions (such as `simTs2D`) for simulating realisations of SPN models in space and time.

Usage

```
StepCLE2D(N,d,dt=0.01)
```

Arguments

`N`	An R list with named components representing a stochastic Petri net (SPN). Should contain `N$Pre`, a matrix representing the LHS stoichiometries, `N$Post`, a matrix representing the RHS stoichiometries, and `N$h`, a function representing the rates of the reaction processes. `N$h` should have first argument `x`, a vector representing the current state of the system, and second argument `t`, a scalar representing the current simulation time (in the typical time-homogeneous case, `N$h` will ignore this argument). `N$h` may possess additional arguments, representing reaction rates, for example. `N` does not need to contain an initial marking, `N$M`. `N$M` will be ignored by most functions which use the resulting function closure.
`d`	A vector of diffusion coefficients - one coefficient for each reacting species, in order. The coefficient is the reaction rate for a reaction for a molecule moving into an adjacent compartment. The hazard for a given molecule leaving the compartment is therefore four times this value (as it can leave in one of 4 directions).
`dt`	Time step for the Euler-Maruyama discretisation.

Value

An R function which can be used to advance the state of the SPN model `N` by using a simple Euler-Maruyama algorithm. The function closure has interface `function(x0,t0,deltat,...)`, where `x0` is a 3D array with rows corresponding to species and columns corresponding to voxels, representing the initial condition (with dimensions species, x, and y), `t0` represent the initial state and time, and `deltat` represents the amount of time by which the process should be advanced. The function closure returns a matrix representing the simulated state of the system at the new time.

See Also

`StepGillespie2D`,`StepCLE`, `simTs1D`, `StepCLE1D`

Examples

```
m=150
n=100
T=15
data(spnModels)
x0=array(0,c(2,m,n))
dimnames(x0)[[1]]=c("x1","x2")
x0[,round(m/2),round(n/2)]=LV$M
stepLV2D = StepCLE2D(LV,c(0.6,0.6),dt=0.05)
xx = simTs2D(x0,0,T,0.5,stepLV2D,verb=TRUE)
N = dim(xx)[4]
op=par(mfrow=c(1,2))
image(xx[1,,,N],main="Prey",xlab="Space",ylab="Time")
image(xx[2,,,N],main="Predator",xlab="Space",ylab="Time")
par(op)
```

`StepEuler` *Create a function for advancing the state of an ODE model by using a simple Euler integration method*

Description

This function creates a function for advancing the state of an ODE model using a simple Euler integration method. The resulting function (closure) can be used in conjunction with other functions (such as `simTs`) for simulating realisations of ODE models. This function is intended to be pedagogic. See `StepODE` for a more accurate integration function.

Usage

```
StepEuler(RHSfun,dt=0.01)
```

Arguments

`RHSfun`	A function representing the RHS of the ODE model. `RHSfun` should have prototype `RHSfun(x,t,...)`, with `x` representing current system state and `t` representing current system time. The value of the function should be a vector of the same dimension as `x`, representing the infinitesimal change in state.
`dt`	Time step to be used by the simple Euler integration method. Defaults to 0.01.

Value

An R function which can be used to advance the state of the ODE model `RHSfun` by using an Euler method with step size `dt`. The function closure has interface `function(x0,t0,deltat,...)`, where `t0` and `x0` represent the initial time and state, and `deltat` represents the amount of time by which the process should be advanced. The function closure returns a vector representing the simulated state of the system at the new time.

See Also

StepEulerSPN, StepODE, simTs, simSample

Examples

```
# Build a RHS for the Lotka-Volterra system
LVrhs <- function(x,t,th=c(c1=1,c2=0.005,c3=0.6))
{
        with(as.list(c(x,th)),{
                c( c1*x1 - c2*x1*x2 ,
                      c2*x1*x2 - c3*x2 )
        })
}
# create a stepping function
stepLV = StepEuler(LVrhs)
# step the function
print(stepLV(c(x1=50,x2=100),0,1))
# integrate the process and plot it
out = simTs(c(x1=50,x2=100),0,20,0.1,stepLV)
plot(out,plot.type="single",lty=1:2)
```

StepEulerSPN *Create a function for advancing the state of an SPN by using a simple continuous deterministic Euler integration method*

Description

This function creates a function for advancing the state of an SPN model using a simple continuous deterministic Euler integration method. The resulting function (closure) can be used in conjunction with other functions (such as simTs) for simulating realisations of SPN models.

