Package 'GillespieSSA'

July 2, 2014

Title Gillespie's Stochastic Simulation Algorithm (SSA)

Version 0.5-4

Date 2010-08-16

Author Mario Pineda-Krch <mpineda@math.ualbeta.ca>

Maintainer ORPHANED

Depends R (>= 2.0.0)

Description GillespieSSA provides a simple to use, intuitive, and extensible interface to several stochastic simulation algorithms for generating simulated trajectories of finite population continuous-time model. Currently it implements Gillespie's exact stochastic simulation algorithm (Direct method) and several approximate methods (Explicit tau-leap,Binomial tau-leap, and Optimized tau-leap). The package also contains a library of template models that can be run as demo models and can easily be customized and extended. Currently the following models are included, decaying-dimerization reaction set, linear chain system, logistic growth model, Lotka predator-prey model, Rosenzweig-MacArthur predator-prey model,Kermack-McKendrick SIR model, and a metapopulation SIRS model.

License GPL (>= 3)

URL http://pineda-krch.com/gillespiessa

Repository CRAN

Date/Publication 2012-01-15 12:32:46

NeedsCompilation no

X-CRAN-Original-Maintainer Mario Pineda-Krch <mpineda@math.ualbeta.ca>

X-CRAN-Comment Orphaned on 2014-01-12 as maintainer address <mpineda@math.ualbeta.ca> bounced.

R topics documented:

`GillespieSSA-package` *Gillespie Stochastic Simulation Algorithm package*

Description

Package description and overview of basic SSA theory

Details

GillespieSSA is a versatile and extensible framework for stochastic simulation in R and provides a simple interface to a number of Monte Carlo implementations of the stochastic simulation algorithm (SSA). The methods currently implemented are: the Direct method (D), Explicit tau-leaping (ETL), Binomial tau-leaping (BTL), and Optimized tau-leaping (OTL). The package also provides a library of ecological, epidemiological, and evolutionary continuous-time (demo) models that can easily be customized and extended. Currently the following models are included, Decaying-Dimerization Reaction Set, Linear Chain System, single-species logistic growth model, Lotka predator-prey model, Rosenzweig-MacArthur predator-prey model, Kermack-McKendrick SIR model, and a metapopulation SIRS model.

The stochastic simulation algorithm

The stochastic simulation algorithm (SSA) is a procedure for constructing simulated trajectories of finite populations in continuous time. If $X_i(t)$ is the number of individuals in population i ($i = 1, \ldots, N$) at time t the SSA estimates the state vector $\mathbf{X}(t) \equiv (X_1(t), \ldots, X_N(t))$, given that the system initially (at time t_0) was in state $\mathbf{X}(t_0) = \mathbf{x_0}$. Reactions, single instantaneous events changing at least one of the populations (e.g. birth, death, movement, collision, predation, infection, etc), cause the state of the system to change over time. The SSA procedure samples the time τ to the next reaction R_j ($j = 1, \ldots, M$) and updates the system state $\mathbf{X}(t)$ accordingly. Each reaction R_j is characterized mathematically by two quantities; its state-change vector $\boldsymbol{\nu}_j \equiv (\nu_{1j}, \ldots, \nu_{Nj})$, where ν_{ij} is the change in the number of individuals in population i caused by one reaction of type j and its propensity function $a_j(\mathbf{x})$, where $a_j(\mathbf{x})dt$ is the probability that a particular reaction j will occur in the next infinitesimal time interval $[t, t + dt]$.

SSA implementations

There are numerous exact Monte Carlo procedures implementing the `SSA`. Perhaps the simplest is the Direct method of Gillespie (1977. The Direct method is an exact continuous-time numerical realization of the corresponding stochastic time-evolution equation. Because the Direct method simulates one reaction at a time it is often, however, computationally too slow for practical applications.

Approximate implementations of the `SSA` sacrifices exactness for large improvements in computational efficiency. The most common technique used is tau-leaping where reaction-bundles are attempted in coarse-grained time increments τ. Speed-ups of several orders of magnitude compared to the Direct method are common. Tau-leaping must be used with care, however, as it is not as foolproof as the Direct method.

Example models

Individual demo models can be run by issuing `demo(<model name>)`, alternatively all of the demo models can be run using `demo(GillespieSSA)`. The following example models are available:

Decaying-Dimerization Reaction Set (Gillespie, 2001)
`file.show(system.file("demo/decayingDimer.R", package = "GillespieSSA"))`

Linear Chain System (Cao et al., 2004)
`file.show(system.file("demo/linearChain.R", package = "GillespieSSA"))`

Logistic growth model (Kot, 2001)
`file.show(system.file("demo/logisticGrowth.R", package = "GillespieSSA"))`

Lotka predator-prey model (Gillespie, 1977; Kot, 2001)
`file.show(system.file("demo/lotka.R", package = "GillespieSSA"))`

Kermack-McKendrick SIR model (Brown & Rothery, 1993)
`file.show(system.file("demo/sir.R", package = "GillespieSSA"))`

Logistic growth (Pearl-Verhulst model) (Kot, 2001, Pineda-Krch, 2008)
Rosenzweig-MacArthur predator-prey model (Pineda-Krch et al., 2007, Pineda-Krch, 2008)
`file.show(system.file("demo/rma.R", package = "GillespieSSA"))`

Metapopulation SIRS model (Pineda-Krch, 2008)
`file.show(system.file("demo/epiChain.R", package = "GillespieSSA"))`

Note, the last three models are part of a manuscript to be published in the Journal of Statistical Software (preprint available on request).

