Package 'StochSimR'

June 1, 2026

Type Package

Title Stochastic Process Simulation Engine

Version 1.1.0

Description A modular simulation engine for a wide range of stochastic processes. Provides exact and approximate simulation methods for Poisson processes (homogeneous and inhomogeneous), Brownian motion (standard, drifted, and bridge), discrete- and continuous-time Markov chains, birth-death processes, the Yule pure-birth process, infinitesimal generator matrix utilities, Markovian queuing systems (M/M/1, M/M/c, M/M/c/K) with exact steady-state statistics, Levy processes (gamma, normal inverse Gaussian, variance-gamma, alpha-stable), Merton jump-diffusion models, Hawkes self-exciting processes, geometric Brownian motion, and Ornstein-Uhlenbeck mean-reverting diffusions. Includes variance reduction techniques (antithetic variates, control variates, importance sampling, stratified sampling), parallel simulation via the 'future' framework, rare-event simulation (cross-entropy and multilevel splitting), path visualisation, and summary statistics. Methods are based on Glasserman (2003) <doi:10.1007/978-0-387-21617-1>, Asmussen & Glynn (2007) <doi:10.1007/978-0-387-69033-9>, Norris (1997) <doi:10.1017/CBO9780511810633>, and Kleinrock (1975, ISBN:0471491101).

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.5.0)

Imports ggplot2 (>= 3.4.0), rlang (>= 1.0.0), stats, parallel, future (>= 1.25.0), future.apply (>= 1.10.0)

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/Ayush291202/StochSimR

BugReports https://github.com/Ayush291202/StochSimR/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Ayush Kundu [aut, cre] (ORCID: <https://orcid.org/0009-0009-8715-2624>)

Maintainer Ayush Kundu <ayushkundu25@iitk.ac.in>

Repository CRAN

Date/Publication 2026-06-01 08:50:13 UTC

Contents

compare_methods *Compare Simulation Methods*

Description

Runs the same stochastic process under different simulation methods (e.g., "exact" vs "euler") and compares bias, variance, and computation time.

Usage

```
compare_methods(
  sim_fn,
  methods,
  n_paths = 1000L,
  reference_fn = NULL,
  n_reps = 20L,
  ...
)
```

Arguments

sim_fn	A StochSimR simulation function that accepts a method argument.
methods	Character vector. Method names to compare.
n_paths	Positive integer. Paths per replication.
reference_fn	Optional function receiving the argument list and returning the exact expected terminal value, for bias computation.
n_reps	Positive integer. Independent replications per method.
...	Additional arguments to sim_fn.

Value

A data frame with one row per method and columns: method, mean_terminal, sd_terminal, bias, rmse, time_seconds.

Examples

```
# Compare exact vs Euler for OU (exact E[X_T] = mu for large T)
compare_methods(sim_ou, methods = c("exact", "euler"),
  n_paths = 200, n_reps = 10,
  T_max = 5, n_steps = 500, theta = 2, mu = 1, sigma = 0.5, x0 = 0)
```

`ctmc_generator` *Validate an Infinitesimal Generator Matrix*

Description

Checks that a matrix `Q` satisfies the three conditions for a valid infinitesimal generator (Q-matrix) of a continuous-time Markov chain: (i) off-diagonal entries are non-negative, (ii) row sums are zero, and (iii) `Q` is square. The validated matrix is returned invisibly, making it easy to use in a pipeline with `sim_ctmc`.

Usage

```
ctmc_generator(Q, tol = 1e-08)
```

Arguments

`Q`	Numeric square matrix. Candidate generator matrix with non-negative off-diagonal entries and zero row sums.
`tol`	Numeric. Absolute tolerance used when checking that row sums equal zero. Default `1e-8`.

Details

An infinitesimal generator Q satisfies:

- $Q_{ij} \geq 0$ for $i \neq j$.
- $Q_{ii} = -\sum_{j \neq i} Q_{ij}$ (row sums are zero).

Equivalently, one can specify only the off-diagonal rates and set the diagonal so that rows sum to zero. The holding time in state i is exponential with rate $q_i = -Q_{ii}$.

Value

The matrix `Q`, invisibly, after validation.

Examples

```
# Simple 3-state CTMC
Q <- matrix(c(-2,  1,  1,
               2, -3,  1,
               1,  2, -3), nrow = 3, byrow = TRUE)
ctmc_generator(Q)

# Build Q from off-diagonal rates (set diagonal automatically)
R <- matrix(c(0, 1, 1, 2, 0, 1, 1, 2, 0), nrow = 3, byrow = TRUE)
diag(R) <- -rowSums(R)
ctmc_generator(R)
```

`ctmc_stationary` *Compute the Stationary Distribution of a CTMC*

Description

Solves the global balance equations $\pi Q = 0$, $\sum_i \pi_i = 1$ to find the stationary (limiting) distribution of an ergodic continuous-time Markov chain.