Usage

```
StepEulerSPN(N,dt=0.01)
```

Arguments

N	An R list with named components representing a stochastic Petri net. Should contain N$Pre, a matrix representing the LHS stoichiometries, N$Post, a matrix representing the RHS stoichiometries, and N$h, a function representing the rates of the reaction processes. N$h should have first argument x, a vector representing the current state of the system, and second argument t, a scalar representing the current simulation time (in the typical time-homogeneous case, N$h will ignore this argument). N$h may possess additional arguments, representing reaction rates, for example. N does not need to contain an initial marking, N$M. N$M will be ignored by most functions which use the resulting function closure.
dt	Time step to be used by the simple Euler integration method. Defaults to 0.01.

Value

An R function which can be used to advance the state of the SPN model `N` by using an Euler method with step size `dt`. The function closure has interface `function(x0,t0,deltat,...)`, where `x0` and `t0` represent the initial state and time, and `deltat` represents the amount of time by which the process should be advanced. The function closure returns a vector representing the simulated state of the system at the new time.

See Also

`StepGillespie`, `StepODE`, `StepCLE`, `simpleEuler`, `simTs`, `simSample`

Examples

```
# load the LV model
data(spnModels)
# create a stepping function
stepLV = StepEulerSPN(LV)
# step the function
print(stepLV(c(x1=50,x2=100),0,1))
# integrate the process and plot it
out = simTs(c(x1=50,x2=100),0,100,0.1,stepLV)
plot(out)
plot(out,plot.type="single",lty=1:2)
```

StepFRM — *Create a function for advancing the state of an SPN by using Gillespie's first reaction method*

Description

This function creates a function for advancing the state of an SPN model using Gillespie's first reaction method. The resulting function (closure) can be used in conjunction with other functions (such as `simTs`) for simulating realisations of SPN models.

Usage

```
StepFRM(N)
```

Arguments

`N`	An R list with named components representing a stochastic Petri net. Should contain `N$Pre`, a matrix representing the LHS stoichiometries, `N$Post`, a matrix representing the RHS stoichiometries, and `N$h`, a function representing the rates of the reaction processes. `N$h` should have first argument `x`, a vector representing the current state of the system, and second argument `t`, a scalar representing the current simulation time (in the typical time-homogeneous case, `N$h` will ignore this argument). `N$h` may possess additional arguments, representing reaction rates, for example. `N` does not need to contain an initial marking, `N$M`. `N$M` will be ignored by most functions which use the resulting function closure.

Value

An R function which can be used to advance the state of the SPN model `N` by using Gillespie's first reaction method. The function closure has interface `function(x0,t0,deltat,...)`, where `x0` and `t0` represent the initial state and time, and `deltat` represents the amount of time by which the process should be advanced. The function closure returns a vector representing the simulated state of the system at the new time.

See Also

`StepEulerSPN`, `StepGillespie`, `simTs`, `simSample`

Examples

```
# load the LV model
data(spnModels)
# create a stepping function
stepLV = StepFRM(LV)
# step the function
print(stepLV(c(x1=50,x2=100),0,1))
# simulate a realisation of the process and plot it
out = simTs(c(x1=50,x2=100),0,100,0.1,stepLV)
plot(out,plot.type="single",lty=1:2)
```

`StepGillespie` *Create a function for advancing the state of an SPN by using the Gillespie algorithm*

Description

This function creates a function for advancing the state of an SPN model using the Gillespie algorithm. The resulting function (closure) can be used in conjunction with other functions (such as `simTs`) for simulating realisations of SPN models.

Usage

```
StepGillespie(N)
```

Arguments

`N`	An R list with named components representing a stochastic Petri net (SPN). Should contain `N$Pre`, a matrix representing the LHS stoichiometries, `N$Post`, a matrix representing the RHS stoichiometries, and `N$h`, a function representing the rates of the reaction processes. `N$h` should have first argument `x`, a vector representing the current state of the system, and second argument `t`, a scalar representing the current simulation time (in the typical time-homogeneous case, `N$h` will ignore this argument). `N$h` may possess additional arguments, representing reaction rates, for example. `N` does not need to contain an initial marking, `N$M`. `N$M` will be ignored by most functions which use the resulting function closure.