How to cite this package

```
author = Mario Pineda-Krch
title  = GillespieSSA: a stochastic simulation package for R
year   = 2010
url    = http://pineda-krch.com/gillespiessa
```

License

Acknowledgements

- Heinrich zu Dohna for many caffein induced discussions on the package and reference manual, and for providing comments on the vignette documentation.
- Ben Bolker for comments on the initial release of the package and for providing a hint for how to more elegantly handle model parameters as arguments to the ssa() function.
- Josh Obrien for copy editing and feedback on the JSS manuscript.
- Thomas Petzoldt for comments on the package, the JSS manuscript and for preparing version 0.5-4.
- Three anonymous referees whose comments substantially improved some of the functionality.

Author(s)

Mario Pineda-Krch (http://pineda-krch.com)

References

- Brown D. and Rothery P. 1993. Models in biology: mathematics, statistics, and computing. John Wiley & Sons.
- Cao Y., Li H., and Petzold L. 2004. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. J. Chem. Phys. 121:4059-4067. http://dx.doi.org/10.1063/1.1778376

- Cao Y., Gillespie D.T., and Petzold L.R. 2006. Efficient step size selection for the tau-leaping method. J. Chem. Phys. 124:044109. http://dx.doi.org/10.1063/1.2159468
- Cao Y., Gillespie D.T., and Petzold L.R. 2007. Adaptive explicit tau-leap method with automatic tau selection. J. Chem. Phys. 126:224101. http://dx.doi.org/10.1063/1.2745299
- Chatterjee A., Vlachos D.G., and Katsoulakis M.A. 2005. Binomial distribution based tau-leap accelerated stochastic simulation. J. Chem. Phys. 122:024112. http://dx.doi.org/10.1063/1.1833357
- Gillespie D.T. 1977. Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. 81:2340. http://dx.doi.org/10.1021/j100540a008
- Gillespie D.T. 2001. Approximate accelerated stochastic simulation of chemically reacting systems. J. Chem. Phys. 115:1716-1733. http://dx.doi.org/10.1063/1.1378322
- Gillespie D.T. 2007. Stochastic simulation of chemical kinetics. Annu. Rev. Chem. 58:35 http://dx.doi.org/10.1146/annurev.physchem.58.032806.104637
- Kot M. 2001. Elements of mathematical ecology. Cambridge University Press. http://dx.doi.org/10.2277/052180213X
- Pineda-Krch M. 2008. Implementing the stochastic simulation algorithm in R. Submitted to the Journal of Statistical Software 25(12): 1-18. http://www.jstatsoft.org/v25/i12
- Pineda-Krch M., Blok H.J., Dieckmann U., and Doebeli M. 2007. A tale of two cycles — distinguishing quasi-cycles and limit cycles in finite predator-prey populations. Oikos 116:53-64. http://dx.doi.org/10.1111/j.2006.0030-1299.14940.x

See Also

ssa, ssa.d, ssa.etl, ssa.btl, ssa.otl, ssa.plot

---

ssa *Invoking the stochastic simulation algorithm*

Description

Main interface function to the implemented SSA methods. Runs a single realization of a predefined system.

Usage

```
ssa( x0,          # initial state vector
      a,          # propensity vector
     nu,          # state-change matrix
  parms = NULL, # model parameters
     tf,          # final time
 method = "D",  # SSA method
simName = "",
    tau = 0.3,  # only applicable for ETL
      f = 10,   # only applicable for BTL
epsilon = 0.03, # only applicable for OTL
```

```
                   nc = 10,   # only applicable for OTL
                  hor = NaN,  # only applicable for OTL
                  dtf = 10,   # only applicable for OTL
                   nd = 100,  # only applicable for OTL
ignoreNegativeState = TRUE,
    consoleInterval = 0,
     censusInterval = 0,
            verbose = FALSE,
        maxWallTime = Inf)
```

Arguments

`x0`	numerical vector of initial states where the component elements must be named using the same notation as the corresponding state variable in the propensity vector, `a`.
`a`	character vector of propensity functions where state variables correspond to the names of the elements in `x0`.
`nu`	numerical matrix of change if the number of individuals in each state (rows) caused by a single reaction of any given type (columns).
`parms`	named vector of model parameters.
`tf`	final time.
`method`	text string indicating the `SSA` method to use, the valid options are: `D` — Direct method (default method), `ETL` - Explicit tau-leap, `BTL` — Binomial tau-leap, or `OTL` — Optimized tau-leap.
`simName`	optional text string providing an arbitrary name/label for the simulation.
`tau`	step size for the `ETL` method (> 0).
`f`	coarse-graining factor for the `BTL` method (> 1) where a higher value results in larger step-size.
`epsilon`	accuracy control parameter for the `OTL` method (> 0).
`nc`	critical firing threshold for the `OTL` method (positive integer).
`hor`	numerical vector of the highest order reaction for each species where `hor` $\in \{1, 2, 22\}$. Setting `hor=NaN` uses the default `hor=rep(22,N)` where `N` is the number of species (See page 6 in Cao et al. 2006). Unless `hor=NaN` the number of elements must equal the number of states N. Only applicable in the `OTL` method.
`dtf`	`D` method threshold factor for the `OTL` method. The `OTL` method is suspended if `tau` it estimates is smaller than the `dtf` multiple of the `tau` that the `D` method would have used (i.e. $\tau_{\texttt{OTL}} < \texttt{dtf} \times \tau_{\texttt{D}}$) (See step 3, page 3 in Cao et al. 2006).
`nd`	number of single-reaction steps performed using the Direct method during `otl` suspension (See step 3, page 3, Cao et al. 2006).
`ignoreNegativeState`	boolean object indicating if negative state values should be ignored (this can occur in the `etl` method). If `ignoreNegativeState=TRUE` the simulation finishes gracefully when encountering a negative population size (i.e. does not throw an error). If `ignoreNegativeState=FALSE` the simulation stops with an error message when encountering a negative population size.