Usage

```
ctmc_stationary(Q)
```

Arguments

`Q`	Numeric square generator matrix. Must pass `ctmc_generator`.

Details

The stationary distribution satisfies $\pi Q = 0$, which is equivalent to the linear system $Q^T \pi^T = 0$. The last equation is replaced by the normalisation constraint $\sum_i \pi_i = 1$, and the resulting square system is solved via `base::solve`. Negative values due to numerical error are clipped to zero before renormalisation.

If `Q` has absorbing states (rows of all zeros), the chain is not ergodic and the function returns a warning.

Value

A named numeric vector of length `nrow(Q)` with the stationary probabilities.

Examples

```
Q <- matrix(c(-3,  2,  1,
               1, -2,  1,
               2,  1, -3), nrow = 3, byrow = TRUE)
pi <- ctmc_stationary(Q)
print(pi)
stopifnot(abs(sum(pi) - 1) < 1e-10)
```

is_stoch_path *Test if an Object is a stoch_path*

Description

Test if an Object is a stoch_path

Usage

```
is_stoch_path(x)
```

Arguments

x	Any R object.

Value

TRUE if x inherits from class "stoch_path", FALSE otherwise.

Examples

```
sp <- new_stoch_path(0:5, 1:6, "test", "test")
is_stoch_path(sp)   # TRUE
is_stoch_path(42)   # FALSE
```

mc_estimate *Monte Carlo Estimator with Diagnostics*

Description

General-purpose Monte Carlo estimator that simulates paths from any StochSimR process, evaluates a target functional, and provides convergence diagnostics via batch means.

Usage

```
mc_estimate(sim_fn, h, n_paths = 10000L, batch_size = 500L, alpha = 0.05, ...)
```

Arguments

sim_fn	A StochSimR simulation function.
h	Functional to evaluate on paths. Receives the values matrix (rows = time, columns = paths) and returns a numeric vector of length n_paths.
n_paths	Positive integer. Total Monte Carlo replications.
batch_size	Positive integer. Size of each batch for batch-means standard error estimation.
alpha	Numeric in (0, 1). Significance level for confidence interval (default 0.05 gives 95% CI).
...	Additional arguments passed to sim_fn.

Value

A list with components:

estimate Numeric. Monte Carlo estimate of $E[h(X)]$.

se Numeric. Standard error.

ci Numeric vector of length 2. Confidence interval.

alpha Numeric. Significance level used.

n_samples Integer. Actual number of samples.

batch_means Numeric vector. Per-batch means.

batch_vars Numeric vector. Per-batch variances.

convergence Data frame with columns `n`, `running_mean`, and `running_se` for monitoring convergence.

method Character. `"monte_carlo"`.

Examples

```
# Estimate E[max(B_1, 0)] where B is standard Brownian motion
# Exact answer: 1/sqrt(2*pi) ~ 0.3989
h_pos <- function(vals) pmax(vals[nrow(vals), ], 0)
mc <- mc_estimate(sim_brownian, h = h_pos, n_paths = 10000,
                  T_max = 1, n_steps = 100)
cat("Estimate:", mc$estimate, "+/-", mc$se, "\\n")
```

`mc_rare_event` *Rare-Event Simulation*

Description

Estimates probabilities of rare events using either the cross-entropy method (adaptive importance sampling) or multilevel splitting.

Usage

```
mc_rare_event(
  sim_fn,
  indicator,
  score,
  n_paths = 5000L,
  n_iter = 5L,
  quantile_level = 0.9,
  method = c("cross_entropy", "splitting"),
  ...
)
```

Arguments

`sim_fn`	A StochSimR simulation function.
`indicator`	Indicator function for the rare event. Receives the values matrix and returns a 0/1 numeric vector of length `n_paths`.
`score`	Continuous scoring function used for adaptive thresholding. Higher scores indicate proximity to the rare event. Receives the values matrix and returns a numeric vector. Required for both methods.
`n_paths`	Positive integer. Paths per iteration.
`n_iter`	Positive integer. Number of adaptive iterations.
`quantile_level`	Numeric in (0, 1). Quantile for threshold selection in each iteration (default 0.9).
`method`	Character. `"cross_entropy"` (default) or `"splitting"`.
`...`	Additional arguments to `sim_fn`.

Details

The cross-entropy method iteratively updates a drift parameter to maximise the likelihood of the rare event among elite samples (those exceeding the `quantile_level` quantile of the score function).

Multilevel splitting maintains a population of particles, discarding those below a threshold and resampling (branching) from survivors. The probability is estimated as the product of survival fractions across levels.

