Package 'GillespieSSA2'

September 23, 2022

Type Package

Title Gillespie's Stochastic Simulation Algorithm for Impatient People

Version 0.2.10

Description A fast, scalable, and versatile framework for simulating large systems with Gillespie's Stochastic Simulation Algorithm ('SSA'). This package is the spiritual successor to the 'GillespieSSA' package originally written by Mario Pineda-Krch. Benefits of this package include major speed improvements (>100x), easier to understand documentation, and many unit tests that try to ensure the package works as intended. Cannoodt and Saelens et al. (2020) <doi:10.1101/2020.02.06.936971>.

License GPL (>= 3)

URL https://github.com/rcannood/GillespieSSA2

BugReports https://github.com/rcannood/GillespieSSA2/issues

Depends R (>= 3.3)

Imports assertthat, dplyr, dynutils, Matrix, methods, purrr, Rcpp (>= 0.12.3), RcppXPtrUtils, readr, rlang, stringr, tidyr

Suggests covr, ggplot2, GillespieSSA, knitr, rmarkdown, testthat (>= 2.1.0)

LinkingTo Rcpp

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation yes

Author Robrecht Cannoodt [aut, cre] (<https://orcid.org/0000-0003-3641-729X>), Wouter Saelens [aut] (<https://orcid.org/0000-0002-7114-6248>)

Maintainer Robrecht Cannoodt <rcannood@gmail.com>

Repository CRAN

Date/Publication 2022-09-23 05:10:02 UTC

R topics documented:

`compile_reactions` *Precompile the reactions*

Description

By precompiling the reactions, you can run multiple SSA simulations repeatedly without having to recompile the reactions every time.

Usage

```
compile_reactions(
  reactions,
  state_ids,
  params,
  buffer_ids = NULL,
  hardcode_params = FALSE,
  fun_by = 10000L
)
```

Arguments

`reactions`	'reaction' A list of multiple `reaction()` objects.
`state_ids`	`[character]` The names of the states in the correct order.
`params`	`[named numeric]` Constants that are used in the propensity functions.
`buffer_ids`	`[character]` The order of any buffer calculations that are made as part of the propensity functions.
`hardcode_params`	`[logical]` Whether or not to hardcode the values of `params` in the compilation of the propensity functions. Setting this to `TRUE` will result in a minor sacrifice in accuracy for a minor increase in performance.
`fun_by`	`[integer]` Combine this number of propensity functions into one function.

Value

A list of objects solely to be used by `ssa()`.

- `x[["state_change"]]`: A sparse matrix of reaction effects.
- `x[["reaction_ids"]]`: The names of the reactions.
- `x[["buffer_ids"]]`: A set of buffer variables found in the propensity functions.
- `x[["buffer_size"]]`: The minimum size of the buffer required.
- `x[["function_pointers"]]`: A list of compiled propensity functions.
- `x[["hardcode_params"]]`: Whether the parameters were hard coded into the source code.‘

Examples

```
initial_state <- c(prey = 1000, predators = 1000)
params <- c(c1 = 10, c2 = 0.01, c3 = 10)
reactions <- list(
  #        propensity function     effects                        name for reaction
  reaction(~c1 * prey,             c(prey = +1),                  "prey_up"),
  reaction(~c2 * prey * predators, c(prey = -1, predators = +1), "predation"),
  reaction(~c3 * predators,        c(predators = -1),             "pred_down")
)

compiled_reactions <- compile_reactions(
  reactions = reactions,
  state_ids = names(initial_state),
  params = params
)

out <-
  ssa(
    initial_state = initial_state,
    reactions = compiled_reactions,
    params = params,
    method = ssa_exact(),
    final_time = 5,
    census_interval = .001,
    verbose = TRUE
  )

plot_ssa(out)
```

`GillespieSSA2` **GillespieSSA2**: *Gillespie's Stochastic Simulation Algorithm for impatient people.*

Description

GillespieSSA2 is a fast, scalable, and versatile framework for simulating large systems with Gillespie's Stochastic Simulation Algorithm (SSA). This package is the spiritual successor to the **GillespieSSA** package originally written by Mario Pineda-Krch.

Details

GillespieSSA2 has the following added benefits:

- The whole algorithm is run in Rcpp which results in major speed improvements (>100x). Even your propensity functions (reactions) are being compiled to Rcpp!
- Parameters and variables have been renamed to make them easier to understand.
- Many unit tests try to ensure that the code works as intended.

The SSA methods currently implemented are: Exact (`ssa_exact()`), Explicit tau-leaping (`ssa_etl()`), and the Binomial tau-leaping (`ssa_btl()`).

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 are single instantaneous events changing at least one of the populations (e.g. birth, death, movement, collision, predation, infection, etc). These 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 $\boldsymbol{\nu_j}$ and its propensity function $a_j(\mathbf{x})$. The state-change vector is defined as $\boldsymbol{\nu}_j \equiv (\nu_{1j}, \ldots, \nu_{Nj})$, where ν_{ij} is the change in the number of individuals in population i caused by one reaction of type j. The propensity function is defined as $a_j(\mathbf{x})$, where $a_j(\mathbf{x})dt$ is the probability that a particular reaction j will occur in the next infinitesimal time interval $[t, t + dt]$.