`consoleInterval`
: (approximate) interval at which `ssa` produces simulation status output on the console (assumes `verbose=TRUE`). If `consoleInterval=0` console output is generated each time step (or tau-leap). If `consoleInterval=Inf` no console output is generated. Note, `verbose=FALSE` disables all console output. **Console output drastically slows down simulations.**

`censusInterval`
: (approximate) interval between recording the state of the system. If `censusInterval=0` (t, x) is recorded at each time step (or tau-leap). If `censusInterval=Inf` only (t_0, x_0) and (t_f, x_t) is recorded. Note, the size of the time step (or tau-leaps) ultimately limits the interval between subsequent recordings of the system state since the state of the system cannot be recorded at a finer time interval the size of the time steps (or tau-leaps).

`verbose`
: boolean object indicating if the status of the simulation simulation should be displayed on the console. If `verbose=TRUE` the elapsed wall time and (t, x) is displayed on the console every `consoleInterval` time step and a brief summary is displayed at the end of the simulation. If `verbose=FALSE` the simulation runs *entirely* silent (overriding `consoleInterval`). **Verbose runs drastically slows down simulations.**

`maxWallTime`
: maximum wall time duration (in seconds) that the simulation is allowed to run for before terminated. This option is usefull, in particular, for systems that can end up growing uncontrolably.

Details

Although `ssa` can be invoked by only specifying the system arguments (initial state vector `x0`, propensity vector `a`, state-change matrix `nu`), the final time (`tf`), and the `SSA` method to use, substantial improvements in speed and accuracy can be obtained by adjusting the additional (and optional) `ssa` arguments. By default `ssa` (tries to) use conservative default values for the these arguments, prioritizing computational accuracy over computational speed. These default values are, however, **not** fool proof for the approximate methods, and occasionally one will have to hand tweak them in order for a stochastic model to run appropriately.

Value

Returns a list object with the following elements,

`data`
: a numerical matrix object of the simulation time series where the first column is the time vector and subsequent columns are the state frequencies.

`stats`
: sub-list object with elements containing various simulation statistics. The of the sub-list are:

`stats\$startWallTime`
: start wall clock time (YYYY-mm-dd HH:MM:SS).

`stats\$endWallTime`
: end wall clock time (YYYY-mm-dd HH:MM:SS).

`stats\$elapsedWallTime`
: elapsed wall time in seconds.

`stats\$terminationStatus`	string vector listing the reason(s) for the termination of the realization in 'plain words'. The possible termination statuses are: `finalTime` = if the simulation reached the maximum simulation time `tf`, `extinction` = if the population size of all states is zero, `negativeState` = if one or several states have a negative population size (can occur in the ETL method), `zeroProp` = if all the states have a zero propensity function, `maxWallTime` = if the maximum wall time has been reached. Note the termination status may have more than one message.
`stats\$nSteps`	total number of time steps (or tau-leaps) executed.
`stats\$meanStepSize`	mean step (or tau-leap) size.
`stats\$sdStepSize`	one standard deviation of the step (or tau-leap) size.
`stats\$SuspendedTauLeaps`	number of steps performed using the Direct method due to `OTL` suspension (only applicable for the `OTL` method).
`arg\$...`	sub-list with elements containing all the arguments and their values used to invoke `ssa` (see Usage and Arguments list above).

Preparing a run

In order to invoke SSA the stochastic model needs at least four components, the initial state vector (`x0`), state-change matrix (`nu`), propensity vector (`a`), and the final time of the simulation (`tf`). The initial state vector defines the population sizes in all the states at $t = 0$, e.g. for a system with two species `X1` and `X2` where both have an initial population size of 1000 the initial state vector is defined as `x0 <- c(X1=1000,X2=1000)`. The elements of the vector have to be labelled using the same notation as the state variables used in the propensity functions. The state-change matrix defines the change in the number of individuals in each state (rows) as caused by one reaction of a given type (columns). For example, the state-change matrix for system with the species S_1 and S_2 with two reactions

$$S_1 \xrightarrow{c_1} S_2$$
$$S_2 \xrightarrow{c_2} 0$$

is defined as `nu <- matrix(c(-1,0,+1,-1),nrow=2,byrow=TRUE)` where c_1 and c_2 are the per capita reaction probabilities. The propensity vector, `a`, defines the probabilities that a particular reaction will occur over the next infinitesimal time interval $[t, t + dt]$. For example, in the previous example the propensity vector is defined as `a <- c("c1*X1","c2*X2")`. The propensity vector consists of character elements of each reaction's propensity function where each state variable requires the corresponding named element label in the initial state vector (`x0`).