Value

An R function which can be used to advance the state of the SPN model `N` by using the Gillespie algorithm. The function closure has interface `function(x0,t0,deltat,...)`, where `x0` and `t0` represent the initial state and time, and `deltat` represents the amount of time by which the process should be advanced. The function closure returns a vector representing the simulated state of the system at the new time.

See Also

StepEulerSPN, StepGillespie1D, simTs, simTimes, simSample, StepFRM, StepPTS, StepCLE

Examples

```
# load up the Lotka-Volterra (LV) model
data(spnModels)
LV
# create a stepping function
stepLV = StepGillespie(LV)
# step the function
print(stepLV(c(x1=50,x2=100),0,1))
# simulate a realisation of the process and plot it
out = simTs(c(x1=50,x2=100),0,100,0.1,stepLV)
plot(out)
plot(out,plot.type="single",lty=1:2)
# simulate a realisation using simTimes
times = seq(0,100,by=0.1)
plot(ts(simTimes(c(x1=50,x2=100),0,times,stepLV),start=0,deltat=0.1),plot.type="single",lty=1:2)
# simulate a realisation at irregular times
times = c(0,10,20,50,100)
out2 = simTimes(c(x1=50,x2=100),0,times,stepLV)
print(out2)
```

StepGillespie1D — *Create a function for advancing the state of an SPN by using the Gillespie algorithm on a 1D regular grid*

Description

This function creates a function for advancing the state of an SPN model using the Gillespie algorithm. The resulting function (closure) can be used in conjunction with other functions (such as simTs1D) for simulating realisations of SPN models in space and time.

Usage

```
StepGillespie1D(N,d)
```

Arguments

N	An R list with named components representing a stochastic Petri net (SPN). Should contain N$Pre, a matrix representing the LHS stoichiometries, N$Post, a matrix representing the RHS stoichiometries, and N$h, a function representing the rates of the reaction processes. N$h should have first argument x, a vector representing the current state of the system, and second argument t, a scalar representing the current simulation time (in the typical time-homogeneous case, N$h will ignore this argument). N$h may possess additional arguments, representing reaction rates, for example. N does not need to contain an initial marking, N$M. N$M will be ignored by most functions which use the resulting function closure.
d	A vector of diffusion coefficients - one coefficient for each reacting species, in order. The coefficient is the reaction rate for a reaction for a molecule moving into an adjacent compartment. The hazard for a given molecule leaving the compartment is therefore twice this value (as it can leave to the left or the right).

Value

An R function which can be used to advance the state of the SPN model N by using the Gillespie algorithm. The function closure has interface function(x0,t0,deltat,...), where x0 is a matrix with rows corresponding to species and columns corresponding to voxels, representing the initial condition, t0 represent the initial state and time, and deltat represents the amount of time by which the process should be advanced. The function closure returns a matrix representing the simulated state of the system at the new time.

See Also

StepGillespie, simTs1D, StepGillespie2D

Examples

```
data(spnModels)
N=20; T=30
x0=matrix(0,nrow=2,ncol=N)
rownames(x0)=c("x1","x2")
x0[,round(N/2)]=LV$M
stepLV1D = StepGillespie1D(LV,c(0.6,0.6))
xx = simTs1D(x0,0,T,0.2,stepLV1D,verb=TRUE)
op=par(mfrow=c(1,2))
image(xx[1,,],main="Prey",xlab="Space",ylab="Time")
image(xx[2,,],main="Predator",xlab="Space",ylab="Time")
par(op)
```

StepGillespie2D *Create a function for advancing the state of an SPN by using the Gillespie algorithm on a 2D regular grid*

Description

This function creates a function for advancing the state of an SPN model using the Gillespie algorithm. The resulting function (closure) can be used in conjunction with other functions (such as simTs2D) for simulating realisations of SPN models in space and time.

Usage

```
StepGillespie2D(N,d)
```

Arguments

N	An R list with named components representing a stochastic Petri net (SPN). Should contain N$Pre, a matrix representing the LHS stoichiometries, N$Post, a matrix representing the RHS stoichiometries, and N$h, a function representing the rates of the reaction processes. N$h should have first argument x, a vector representing the current state of the system, and second argument t, a scalar representing the current simulation time (in the typical time-homogeneous case, N$h will ignore this argument). N$h may possess additional arguments, representing reaction rates, for example. N does not need to contain an initial marking, N$M. N$M will be ignored by most functions which use the resulting function closure.
d	A vector of diffusion coefficients - one coefficient for each reacting species, in order. The coefficient is the reaction rate for a reaction for a molecule moving into an adjacent compartment. The hazard for a given molecule leaving the compartment is therefore four times this value (as it can leave in one of 4 directions).