Value

A list with components:

probability Numeric. Estimated rare-event probability.

se Numeric. Standard error (cross-entropy only).

ci Numeric vector of length 2. 95% confidence interval (cross-entropy only).

log10_probability Numeric. Log-10 of the estimated probability.

diagnostics List. Per-iteration diagnostic information.

method Character. Method used.

n_total_samples Integer. Total paths simulated.

References

Rubinstein, R. Y. & Kroese, D. P. (2004). *The Cross-Entropy Method*. Springer.

Cerou, F. & Guyader, A. (2007). Adaptive multilevel splitting for rare event analysis. *Stochastic Analysis and Applications*, 25(2), 417–443.

Examples

```
# Estimate P(S_T > 200) for GBM
indicator_fn <- function(vals) as.numeric(vals[nrow(vals), ] > 200)
score_fn <- function(vals) vals[nrow(vals), ]
result <- mc_rare_event(sim_gbm, indicator = indicator_fn,
  score = score_fn, n_paths = 5000, n_iter = 5,
  T_max = 1, n_steps = 252, mu = 0.05, sigma = 0.2, x0 = 100)
cat("P(S_T > 200):", result$probability, "\\n")
```

---

new_stoch_path *Create a stoch_path Object*

Description

Constructor for the core path object returned by all simulators in **StochSimR**. Users rarely need to call this directly.

Usage

```
new_stoch_path(times, values, process, method, params = list())
```

Arguments

`times`	Numeric vector of time points (length `n`).
`values`	Numeric vector or matrix of path values. If a vector, it is treated as a single path. If a matrix, each column represents one sample path and must have `n` rows matching `length(times)`.
`process`	Character string naming the stochastic process (e.g., `"Brownian Motion"`).
`method`	Character string naming the simulation algorithm (e.g., `"exact"`, `"euler"`).
`params`	Named list of process parameters used in the simulation.

Value

An S3 object of class `"stoch_path"` with components:

times Numeric vector of time points.

values Numeric matrix of path values (rows = time, cols = paths).

process Character string naming the process.

method Character string naming the simulation method.

params Named list of parameters.

n_paths Integer, number of sample paths.

n_steps Integer, number of time points.

Examples

```
sp <- new_stoch_path(
  times = seq(0, 1, length.out = 11),
  values = cumsum(rnorm(11)),
  process = "Custom Process",
  method = "manual"
)
sp
```

`path_summary` *Path Summary Statistics*

Description

Computes comprehensive summary statistics across all simulated paths, including terminal value distribution, extremes, and increment properties.

Usage

```
path_summary(x)
```

Arguments

x	A `new_stoch_path` object.

Value

A data frame with two columns: `statistic` (character) and `value` (numeric). Statistics include number of paths and steps, terminal value moments and quantiles, path extremes, and increment statistics.

Examples

```
paths <- sim_brownian(T_max = 1, n_steps = 500, n_paths = 1000)
path_summary(paths)
```

plot_acf_paths *Plot Autocorrelation of Path Increments*

Description

Displays the sample autocorrelation function (ACF) of path increments, useful for verifying independence structure or detecting serial dependence.

Usage

```
plot_acf_paths(x, path_idx = 1L, lag.max = 40L, title = NULL)
```

Arguments

x	A new_stoch_path object.
path_idx	Positive integer. Which path to compute ACF for (default 1).
lag.max	Positive integer. Maximum lag to display.
title	Character or NULL. Custom title.

Value

A ggplot object.

Examples

```
paths <- sim_ou(T_max = 10, n_steps = 1000, theta = 2, mu = 0,
                sigma = 1, x0 = 5, n_paths = 1)
plot_acf_paths(paths)
```

plot_distribution *Plot Terminal Value Distribution*

Description

Histogram with kernel density overlay of the terminal values of simulated paths.

Usage

```
plot_distribution(x, bins = 50L, title = NULL)
```

Arguments

x	A new_stoch_path object.
bins	Positive integer. Number of histogram bins.
title	Character or NULL. Custom title.

Value

A ggplot object.

Examples

```
paths <- sim_gbm(T_max = 1, n_steps = 252, mu = 0.05, sigma = 0.3,
                 x0 = 100, n_paths = 5000)
plot_distribution(paths)
```

---

plot_paths *Plot Sample Paths*

Description

Creates a **ggplot2** visualisation of simulated sample paths with optional sample mean overlay and pointwise confidence bands.