Example: Irreversible isomerization

Perhaps the simplest model that can be formulated using the SSA is the irreversible isomerization (or radioactive decay) model. This model is often used as a first pedagogic example to illustrate the SSA (see e.g. Gillespie 1977). The deterministic formulation of this model is

$$\frac{dX}{dt} = -cX$$

where the single reaction channel is

$$S \xrightarrow{c} 0$$

By setting $X_0 = 1000$ and $c = 0.5$ it is now simple to define this model and run it for 10 time steps using the Direct method,

```
out <- ssa(x0=c(X=1000),a=c("c*X"),nu=matrix(-1),parms=c(c=0.5),tf=10)
```

The resulting time series can then be displayed by,

```
ssa.plot(out)
```

Note

Selecting the appropriate SSA method is a trade-off between computational speed, accuracy of the results, and which SSA actually works for a given scenario. This depends on the characteristics of the defined system (e.g. number of reaction channels, number of species, and the absolute and relative magnitude of the propensity functions). **All methods are not appropriate for all models.** When selecting a SSA method all of these factors have to be taken into consideration. The various tau-leap methods accept a number of additional arguments. While the default values of these arguments may work for some scenarios they may have to be adjusted for others. The default values for the tau-leap methods are conservative in terms of computational speed and substantial increase in efficiency may be gained by optimizing their values for a specific system.

Author(s)

Mario Pineda-Krch (http://pineda-krch/gillespiessa)

See Also

GillespieSSA-package, ssa.d, ssa.etl, ssa.btl, ssa.otl, ssa.plot

Examples

```
## Irreversible isomerization
## Large initial population size (X=1000)
## Not run:
parms <- c(c=0.5)
x0  <- c(X=10000)
a   <- c("c*X")
nu  <- matrix(-1)
out <- ssa(x0,a,nu,parms,tf=10,simName="Irreversible isomerization") # Direct method
plot(out$data[,1],out$data[,2]/10000,col="red",cex=0.5,pch=19)

## End(Not run)

## Smaller initial population size (X=100)
## Not run:
x0  <- c(X=100)
```

```
out <- ssa(x0,a,nu,parms,tf=10) # Direct method
points(out$data[,1],out$data[,2]/100,col="green",cex=0.5,pch=19)

## End(Not run)

## Small initial population size (X=10)
## Not run:
x0  <- c(X=10)
out <- ssa(x0,a,nu,parms,tf=10) # Direct method
points(out$data[,1],out$data[,2]/10,col="blue",cex=0.5,pch=19)

## End(Not run)

## Logistic growth
## Not run:
parms <- c(b=2, d=1, K=1000)
x0  <- c(N=500)
a   <- c("b*N", "(d+(b-d)*N/K)*N")
nu  <- matrix(c(+1,-1),ncol=2)
out <- ssa(x0,a,nu,parms,tf=10,method="D",maxWallTime=5,simName="Logistic growth")
ssa.plot(out)

## End(Not run)

## Kermack-McKendrick SIR model
## Not run:
parms <- c(beta=0.001, gamma=0.1)
x0  <- c(S=499,I=1,R=0)
a   <- c("beta*S*I","gamma*I")
nu  <- matrix(c(-1,0,+1,-1,0,+1),nrow=3,byrow=TRUE)
out <- ssa(x0,a,nu,parms,tf=100,simName="SIR model")
ssa.plot(out)

## End(Not run)

## Lotka predator-prey model
## Not run:
parms <- c(c1=10, c2=.01, c3=10)
x0  <- c(Y1=1000,Y2=1000)
a   <- c("c1*Y1","c2*Y1*Y2","c3*Y2")
nu  <- matrix(c(+1,-1,0,0,+1,-1),nrow=2,byrow=TRUE)
out <- ssa(x0,a,nu,parms,tf=100,method="ETL",simName="Lotka predator-prey model")
ssa.plot(out)

## End(Not run)
```

ssa.btl *Binomial tau-leap method (BTL)*

Description

Binomial tau-leap method implementation of the SSA as described by Chatterjee et al. (2005). It is usually called from within `ssa`, but can be invoked directly.

Usage

```
ssa.btl(x = stop("missing state vector (x)"),
         a = stop("missing propensity vector (a)"),
        nu = stop("missing state-change matrix (nu)"),
         f = stop("missing coarse-graining factor (f)"))
```

Arguments

`x`	state vector.
`a`	vector of evaluated propensity functions.
`nu`	state-change matrix.
`f`	coarse-graining factor (see page 4 in Chatterjee et al. 2005).

Details

Performs one time step using the Binomial tau-leap method. Intended to be invoked by `ssa`.

Value

A list with two elements, 1) the time leap (`tau`) and 2) the realized state change vector (`nu_j`).

References

Chatterjee et al. (2005)

See Also

GillespieSSA-package, `ssa`

Examples

```
a = function(parms,x){
 b <- parms[1]
 d <- parms[2]
 K <- parms[3]
 N <- x[1]
 return(c(b*N , N*b + (b-d)*N/K))
}
parms <- c(2,1,1000,500)
x <- 500
nu <- matrix(c(+1, -1),ncol=2)
t <- 0
for (i in seq(100)) {
  out <- ssa.btl(x,a(parms,x),nu,f=10)
  x <- x + out$nu_j
  t <- t + 1
  cat("t:",t,", x:",x,"\n")
}
```

ssa.btl.diag *Binomial tau-leap method (BTL) for nu-diagonalized systems*

Description

Binomial tau-leap method for nu-diagonalized systems

Usage

```
ssa.btl.diag(x,a,nu_tile,f)
```

Arguments

`x`	state vector.
`a`	vector of evaluated propensity functions.
`nu_tile`	state-change matrix.
`f`	coarse-graining factor (see page 4 in Chatterjee et al. 2005).