Value

An R function which can be used to advance the state of the SPN model N by using the Gillespie algorithm. The function closure has interface function(x0,t0,deltat,...), where x0 is a 3d array with dimensions corresponding to species followed by two spatial dimensions, representing the initial condition, t0 represent the initial state and time, and deltat represents the amount of time by which the process should be advanced. The function closure returns an array representing the simulated state of the system at the new time.

See Also

StepGillespie, simTs2D, StepGillespie1D

Examples

```
data(spnModels)
m=20; n=30; T=10
x0=array(0,c(2,m,n))
dimnames(x0)[[1]]=c("x1","x2")
x0[,round(m/2),round(n/2)]=LV$M
stepLV2D = StepGillespie2D(LV,c(0.6,0.6))
xx = simTs2D(x0,0,T,0.2,stepLV2D,verb=TRUE)
N = dim(xx)[4]
op=par(mfrow=c(1,2))
image(xx[1,,,N],main="Prey",xlab="Space",ylab="Time")
image(xx[2,,,N],main="Predator",xlab="Space",ylab="Time")
par(op)
```

`stepLVc` *A function for advancing the state of a Lotka-Volterra model by using the Gillespie algorithm*

Description

A function for advancing the state of a Lotka-Volterra model by calling some C code implementing the Gillespie algorithm. The function can be used in conjunction with other functions (such as `simTs`) for simulating realisations of Lotka-Volterra models. Should be functionally identical to the function obtained by `data(spnModels)`, `stepLV=StepGillespie(LV)`, but much faster.

Usage

```
stepLVc(x0,t0,deltat,th=c(1,0.005,0.6))
```

Arguments

`x0`	A vector representing the state of the system at the initial time, `t0`.
`t0`	The time corresponding to the initial state, `x0`.
`deltat`	The time in advance of the initial time at which the new state is required.
`th`	A vector of length 3 representing the rate constants associated with the 3 LV reactions. Defaults to `c(1,0.005,0.6)`.

Value

A 2-vector representing the new state of the LV system.

See Also

`StepGillespie`, `spnModels`, `simTs`, `simSample`

Examples

```
# load the LV model
data(spnModels)
# create a stepping function
stepLV = StepGillespie(LV)
# step the function
print(stepLV(c(x1=50,x2=100),0,1))
# simulate a realisation of the process and plot it
out = simTs(c(x1=50,x2=100),0,100,0.1,stepLV)
plot(out)
# now use "stepLVc" instead...
out = simTs(c(x1=50,x2=100),0,100,0.1,stepLVc)
plot(out)
```

`StepODE` *Create a function for advancing the state of an ODE model by using the deSolve package*

Description

This function creates a function for advancing the state of an ODE model using an integration method from the `deSolve` package. The resulting function (closure) can be used in conjunction with other functions (such as `simTs`) for simulating realisations of ODE models. This function is used similarly to `StepEuler`, but `StepODE` should be more accurate and efficient.

Usage

```
StepODE(RHSfun)
```

Arguments

`RHSfun`	A function representing the RHS of the ODE model. `RHSfun` should have prototype `RHSfun(x,t,parms,...)`, with `t` representing current system time, `x` representing current system state and `parms` representing the model parameters. The value of the function should be a vector of the same dimension as `x`, representing the infinitesimal change in state.

Value

An R function which can be used to advance the state of the ODE model `RHSfun` by using an efficient ODE solver. The function closure has interface `function(x0,t0,deltat,parms,...)`, where `t0` and `x0` represent the initial time and state, and `deltat` represents the amount of time by which the process should be advanced. The function closure returns a vector representing the simulated state of the system at the new time.

See Also

`StepEulerSPN`, `StepEuler`, `simTs`, `ode`

Examples

```
# Build a RHS for the Lotka-Volterra system
LVrhs <- function(x,t,parms)
{
        with(as.list(c(x,parms)),{
                c( c1*x1 - c2*x1*x2 ,
                      c2*x1*x2 - c3*x2 )
        })
}
# create a stepping function
stepLV = StepODE(LVrhs)
# step the function
print(stepLV(c(x1=50,x2=100),0,1,parms=c(c1=1,c2=0.005,c3=0.6)))
# integrate the process and plot it
out = simTs(c(x1=50,x2=100),0,50,0.1,stepLV,parms=c(c1=1,c2=0.005,c3=0.6))
plot(out,plot.type="single",lty=1:2)
```

`StepPTS` *Create a function for advancing the state of an SPN by using a simple approximate Poisson time stepping method*

Description

This function creates a function for advancing the state of an SPN model using a simple approximate Poisson time stepping method. The resulting function (closure) can be used in conjunction with other functions (such as `simTs`) for simulating realisations of SPN models.