Usage

```
plot_paths(
  x,
  max_paths = 100L,
  show_mean = TRUE,
  show_bands = TRUE,
  alpha_line = 0.3,
  title = NULL
)
```

Arguments

x	A new_stoch_path object.
max_paths	Positive integer. Maximum paths to plot (subsampled randomly if x$n_paths > max_paths).
show_mean	Logical. If TRUE (default), overlay the sample mean path in red.
show_bands	Logical. If TRUE (default), show pointwise 2.5% and 97.5% quantile bands.
alpha_line	Numeric in (0, 1]. Transparency for individual paths.
title	Character or NULL. Custom plot title. If NULL, a default title is generated from the process name.

Value

A ggplot object.

Examples

```
paths <- sim_brownian(T_max = 1, n_steps = 500, n_paths = 100)
plot_paths(paths, max_paths = 50, show_mean = TRUE, show_bands = TRUE)
```

`queue_stats` *Exact Steady-State Statistics for Markovian Queues*

Description

Computes exact theoretical steady-state performance measures for M/M/1, M/M/c, and M/M/c/K queues using closed-form or global-balance solutions.

Usage

```
queue_stats(lambda, mu, servers = 1L, capacity = Inf)
```

Arguments

`lambda`	Positive numeric. Arrival rate.
`mu`	Positive numeric. Per-server service rate.
`servers`	Positive integer. Number of servers c.
`capacity`	Either a positive integer K or `Inf`.

Details

M/M/1 (`servers` = 1, `capacity` = `Inf`):

$$L = \frac{\rho}{1-\rho}, \quad L_q = \frac{\rho^2}{1-\rho}, \quad W = \frac{1}{\mu-\lambda}, \quad W_q = \frac{\rho}{\mu-\lambda}.$$

M/M/c (`servers` = `c`, `capacity` = `Inf`): Uses the Erlang C formula for $P(\text{wait})$.

M/M/c/K: Uses global balance to obtain the exact stationary distribution π_n, $n = 0, \ldots, K$.

Value

A named list with components:

`type` Character. Queue type (e.g.\ `"M/M/1"`).

`rho` Numeric. Traffic intensity $\lambda/(c\mu)$.

`stable` Logical. Whether the queue is stable (infinite-capacity only).

`L` Numeric. Mean number of customers in the system (Little).

`Lq` Numeric. Mean number of customers in the queue.

`W` Numeric. Mean sojourn time in system (Little).

`Wq` Numeric. Mean waiting time in queue.

`p0` Numeric. Steady-state probability that the system is empty.

`erlang_c` Numeric (M/M/c only). Erlang C probability – the probability that an arriving customer waits.

`p_block` Numeric (finite capacity only). Blocking probability $P(N = K)$.

`lambda_eff` Numeric (finite capacity only). Effective throughput $\lambda(1 - P(N = K))$.

`pi` Numeric vector (finite capacity only). Full stationary distribution $\pi_0, \ldots, \pi_K$.

References

Kleinrock, L. (1975). *Queueing Systems, Volume 1: Theory*. Wiley.

Gross, D. & Harris, C. M. (1998). *Fundamentals of Queueing Theory*, 3rd ed. Wiley.

Examples

```
# M/M/1 queue
s <- queue_stats(lambda = 2, mu = 3)
cat("E[L] =", s$L, "  E[W] =", s$W, "\\n")

# M/M/2 queue
s2 <- queue_stats(lambda = 4, mu = 3, servers = 2)
cat("Erlang C =", s2$erlang_c, "\\n")

# M/M/1/5 (finite capacity)
sk <- queue_stats(lambda = 5, mu = 3, capacity = 5)
cat("P(block) =", sk$p_block, "\\n")
```

`sim_birth_death` *Simulate a Birth-Death Process*

Description

Simulates exact sample paths of a continuous-time birth-death process using the Gillespie algorithm. Birth and death rates may be constant (scalar), state-dependent (vector), or arbitrary functions of the current state.

Usage

```
sim_birth_death(
  lambda,
  mu,
  n_states = 20L,
  T_max = 10,
  x0 = 0L,
  n_paths = 1L,
  n_grid = NULL
)
```

Arguments

`lambda`	Birth rates. One of: • A single positive number: constant birth rate λ for all states. • A numeric vector of length `n_states`: `lambda[i+1]` is the birth rate from state `i` (0-indexed states). • A function `lambda(n)` returning the birth rate from state `n`.
`mu`	Death rates. Same format as `lambda`. The death rate from state 0 is always forced to 0 (cannot go negative).
`n_states`	Positive integer. Number of states: $\{0, 1, \ldots, n_{\text{states}} - 1\}$. Acts as a reflecting upper boundary.
`T_max`	Positive numeric. Simulation end time.
`x0`	Non-negative integer. Initial state (0-indexed). Default 0.
`n_paths`	Positive integer. Number of independent paths.
`n_grid`	Positive integer. Grid resolution. Defaults to an adaptive value based on `T_max`.