Details

Performs one time step using the Binomial tau-leap method. It is usually called from within `ssa`, but can be invoked directly, see `ssa.btl` for Examples.

Value

A list with two elements, 1) the time leap (`tau`) and 2) the realized state change vector (`nu_j`).

See Also

`ssa.btl`,

Examples

```
## Not intended to be invoked stand alone.
```

`ssa.check.args` *Validates the arguments for the ssa wrapper function*

Description

Validates the arguments for the ssa wrapper function.

Usage

```
ssa.check.args(x0,a,nu,tf,method,tau,f,epsilon,nc,hor,dtf,nd,
               ignoreNegativeState,consoleInterval,
               censusInterval,verbose)
```

Arguments

`x0`	numerical vector of initial states where the component elements must be named using the same notation as the corresponding state variable in the propensity vector, `a`.
`a`	character vector of propensity functions where state variables correspond to the names of the elements in `x0`.
`nu`	numerical matrix of change if the number of individuals in each state (rows) caused by a single reaction of any given type (columns).
`tf`	final time.
`method`	text string indicating the SSA method to use, the valid options are: `D` — Direct method (default method), `ETL` - Explicit tau-leap, `BTL` — Binomial tau-leap, or `OTL` — Optimized tau-leap.
`tau`	step size for the `ETL` method (> 0).
`f`	coarse-graining factor for the `BTL` method (> 1) where a higher value results in larger step-size.
`epsilon`	accuracy control parameter for the `OTL` method (> 0).
`nc`	critical firing threshold for the `OTL` method (positive integer).
`hor`	numerical vector of the highest order reaction for each species where `hor` $\in \{1, 2, 22\}$. Setting `hor=NaN` uses the default `hor=rep(22,N)` where `N` is the number of species (See page 6 in Cao et al. 2006). Unless `hor=NaN` the number of elements must equal the number of states N. Only applicable in the `OTL` method.
`dtf`	`D` method threshold factor for the `OTL` method. The `OTL` method is suspended if `tau` it estimates is smaller than the `dtf` multiple of the `tau` that the `D` method would have used (i.e. $\tau_{\texttt{OTL}} < \texttt{dtf} \times \tau_{\texttt{D}}$) (See step 3, page 3 in Cao et al. 2006).
`nd`	number of single-reaction steps performed using the Direct method during `otl` suspension (See step 3, page 3, Cao et al. 2006).

`ignoreNegativeState`
: boolean object indicating if negative state values should be ignored (this can occur in the `etl` method). If `ignoreNegativeState=TRUE` the simulation finishes gracefully when encountering a negative population size (i.e. does not throw an error). If `ignoreNegativeState=FALSE` the simulation stops with an error message when encountering a negative population size.

`consoleInterval`
: (approximate) interval at which `ssa` produces simulation status output on the console (assumes `verbose=TRUE`). If `consoleInterval=0` console output is generated each time step (or tau-leap). If `consoleInterval=Inf` no console output is generated. Note, `verbose=FALSE` disables all console output. **Console output drastically slows down simulations.**

`censusInterval`
: (approximate) interval between recording the state of the system. If `censusInterval=0` (t, x) is recorded at each time step (or tau-leap). If `censusInterval=Inf` only (t_0, x_0) and (t_f, x_t) is recorded. Note, the size of the time step (or tau-leaps) ultimately limits the interval between subsequent recordings of the system state since the state of the system cannot be recorded at a finer time interval the size of the time steps (or tau-leaps).

`verbose`
: boolean object indicating if the status of the simulation simulation should be displayed on the console. If `verbose=TRUE` the elapsed wall time and (t, x) is displayed on the console every `consoleInterval` time step and a brief summary is displayed at the end of the simulation. If `verbose=FALSE` the simulation runs *entirely* silent (overriding `consoleInterval`). **Verbose runs drastically slows down simulations.**

Details

Performs basic type checking of many of the arguments passed to the `ssa` wrapper function. Note that no logical checking is currently performed, e.g. which arguments are required with which method (see `ssa.check.method`). This function is called from within `ssa` and is not intended to be invoked stand alone.

See Also

`ssa` `ssa.check.method`

Examples

```
## Not intended to be invoked stand alone
```

`ssa.check.method` *Validates consistency of the system definition*

Description

Validates consistency of the system definition.

Usage

```
ssa.check.method(x0,a,nu,method,tau,f)
```

Arguments

`x0`	numerical vector of initial states where the component elements must be named using the same notation as the corresponding state variable in the propensity vector, `a`.
`a`	character vector of propensity functions where state variables correspond to the names of the elements in `x0`.
`nu`	numerical matrix of change if the number of individuals in each state (rows) caused by a single reaction of any given type (columns).
`method`	text string indicating the SSA method to use, the valid options are: `D` — Direct method (default method), `ETL` - Explicit tau-leap, `BTL` — Binomial tau-leap, or `OTL` — Optimized tau-leap.
`tau`	step size for the `ETL` method (> 0).
`f`	coarse-graining factor for the `BTL` method (> 1) where a higher value results in larger step-size.

Details

Performs a few basic consistency checks the defined system, e.g. that the number of rows and columns in the state-change matrix and the number of elements in the initial state vector and the vector of propensity functions are consistent. This function is called from within `ssa` and is not intended to be invoked stand alone.