Usage

```
StepPTS(N,dt=0.01)
```

Arguments

`N`	An R list with named components representing a stochastic Petri net. Should contain `N$Pre`, a matrix representing the LHS stoichiometries, `N$Post`, a matrix representing the RHS stoichiometries, and `N$h`, a function representing the rates of the reaction processes. `N$h` should have first argument `x`, a vector representing the current state of the system, second argument `t`, a scalar representing the current simulation time (in the typical time-homogeneous case, `N$h` will ignore this argument). `N$h` may posses additional arguments, representing reaction rates, for example. `N` does not need to contain an initial marking, `N$M`. `N$M` will be ignored by most functions which use the resulting function closure.
`dt`	Time step to be used by the Poisson time stepping integration method. Defaults to 0.01.

Value

An R function which can be used to advance the state of the SPN model `N` by using a Poisson time stepping method with step size `dt`. The function closure has interface `function(x0,t0,deltat,...)`, where `x0` and `t0` represent the initial state and time, and `deltat` represents the amount of time by which the process should be advanced. The function closure returns a vector representing the simulated state of the system at the new time.

See Also

`StepGillespie`, `StepCLE`, `simTs`, `simSample`

Examples

```
# load up the LV model
data(spnModels)
# create a stepping function
stepLV=StepPTS(LV)
# step the function
print(stepLV(c(x1=50,x2=100),0,1))
# integrate the process and plot it
out = simTs(c(x1=50,x2=100),0,20,0.1,stepLV)
plot(out)
plot(out,plot.type="single",lty=1:2)
```

StepSDE — *Create a function for advancing the state of an SDE model by using a simple Euler-Maruyama integration method*

Description

This function creates a function for advancing the state of an SDE model using a simple Euler-Maruyama integration method. The resulting function (closure) can be used in conjunction with other functions (such as `simTs`) for simulating realisations of SDE models.

Usage

```
StepSDE(drift,diffusion,dt=0.01)
```

Arguments

`drift`	A function representing the drift vector of the SDE model (corresponding roughly to the RHS of an ODE model). `drift` should have prototype `drift(x,t,...)`, with `x` representing current system state and `t` representing current system time. The value of the function should be a vector of the same dimension as `x`, representing the infinitesimal mean of the Ito SDE.

`diffusion` A function representing the diffusion matrix of the SDE model (the square root of the infinitesimal variance matrix). `diffusion` should have prototype `diffusion(x,t,...)`, with `x` representing current system state and `t` representing current system time. The value of the function should be a square matrix with both dimensions the same as the length of `x`.

`dt` Time step to be used by the simple Euler-Maruyama integration method. Defaults to 0.01.

Value

An R function which can be used to advance the state of the SDE model with given drift vector and diffusion matrix, by using an Euler-Maruyama method with step size `dt`. The function closure has interface `function(x0,t0,deltat,...)`, where `x0` and `t0` represent the initial state and time, and `deltat` represents the amount of time by which the process should be advanced. The function closure returns a vector representing the simulated state of the system at the new time.

See Also

StepEuler, StepCLE, simTs, simSample

Examples

```
# Immigration-death diffusion approx with death rate a CIR process
myDrift <- function(x,t,th=c(lambda=1,alpha=1,mu=0.1,sigma=0.1))
     {
             with(as.list(c(x,th)),{
                     c( lambda - x*y ,
                           alpha*(mu-y) )
             })
     }
myDiffusion <- function(x,t,th=c(lambda=1,alpha=1,mu=0.1,sigma=0.1))
     {
             with(as.list(c(x,th)),{
                     matrix(c( sqrt(lambda + x*y) , 0,
                           0, sigma*sqrt(y) ),ncol=2,nrow=2,byrow=TRUE)
             })
     }
# create a stepping function
stepProc = StepSDE(myDrift,myDiffusion)
# integrate the process and plot it
out = simTs(c(x=5,y=0.1),0,20,0.1,stepProc)
plot(out)
```

Index