Contents of this package

- `ssa()`: The main entry point for running an SSA simulation.
- `plot_ssa()`: A standard visualisation for generating an overview plot fo the output.
- `ssa_exact()`, `ssa_etl()`, `ssa_btl()`: Different SSA algorithms.
- `ode_em()`: An ODE algorithm.
- `compile_reactions()`: A function for precompiling the reactions.

See Also

`ssa()` for more explanation on how to use **GillespieSSA2**

ode_em *Euler-Maruyama method (EM)*

Description

Euler-Maruyama method implementation of the ODE.

Usage

```
ode_em(tau = 0.01, noise_strength = 2)
```

Arguments

tau	tau parameter
noise_strength	noise_strength parameter

Value

an object of to be used by ssa().

plot_ssa *Simple plotting of ssa output*

Description

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

Usage

```
plot_ssa(
  ssa_out,
  state = TRUE,
  propensity = FALSE,
  buffer = FALSE,
  firings = FALSE,
  geom = c("point", "step")
)
```

Arguments

ssa_out	Data object returned by ssa().
state	Whether or not to plot the state values.
propensity	Whether or not to plot the propensity values.
buffer	Whether or not to plot the buffer values.
firings	Whether or not to plot the reaction firings values.
geom	Which geom to use, must be one of "point", "step".

port_reactions *Port GillespieSSA parameters to GillespieSSA2*

Description

This is a helper function to tranform GillesieSSA-style paramters to GillespieSSA2.

Usage

```
port_reactions(x0, a, nu)
```

Arguments

x0	The x0 parameter of GillespieSSA::ssa().
a	The a parameter of GillespieSSA::ssa().
nu	The nu parameter of GillespieSSA::ssa().

Value

A set of reaction()s to be used by ssa().

Examples

```
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)
port_reactions(x0, a, nu)
```

print.SSA_reaction *Print various SSA objects*

Description

Print various SSA objects

Usage

```
## S3 method for class 'SSA_reaction'
print(x, ...)

## S3 method for class 'SSA_method'
print(x, ...)
```

Arguments

x	An SSA reaction or SSA method
...	Not used

`reaction` *Define a reaction*

Description

During an SSA simulation, at any infinitesimal time interval, a reaction will occur with a probability defined according to its propensity. If it does, then it will change the state vector according to its effects.

Usage

```
reaction(propensity, effect, name = NA_character_)
```

Arguments

`propensity`	`[character/formula]` A character or formula representation of the propensity function, written in C++.
`effect`	`[named integer vector]` The change in state caused by this reaction.
`name`	`[character]` A name for this reaction (Optional). May only contain characters matching `[A-Za-z0-9_]`.

Details

It is possible to use 'buffer' values in order to speed up the computation of the propensity functions. For instance, instead of `"(c3 * s1) / (1 + c3 * c1)"`, it is possible to write `"buf = c3 * s1; buf / (buf + 1)"` instead.

Value

`[SSA_reaction]` This object describes a single reaction as part of an SSA simulation. It contains the following member values:

- `r[["propensity"]]`: The propensity function as a character.
- `r[["effect"]]`: The change in state caused by this reaction.
- `r[["name"]]`: The name of the reaction, `NA_character_` if no name was provided.

Examples

```
#        propensity                        effect
reaction(~ c1 * s1,                          c(s1 = -1))
reaction("c2 * s1 * s1",                     c(s1 = -2, s2 = +1))
reaction("buf = c3 * s1; buf / (buf + 1)",   c(s1 = +2))
```

ssa *Invoking the stochastic simulation algorithm*

Description

Main interface function to the implemented SSA methods. Runs a single realization of a predefined system. For a detailed explanation on how to set up your first SSA system, check the introduction vignette: `vignette("an_introduction", package = "GillespieSSA2")`. If you're transitioning from **GillespieSSA** to **GillespieSSA2**, check out the corresponding vignette: `vignette("converting_from_GillespieSSA package = "GillespieSSA2")`.

Usage

```
ssa(
  initial_state,
  reactions,
  final_time,
  params = NULL,
  method = ssa_exact(),
  census_interval = 0,
  stop_on_neg_state = TRUE,
  max_walltime = Inf,
  log_propensity = FALSE,
  log_firings = FALSE,
  log_buffer = FALSE,
  verbose = FALSE,
  console_interval = 1,
  sim_name = NA_character_,
  return_simulator = FALSE
)
```

Arguments

`initial_state`	`[named numeric vector]` The initial state to start the simulation with.
`reactions`	A list of reactions, see `reaction()`.
`final_time`	`[numeric]` The final simulation time.
`params`	`[named numeric vector]` Constant parameters to be used in the propensity functions.
`method`	`[ssa_method]]` Which SSA algorithm to use. Must be one of: `ssa_exact()`, `ssa_btl()`, or `ssa_etl()`.
`census_interval`	`[numeric]` The approximate interval between recording the state of the system. Setting this parameter to `0` will cause each state to be recorded, and to `Inf` will cause only the end state to be recorded.
`stop_on_neg_state`	`[logical]` Whether or not to stop the simulation when the a negative value in the state has occured. This can occur, for instance, in the `ssa_etl()` method.