Details

A birth-death process is a CTMC on $\{0, 1, \ldots\}$ where the only transitions are $n \to n+1$ (birth, rate λ_n) and $n \to n-1$ (death, rate μ_n), with $\mu_0 = 0$. The simulation uses the direct Gillespie method: from state n, a holding time $\text{Exp}(\lambda_n + \mu_n)$ is drawn, then birth or death is chosen with probability proportional to the respective rate.

The stationary distribution (when it exists) satisfies the detailed balance equations:

$$\pi_n = \pi_0 \prod_{k=0}^{n-1} \frac{\lambda_k}{\mu_{k+1}}.$$

Value

A `new_stoch_path` object with non-negative integer values representing the population count at each time point.

References

Karlin, S. & Taylor, H. M. (1975). *A First Course in Stochastic Processes*, 2nd ed. Academic Press. Chapter 4.

Examples

```
# Linear birth-death: lambda = 2, mu = 1, start at 5
paths <- sim_birth_death(lambda = 2, mu = 1, n_states = 30,
                         T_max = 5, x0 = 5, n_paths = 20)
plot_paths(paths)

# M/M/1 queue as a birth-death process
paths_mm1 <- sim_birth_death(lambda = 2, mu = 3, n_states = 20,
                             T_max = 10, x0 = 0, n_paths = 10)
```

```
plot_paths(paths_mm1)

# State-dependent birth rate: lambda_n = sqrt(n+1)
paths_sd <- sim_birth_death(
  lambda = function(n) sqrt(n + 1),
  mu = 1,
  n_states = 25, T_max = 5, x0 = 0, n_paths = 10
)
plot_paths(paths_sd)
```

`sim_brownian` *Simulate Brownian Motion*

Description

Generates sample paths of standard or drifted Brownian motion (Wiener process).

Usage

```
sim_brownian(
  T_max = 1,
  n_steps = 1000L,
  mu = 0,
  sigma = 1,
  x0 = 0,
  n_paths = 1L,
  method = c("exact", "bridge")
)
```

Arguments

`T_max`	Positive numeric. End time.
`n_steps`	Positive integer. Number of time steps.
`mu`	Numeric. Drift coefficient (default 0).
`sigma`	Positive numeric. Volatility / diffusion coefficient (default 1).
`x0`	Numeric. Starting value (default 0).
`n_paths`	Positive integer. Number of independent paths.
`method`	Character. `"exact"` (default) uses cumulative normal increments; `"bridge"` uses Brownian bridge interpolation on a coarse skeleton.

Details

With method `"exact"`, increments are drawn as $X_{i+1} - X_i \sim N(\mu\Delta t,\ \sigma^2\Delta t)$.

With method `"bridge"`, a coarse skeleton is first simulated exactly, then intermediate points are filled using the Brownian bridge conditional distribution.

Value

A new_stoch_path object.

Examples

```
paths <- sim_brownian(T_max = 1, n_steps = 1000, n_paths = 50)
plot_paths(paths, show_mean = TRUE, show_bands = TRUE)

# Drifted Brownian motion
paths_drift <- sim_brownian(T_max = 2, n_steps = 500, mu = 1, sigma = 0.5,
                            n_paths = 30)
plot_paths(paths_drift)
```

---

sim_ctmc *Simulate a Continuous-Time Markov Chain*

Description

Generates sample paths of a CTMC specified by an infinitesimal generator matrix, using the exact Gillespie (next-reaction) algorithm.

Usage

```
sim_ctmc(Q, T_max = 1, x0 = 1L, n_paths = 1L, n_grid = NULL)
```

Arguments

Q	Numeric square generator matrix. Must pass ctmc_generator.
T_max	Positive numeric. Simulation end time.
x0	Positive integer. Initial state (1-indexed). Defaults to 1.
n_paths	Positive integer. Number of independent paths.
n_grid	Positive integer. Number of equally spaced time points at which the piecewise-constant path is evaluated. Default adapts to T_max.

Details

At each event, the holding time in the current state i is drawn as $\mathrm{Exp}(q_i)$, where $q_i = -Q_{ii}$. The next state $j \neq i$ is chosen with probability Q_{ij}/q_i. The process stops when the next event time exceeds T_max. Paths are returned as step functions evaluated on a regular grid.

Value

A new_stoch_path object with integer-valued paths (state indices, 1-indexed).

References

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25), 2340–2361.

Norris, J. R. (1997). *Markov Chains*. Cambridge University Press.