See Also

`ssa` `ssa.check.args`

Examples

```
## Not intended to be invoked stand alone
```

`ssa.d` *Direct method (D)*

Description

Direct method implementation of the SSA as described by Gillespie (1977). It is usually called from within `ssa`, but can be invoked directly.

Usage

```
ssa.d(a = stop("missing propensity vector (a)"),
     nu = stop("missing state-change matrix (nu)"))
```

Arguments

a	vector of evaluated propensity functions.
nu	state-change matrix.

Details

Performs one time step using the Direct method.

Value

A list with two elements, 1) the time leap (tau) and 2) the realized state change vector (nu_j).

References

Gillespie (1977)

See Also

GillespieSSA-package, ssa

Examples

```
## Logistic growth model
a = function(parms,x){
 b <- parms[1]
 d <- parms[2]
 K <- parms[3]
 N <- x[1]
 return(c(b*N , N*b + (b-d)*N/K))
}
parms <- c(2,1,1000,500)
x <- 500
nu <- matrix(c(+1, -1),ncol=2)
t <- 0
for (i in seq(100)) {
  out <- ssa.d(a(parms,x),nu)
  x <- x + out$nu_j
  t <- t + 1
  cat("t:",t,", x:",x,"\n")
}
```

---

ssa.d.diag *Direct method (D) for nu-diagonalized systems*

Description

Direct method for nu-diagonalized systems.

Usage

```
ssa.d.diag(a,nu)
```

Arguments

`a`	vector of evaluated propensity functions.
`nu`	state-change matrix.

Details

Performs one time step using the Direct method. It is usually called from within `ssa`, but can be invoked directly, see `ssa.d` for Examples.

Value

A list with two elements, 1) the time leap (`tau`) and 2) the realized state change vector (`nu_j`).

See Also

`ssa.d`

Examples

```
## Not intended to be invoked stand alone
```

`ssa.etl` *Explicit tau-leap method (ETL)*

Description

Explicit tau-leap method implementation of the SSA as described by Gillespie (2001). It is usually called from within `ssa`, but can be invoked directly.

Usage

```
ssa.etl(a = stop("missing propensity vector (a)"),
       nu = stop("missing state-change matrix (nu)"),
      tau = stop("missing step size (tau)"))
```

Arguments

`a`	vector of evaluated propensity functions.
`nu`	state-change matrix.
`tau`	tau-leap.

Details

Performs one time step using the Explicit tau-leap method. Intended to be invoked by ssa.

Value

A list with two elements, 1) the time leap (tau) and 2) the realized state change vector (nu_j).

References

Gillespie (2001)

See Also

GillespieSSA-package, ssa

Examples

```
a = function(parms,x){
 b <- parms[1]
 d <- parms[2]
 K <- parms[3]
 N <- x[1]
 return(c(b*N , N*b + (b-d)*N/K))
}
parms <- c(2,1,1000,500)
x <- 500
nu <- matrix(c(+1, -1),ncol=2)
t <- 0
for (i in seq(100)) {
  out <- ssa.etl(a(parms,x),nu,tau=0.3)
  x <- x + out$nu_j
  t <- t + 1
  cat("t:",t,", x:",x,"\n")
}
```

---

ssa.etl.diag *Explicit tau-leap method (ETL) for nu-diagonalized systems*

Description

Explicit tau-leap method for nu-diagonalized systems.

Usage

```
ssa.etl.diag(a,nu_tile,tau)
```

Arguments

`a`	vector of evaluated propensity functions.
`nu_tile`	state-change matrix.
`tau`	tau-leap.

Details

Performs one time step using the Explicit tau-leap method. It is usually called from within `ssa`, but can be invoked directly, see `ssa.etl` for Examples.

Value

A list with two elements, 1) the time leap (`tau`) and 2) the realized state change vector (`nu_j`).

See Also

`ssa.etl`,

Examples

```
## Not intended to be invoked stand alone
```

`ssa.nutiling` *Direct method nu-diagonalization mapping*

Description

Auxiliary function for `ssa.d.diag` performing virtual mapping of nu-diagonalized systems.

Usage

```
ssa.nutiling(a,nu,j)
```

Arguments

`a`	vector of evaluated propensity functions.
`nu`	state-change matrix.
`j`	Reaction index to map

Value

The virtual realized state change vector (`nu_j`).

See Also

`ssa.d.diag` `ssa.d`

Examples

```
## Not intended to be invoked stand alone
```

`ssa.otl` *Optimized tau-leap method (OTL)*

Description

Optimized tau-leap method implementation of the SSA as described by Cao et al. (2006). It is usually called from within `ssa`, but can be invoked directly.

Usage

```
ssa.otl(x = stop("missing state vector (x)"),
        a = stop("missing propensity vector (a)"),
       nu = stop("missing state-change matrix (nu)"),
      hor = stop("missing highest order reaction vector (hot)"),
       nc = stop("missing critical reactions threshold parameter (nc)"),
  epsilon = stop("missing error control parameter"),
      dtf = stop("missing direct method threshold factor (dtf)"),
       nd = stop("missing OTL suspension duration parameter (nd)"))
```

Arguments

`x`	state vector.
`a`	vector of evaluated propensity functions.
`nu`	state-change matrix.
`hor`	highest order reaction vector (one entry per species in `x`)
`nc`	number of critical reactions threshold parameter.
`epsilon`	error control parameter.
`dtf`	Direct method threshold factor for temporarily suspending the `OTL` method.
`nd`	number of Direct method steps to perform during an `OTL` suspension.