`max_walltime` [numeric] The maximum duration (in seconds) that the simulation is allowed to run for before terminated.

`log_propensity` [logical] Whether or not to store the propensity values at each census.

`log_firings` [logical] Whether or not to store number of firings of each reaction between censuses.

`log_buffer` [logical] Whether or not to store the buffer at each census.

`verbose` [logical] If `TRUE`, intermediary information pertaining to the simulation will be displayed.

`console_interval` [numeric] The approximate interval between intermediary information outputs.

`sim_name` [character] An optional name for the simulation.

`return_simulator` Whether to return the simulator itself, instead of the output.

Details

Substantial improvements in speed and accuracy can be obtained by adjusting the additional (and optional) `ssa` arguments. By default `ssa` uses conservative parameters (o.a. `ssa_exact()`) which prioritise computational accuracy over computational speed.

Approximate methods (`ssa_etl()` and `ssa_btl()`) are not fool proof! Some tweaking might be required for a stochastic model to run appropriately.

Value

Returns a list containing the output of the simulation:

- `out[["time"]]`: [numeric] The simulation time at which a census was performed.
- `out[["state"]]`: [numeric matrix] The number of individuals at those time points.
- `out[["propensity"]]`: [numeric matrix] If `log_propensity` is `TRUE`, the propensity value of each reaction at each time point.
- `out[["firings"]]`: [numeric matrix] If `log_firings` is `TRUE`, the number of firings between two time points.
- `out[["buffer"]]`: [numeric matrix] If `log_buffer` is `TRUE`, the buffer values at each time point.
- `out[["stats"]]`: [data frame] Various stats:
  - `$method`: The name of the SSA method used.
  - `$sim_name`: The name of the simulation, if provided.
  - `$sim_time_exceeded`: Whether the simulation stopped because the final simulation time was reached.
  - `$all_zero_state`: Whether an extinction has occurred.
  - `$negative_state`: Whether a negative state has occurred. If an SSA method other than `ssa_etl()` is used, this indicates a mistake in the provided reaction effects.
  - `$all_zero_propensity`: Whether the simulation stopped because all propensity values are zero.
  - `$negative_propensity`: Whether a negative propensity value has occurred. If so, there is likely a mistake in the provided reaction propensity functions.

- `$walltime_exceeded`: Whether the simulation stopped because the maximum execution time has been reached.
- `$walltime_elapsed`: The duration of the simulation.
- `$num_steps`: The number of steps performed.
- `$dtime_mean`: The mean time increment per step.
- `$dtime_sd`: THe standard deviation of time increments.
- `$firings_mean`: The mean number of firings per step.
- `$firings_sd`: The standard deviation of the number of firings.

See Also

GillespieSSA2 for a high level explanation of the package

Examples

```
initial_state <- c(prey = 1000, predators = 1000)
params <- c(c1 = 10, c2 = 0.01, c3 = 10)
reactions <- list(
  #        propensity function     effects                        name for reaction
  reaction(~c1 * prey,             c(prey = +1),                  "prey_up"),
  reaction(~c2 * prey * predators, c(prey = -1, predators = +1), "predation"),
  reaction(~c3 * predators,        c(predators = -1),             "pred_down")
)

out <-
  ssa(
    initial_state = initial_state,
    reactions = reactions,
    params = params,
    method = ssa_exact(),
    final_time = 5,
    census_interval = .001,
    verbose = TRUE
  )

plot_ssa(out)
```

---

ssa_btl *Binomial tau-leap method (BTL)*

Description

Binomial tau-leap method implementation of the SSA as described by Chatterjee et al. (2005).

Usage

```
ssa_btl(mean_firings = 10)
```

Arguments

mean_firings	A coarse-graining factor of how many firings will occur at each iteration on average. Depending on the propensity functions, a value for mean_firings will result in warnings generated and a loss of accuracy.

Value

an object of to be used by ssa().

References

Chatterjee A., Vlachos D.G., and Katsoulakis M.A. 2005. Binomial distribution based tau-leap accelerated stochastic simulation. J. Chem. Phys. 122:024112. doi: 10.1063/1.1833357.

---

ssa_etl *Explicit tau-leap method (ETL)*

Description

Explicit tau-leap method implementation of the SSA as described by Gillespie (2001). Note that this method does not attempt to select an appropriate value for tau, nor does it implement estimated-midpoint technique.

Usage

```
ssa_etl(tau = 0.3)
```

Arguments

tau	the step-size (default 0.3).

Value

an object of to be used by ssa().

References

Gillespie D.T. 2001. Approximate accelerated stochastic simulation of chemically reacting systems. J. Chem. Phys. 115:1716-1733. doi: 10.1063/1.1378322.

`ssa_exact` *Exact method*

Description

Exact method implementation of the SSA as described by Gillespie (1977).

Usage

```
ssa_exact()
```

Value

an object of to be used by `ssa()`.

References

Gillespie D.T. 1977. Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. 81:2340. doi: 10.1021/j100540a008

Index