Examples

```
Q <- matrix(c(-2, 1, 1,
              2, -3, 1,
              1,  2, -3), nrow = 3, byrow = TRUE)
paths <- sim_ctmc(Q, T_max = 10, x0 = 1, n_paths = 20)
plot_paths(paths)

# Check convergence to stationary distribution
pi_theory <- ctmc_stationary(Q)
paths_long <- sim_ctmc(Q, T_max = 1000, x0 = 1, n_paths = 1, n_grid = 10000)
print(pi_theory)
```

`sim_gbm` *Simulate Geometric Brownian Motion*

Description

Simulates paths of the geometric Brownian motion (GBM, Black–Scholes model). The exact method uses the log-normal distribution; the Euler method uses Euler–Maruyama discretisation.

Usage

```
sim_gbm(
  T_max = 1,
  n_steps = 252L,
  mu = 0.05,
  sigma = 0.2,
  x0 = 100,
  n_paths = 1L,
  method = c("exact", "euler")
)
```

Arguments

`T_max`	Positive numeric. End time.
`n_steps`	Positive integer. Number of steps.
`mu`	Numeric. Drift rate.
`sigma`	Positive numeric. Volatility.
`x0`	Positive numeric. Initial value (default 100).
`n_paths`	Positive integer. Number of paths.
`method`	Character. `"exact"` (log-normal, default) or `"euler"` (Euler–Maruyama).

Details

The SDE is $dS = \mu S\,dt + \sigma S\,dW$. With method `"exact"`, the solution $S_t = S_0 \exp\big((\mu - \sigma^2/2)t + \sigma W_t\big)$ is used directly.

Value

A `new_stoch_path` object with positive values.

Examples

```
paths <- sim_gbm(T_max = 1, n_steps = 252, mu = 0.08, sigma = 0.25,
                 x0 = 100, n_paths = 50)
plot_paths(paths)
plot_distribution(paths)
```

`sim_hawkes` *Simulate a Hawkes (Self-Exciting) Process*

Description

Simulates a univariate Hawkes process with exponential kernel $\phi(t) = \alpha e^{-\beta t}$ using Ogata's modified thinning algorithm.

Usage

```
sim_hawkes(T_max = 10, mu = 0.5, alpha = 0.8, beta = 1, n_paths = 1L)
```

Arguments

`T_max`	Positive numeric. End time.
`mu`	Positive numeric. Baseline intensity.
`alpha`	Positive numeric. Excitation magnitude.
`beta`	Positive numeric. Exponential decay rate. The process is stationary when `alpha` < `beta` (branching ratio < 1).
`n_paths`	Positive integer. Number of independent paths.

Details

The conditional intensity is

$$\lambda^*(t) = \mu + \alpha \sum_{t_i < t} e^{-\beta(t - t_i)}.$$

Simulation uses Ogata's thinning with an adaptive upper bound that is updated after each event.

Value

A `new_stoch_path` object (counting process on a regular grid).

References

Ogata, Y. (1981). On Lewis' simulation method for point processes. *IEEE Transactions on Information Theory*, 27(1), 23–31.

Hawkes, A. G. (1971). Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1), 83–90.

Examples

```
paths <- sim_hawkes(T_max = 50, mu = 0.5, alpha = 0.8, beta = 1.2,
                    n_paths = 10)
plot_paths(paths)
```

`sim_jump_diffusion` *Simulate a Jump-Diffusion Process (Merton Model)*

Description

Simulates paths from the Merton (1976) jump-diffusion model, which combines geometric Brownian motion with compound Poisson jumps in log-price.

Usage

```
sim_jump_diffusion(
  T_max = 1,
  n_steps = 1000L,
  mu = 0.05,
  sigma = 0.2,
  lambda = 1,
  mu_j = 0,
  sigma_j = 0.1,
  x0 = 1,
  n_paths = 1L,
  log_scale = FALSE
)
```

Arguments

`T_max`	Positive numeric. End time.
`n_steps`	Positive integer. Number of time steps.
`mu`	Numeric. Drift of the diffusion component.
`sigma`	Positive numeric. Volatility of the diffusion component.
`lambda`	Non-negative numeric. Jump intensity (mean jumps per unit time).
`mu_j`	Numeric. Mean of the log-jump size distribution.
`sigma_j`	Positive numeric. Standard deviation of the log-jump size distribution.

x0	Positive numeric. Initial value (default 1).
n_paths	Positive integer. Number of paths.
log_scale	Logical. If TRUE, return log-prices instead of prices. Default FALSE.

Details

The log-price dynamics are

$$d \log S = (\mu - \tfrac{1}{2}\sigma^2 - \lambda k)\, dt + \sigma\, dW + J\, dN$$

where N is a Poisson process with rate λ, jump sizes $J \sim N(\mu_J, \sigma_J^2)$, and $k = e^{\mu_J + \sigma_J^2/2} - 1$ is the compensator.