Details

Performs one time step using the Explicit tau-leap method. Intended to be invoked by `ssa`.

Value

A list with three elements, 1) the time leap (`tau`) and 2) the realized state change vector (`nu_j`), and 3) a boolean value (`suspendedTauLeapMethod`) indicating if the simulation should revert to the Direct method for `nd` time steps.

Note

Third order-reactions ($S_1 + S_2 + S_3 \rightarrow \ldots$) are not supported currently since they are approximations to sets of coupled first- and second-order reactions). See Cao et al. (2006) for more details.

References

Cao et al. (2006)

See Also

GillespieSSA-package, ssa

Examples

```
a = function(parms,x){
 b <- parms[1]
 d <- parms[2]
 K <- parms[3]
 N <- x[1]
 return(c(b*N , N*b + (b-d)*N/K))
}
parms <- c(2,1,1000,500)
x <- 500
nu <- matrix(c(+1, -1),ncol=2)
t <- 0
for (i in seq(100)) {
  out <- ssa.otl(x,a(parms,x),nu,hor=1,nc=10,epsilon=0.03,dtf=10,nd=100)
  x <- x + out$nu_j
  t <- t + 1
  cat("t:",t,", x:",x,"\n")
}
```

ssa.otl.diag — *Optimized tau-leap method (OTL) for nu-diagonalized systems*

Description

Optimized tau-leap method for nu-diagonalized systems.

Usage

```
ssa.otl.diag(x,a,nu_tile,hor,nc,epsilon,dtf,nd)
```

Arguments

`x`	state vector.
`a`	vector of evaluated propensity functions.
`nu_tile`	state-change matrix.
`hor`	highest order reaction vector (one entry per species in `x`)
`nc`	number of critical reactions threshold parameter.
`epsilon`	error control parameter.
`dtf`	Direct method threshold factor for temporarily suspending the `OTL` method.
`nd`	number of Direct method steps to perform during an `OTL` suspension.

Details

Performs one time step using the Explicit tau-leap method. It is usually called from within `ssa`, but can be invoked directly, see `ssa.otl` for Examples.

Value

A list with three elements, 1) the time leap (`tau`) and 2) the realized state change vector (`nu_j`), and 3) a boolean value (`suspendedTauLeapMethod`) indicating if the simulation should revert to the Direct method for `nd` time steps.

Note

Third order-reactions ($S_1 + S_2 + S_3 \rightarrow \ldots$) are not supported currently since they are approximations to sets of coupled first- and second-order reactions). See Cao et al. (2006) for more details.

See Also

`ssa.otl`,

Examples

```
## Not intended to be invoked stand alone
```

`ssa.plot` *Simple plotting of ssa output*

Description

Provides basic functionally for simple and quick time series plot of simulation output from `ssa`.

Usage

```
ssa.plot(        out = stop("requires simulation output object"),
                file = "ssaplot",
                  by = 1,
           plot.from = 2,
             plot.to = dim(out$data)[2],
             plot.by = 1,
          show.title = TRUE,
         show.legend = TRUE)
```

Arguments

`out`	data object returned from `ssa`.
`file`	name of the output file (only applicable if `plot.device!="x11"`.
`by`	time increment in the plotted time series
`plot.from`	first population to plot the time series for (see note)
`plot.to`	last population to plot the time series for (see note)
`plot.by`	increment in the sequence of populations to plot the time series for (see note)
`show.title`	boolean object indicating if the plot should display a title
`show.legend`	boolean object indicating if the legend is displayed

Note

The options by, `plot.from`, `plot.to`, and `plot.by` can be used to plot a sparser sequence of data points. To plot the population sizes using a larger time interval the by option can be set, e.g. to plot only every 10th time point by=`10`. To plot only specific populations the `plot.from`, `plot.to`, and `plot.by` options can be set to subset the state vector. Note that the indexing of the populations is based on the $(t, \mathbf{X})$ vector, i.e. the first column is the time vector while the first population is index by 2 and the last population by $N + 1$. Display of a plot title above the plot and legend is optional (and are set with the arguments show.title and show.legend. Above the plot panel miscellaneous information for the simulation are displayed, i.e. method, elapsed wall time, number of time steps executed, and the number of time steps per data point.

See Also

GillespieSSA-package, `ssa`

Examples

```
## Not run:
## Define the Kermack-McKendrick SIR model and run once using the Direct method
parms <- c(beta=.001, gamma=.100)
x0 <- c(S=500, I=1, R=0)                      # Initial state vector
nu <- matrix(c(-1,0,1,-1,0,1),nrow=3,byrow=T) # State-change matrix
a  <- c("beta*S*I", "gamma*I")                # Propensity vector
tf <- 100                                     # Final time
simName <- "Kermack-McKendrick SIR"
```

```
out <- ssa(x0,a,nu,parms,tf,method="D",simName,verbose=TRUE,consoleInterval=1)

## End(Not run)

## Not run:
## Basic ssa plot
ssa.plot(out)

## End(Not run)

## Not run:
# Plot only the infectious class
ssa.plot(out,plot.from=3,plot.to=3)

## End(Not run)

## Not run:
## Multipanel plot using different SSA methods
layout(matrix(seq(4),ncol=4,byrow=TRUE))

## Using the Direct method
ssa.plot(out)

## Run and plot results using the ETL method
out <- ssa(x0,a,nu,parms,tf=100,method="ETL,simName="Kermack-McKendrick SIR")
ssa.plot(out,show.title=FALSE,show.legend=FALSE)

## Run and plot results using the BTL method
out <- ssa(x0,a,nu,parms,tf=100,method="BTL,simName="Kermack-McKendrick SIR")
ssa.plot(out,show.title=FALSE,show.legend=FALSE)

## Run and plot results using the OTL method
out <- ssa(x0,a,nu,parms,tf=100,method="OTL,simName="Kermack-McKendrick SIR")
ssa.plot(out,show.title=FALSE,show.legend=FALSE)

## End(Not run)
```

`ssa.run` *Higher-level interface to the method functions*

Description

Higher-level interface to the method functions.