Value

A new_stoch_path object. Values are prices (or log-prices if log_scale = TRUE).

References

Merton, R. C. (1976). Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3(1-2), 125–144.

Examples

```
paths <- sim_jump_diffusion(
  T_max = 1, n_steps = 1000, mu = 0.05, sigma = 0.2,
  lambda = 3, mu_j = -0.02, sigma_j = 0.1, n_paths = 30
)
plot_paths(paths)
```

---

sim_levy *Simulate a Levy Process*

Description

Simulates sample paths from a Levy process. Four families are supported: alpha-stable, gamma subordinator, normal inverse Gaussian (NIG), and variance-gamma.

Usage

```
sim_levy(
  T_max = 1,
  n_steps = 500L,
  type = c("stable", "gamma", "nig", "variance_gamma"),
  alpha = 1.5,
  beta = 0,
  shape = 1,
```

```
  rate = 1,
  theta = 0,
  sigma = 1,
  nu = 0.5,
  n_paths = 1L
)
```

Arguments

T_max	Positive numeric. End time.
n_steps	Positive integer. Number of time steps.
type	Character. One of "stable", "gamma", "nig", or "variance_gamma".
alpha	Numeric in $(0, 2]$. Stability index for "stable".
beta	Numeric in $[-1, 1]$. Skewness for "stable".
shape	Positive numeric. Shape parameter for "gamma" and "nig".
rate	Positive numeric. Rate parameter for "gamma".
theta	Numeric. Drift for "nig" and "variance_gamma".
sigma	Positive numeric. Volatility for "nig" and "variance_gamma".
nu	Positive numeric. Variance rate for "variance_gamma".
n_paths	Positive integer. Number of paths.

Details

"stable" Uses the Chambers–Mallows–Stuck algorithm to generate alpha-stable increments.

"gamma" Gamma subordinator with shape shape * dt and rate rate per increment.

"nig" Normal inverse Gaussian: Brownian motion subordinated by an inverse Gaussian process.

"variance_gamma" Brownian motion subordinated by a gamma process with drift theta.

Value

A new_stoch_path object.

References

Chambers, J. M., Mallows, C. L. & Stuck, B. W. (1976). A method for simulating stable random variables. *Journal of the American Statistical Association*, 71(354), 340–344.

Examples

```
# Gamma subordinator (non-decreasing)
paths <- sim_levy(T_max = 5, n_steps = 500, type = "gamma",
                  shape = 2, rate = 1, n_paths = 20)
plot_paths(paths)

# Variance-Gamma
paths_vg <- sim_levy(T_max = 1, n_steps = 500, type = "variance_gamma",
```

```
                      theta = -0.1, sigma = 0.3, nu = 0.5, n_paths = 15)
plot_paths(paths_vg)
```

`sim_markov` *Simulate a Discrete-Time Markov Chain*

Description

Simulates sample paths from a discrete-time Markov chain with a given transition probability matrix.

Usage

```
sim_markov(P, n_steps = 100L, x0 = NULL, n_paths = 1L, states = NULL)
```

Arguments

`P`	Numeric matrix. Transition probability matrix where `P[i, j]` is the probability of moving from state `i` to state `j`. Must be square with non-negative entries and rows summing to 1.
`n_steps`	Positive integer. Number of transitions.
`x0`	Integer. Initial state (1-indexed). If `NULL` (default), sampled uniformly from all states.
`n_paths`	Positive integer. Number of independent chains.
`states`	Optional character vector of state names (length must equal `nrow(P)`).

Details

At each step the next state is sampled from the categorical distribution defined by the current row of `P`, using inverse-CDF sampling on precomputed cumulative row probabilities.

Value

A `new_stoch_path` object with integer-valued paths.

Examples

```
# Simple 3-state chain
P <- matrix(c(0.7, 0.2, 0.1,
              0.3, 0.4, 0.3,
              0.2, 0.3, 0.5), nrow = 3, byrow = TRUE)
paths <- sim_markov(P, n_steps = 200, n_paths = 10)
plot_paths(paths)

# Check convergence to stationary distribution
long_chain <- sim_markov(P, n_steps = 10000, x0 = 1, n_paths = 1)
table(long_chain$values) / length(long_chain$values)
```

`sim_ou` *Simulate an Ornstein–Uhlenbeck Process*

Description

Simulates paths from a mean-reverting Ornstein–Uhlenbeck (OU) diffusion $dX_t = \theta(\mu - X_t)\,dt + \sigma\,dW_t$.