Usage

```
ssa.run(x0,a,nu,parms,tf,method,tau,f,epsilon,nc,hor,dtf,nd,
        ignoreNegativeState,consoleInterval,censusInterval,
        verbose,maxWallTime)
```

Arguments

`x0`	initial states vector.
`a`	vector of propensity functions.
`nu`	state-change matrix.
`parms`	vector of model parameters.
`tf`	final time.
`method`	ssa method to use.
`tau`	step size for the `ETL` method (> 0).
`f`	coarse-graining factor for the `BTL` method (> 1) where a higher value results in larger step-size.
`epsilon`	accuracy control parameter for the `OTL` method (> 0).
`nc`	critical firing threshold for the `OTL` method (positive integer).
`hor`	numerical vector of the highest order reaction for each species where `hor` $\in \{1, 2, 22\}$. Only applicable in the `OTL` method.
`dtf`	`D` method threshold factor for the `OTL` method. The `OTL` method is suspended if `tau` it estimates is smaller than the `dtf` multiple of the `tau` that the `D` method would have used (i.e. $\tau_{\texttt{OTL}} < \texttt{dtf} \times \tau_{\texttt{D}}$) (See step 3, page 3 in Cao et al. 2006).
`nd`	number of single-reaction steps performed using the Direct method during `otl` suspension (See step 3, page 3, Cao et al. 2006).
`ignoreNegativeState`	boolean object indicating if negative state values should be ignored (this can occur in the `etl` method). If `ignoreNegativeState=TRUE` the simulation finishes gracefully when encountering a negative population size (i.e. does not throw an error). If `ignoreNegativeState=FALSE` the simulation stops with an error message when encountering a negative population size.
`consoleInterval`	(approximate) interval at which `ssa` produces simulation status output on the console (assumes `verbose=TRUE`). If `consoleInterval=0` console output is generated each time step (or tau-leap). If `consoleInterval=Inf` no console output is generated. Note, `verbose=FALSE` disables all console output. **Console output drastically slows down simulations.**
`censusInterval`	(approximate) interval between recording the state of the system. If `censusInterval=0` (t, x) is recorded at each time step (or tau-leap). If `censusInterval=Inf` only (t_0, x_0) and (t_f, x_t) is recorded. Note, the size of the time step (or tau-leaps) ultimatelly limits the interval between subsequent recordings of the system state since the state of the system cannot be recorded at a finer time interval the size of the time steps (or tau-leaps).
`verbose`	boolean object indicating if the status of the simulation simulation should be displayed on the console. If `verbose=TRUE` the elapsed wall time and (t, x) is displayed on the console every `consoleInterval` time step and a brief summary is displayed at the end of the simulation. If `verbose=FALSE` the simulation runs *entirely* silent (overriding `consoleInterval`). **Verbose runs drastically slows down simulations.**

`maxWallTime`	maximum wall time duration (in seconds) that the simulation is allowed to run for before terminated. This option is useful, in particular, for systems that can end up growing uncontrollably.

Details

Invokes a specific method function until the termination criteria are fulfilled. Updates the state vector, time, and re-evaluates the propensity functions in-between time steps. Also collects simulation data and returns it as a list object. This function is called from within `ssa` and is not intended to be invoked stand alone.

Value

Returns a list object with the following elements,

`timeSeries`	a numerical matrix object of the simulation time series where the first column is the time vector and subsequent columns are the state frequencies.
`eval_a`	vector of the evaluated propensity functions.
`elapsedWallTime`	elapsed wall time in seconds.
`startWallTime`	start wall clock time (YYYY-mm-dd HH:MM:SS)

.

`endWallTime`	end wall clock time (YYYY-mm-dd HH:MM:SS).
`stepSize`	vector of step sizes (i.e. time increments).
`nSuspendedTauLeaps`	number of steps performed using the Direct method due to `OTL` suspension (only applicable for the `OTL` method).

See Also

`ssa`

Examples

```
## Not intended to be invoked stand alone
```

`ssa.terminate` *Terminates a simulation that was invoked using ssa*

Description

Terminates a simulation that was invoked using ssa.

Usage

```
ssa.terminate(args,out.rxn,tf,method,maxWallTime,verbose)
```

Arguments

`args`	list of arguments and their values passed to `ssa`.
`out.rxn`	list object as returned from `ssa.run`.
`tf`	final time.
`method`	ssa method to use.
`maxWallTime`	maximum simulation wall time duration (in seconds) that the simulation is allowed to run for before terminated.
`verbose`	boolean value indicating if some basic simulation statistics should be displayed on the console.

Details

Terminates an invocation of the `link{ssa}` wrapper function. Returns the same list object as `ssa`. This function is called from within `ssa` and is not intended to be invoked stand alone.

See Also

`ssa`

Examples

```
## Not intended to be invoked stand alone
```

Index