Usage

```
sim_ou(
  T_max = 10,
  n_steps = 1000L,
  theta = 1,
  mu = 0,
  sigma = 1,
  x0 = 0,
  n_paths = 1L,
  method = c("exact", "euler")
)
```

Arguments

`T_max`	Positive numeric. End time.
`n_steps`	Positive integer. Number of steps.
`theta`	Positive numeric. Speed of mean reversion.
`mu`	Numeric. Long-run mean level.
`sigma`	Positive numeric. Volatility.
`x0`	Numeric. Initial value.
`n_paths`	Positive integer. Number of paths.
`method`	Character. `"exact"` (transition density, default) or `"euler"` (Euler–Maruyama).

Details

With method `"exact"`, the conditional distribution $X_{t+\Delta t}|X_t$ is Gaussian with known mean and variance, yielding an exact (discretisation-error-free) simulation. The Euler method introduces $O(\Delta t)$ weak error.

Value

A `new_stoch_path` object.

Examples

```
paths <- sim_ou(T_max = 10, n_steps = 1000, theta = 2, mu = 5,
                sigma = 0.5, x0 = 0, n_paths = 30)
plot_paths(paths, show_mean = TRUE, show_bands = TRUE)
```

`sim_parallel` *Parallel Simulation of Stochastic Processes*

Description

Distributes path simulation across multiple cores using the **future** framework for cross-platform parallelism.

Usage

```
sim_parallel(
  sim_fn,
  n_paths = 1000L,
  n_workers = NULL,
  chunk_size = NULL,
  seed = 42L,
  ...
)
```

Arguments

`sim_fn`	Any StochSimR simulation function.
`n_paths`	Positive integer. Total number of paths to simulate.
`n_workers`	Positive integer. Number of parallel workers. Default `NULL` uses `parallel::detectCores() - 1` (at least 1).
`chunk_size`	Positive integer. Number of paths per chunk. Default `NULL` splits evenly across workers.
`seed`	Integer. Base random seed for reproducibility. Worker `i` receives seed `seed + i`.
`...`	Additional arguments passed to `sim_fn`.

Details

The function temporarily sets `future::plan(multisession)` for parallel execution and restores the previous plan on exit. Each chunk is simulated independently with a deterministic seed derived from `seed`.

Value

A `new_stoch_path` object combining all paths.

Examples

```
paths <- sim_parallel(sim_gbm, n_paths = 1000, n_workers = 2,
  T_max = 1, n_steps = 252, mu = 0.05, sigma = 0.2, x0 = 100)
plot_paths(paths)
```

`sim_poisson` *Simulate a Poisson Process*

Description

Generates sample paths of a homogeneous or inhomogeneous Poisson process. Homogeneous processes use exact inter-arrival time simulation; inhomogeneous processes use the Lewis–Shedler thinning algorithm.

Usage

```
sim_poisson(
  T_max = 1,
  lambda = 1,
  lambda_max = NULL,
  n_paths = 1L,
  method = c("auto", "exact", "thinning")
)
```

Arguments

`T_max`	Positive numeric. End time of the simulation.
`lambda`	Rate parameter. Either a single positive number (homogeneous case) or a function of time `lambda(t)` returning the instantaneous rate (inhomogeneous case).
`lambda_max`	Positive numeric. Upper bound on `lambda(t)`, required when `lambda` is a function.
`n_paths`	Positive integer. Number of independent paths to simulate.
`method`	Character string. One of `"auto"` (default), `"exact"`, or `"thinning"`. `"auto"` selects `"exact"` for homogeneous and `"thinning"` for inhomogeneous processes.

Details

For the homogeneous case with rate λ, inter-arrival times are i.i.d. $\mathrm{Exp}(\lambda)$.

For the inhomogeneous case with rate function $\lambda(t)$, the Lewis–Shedler thinning algorithm generates candidates at rate `lambda_max` and accepts each with probability $\lambda(t)/\lambda_{\max}$.

Value

A `new_stoch_path` object containing the counting process evaluated on a regular time grid.

References

Lewis, P. A. W. & Shedler, G. S. (1979). Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3), 403–413.

Examples

```
# Homogeneous Poisson with rate 5
paths <- sim_poisson(T_max = 10, lambda = 5, n_paths = 20)
plot_paths(paths)

# Inhomogeneous Poisson with sinusoidal rate
paths_ih <- sim_poisson(
  T_max = 10,
  lambda = function(t) 3 + 2 * sin(2 * pi * t / 5),
  lambda_max = 5,
  n_paths = 10
)
plot_paths(paths_ih)
```

`sim_queue` *Simulate a Markovian Queuing Process*

Description

Simulates the queue-length process (number of customers in the system) of M/M/1, M/M/c, or M/M/1/K queues using the exact event-driven Gillespie algorithm. All three are birth-death CTMCs.

Usage

```
sim_queue(
  lambda,
  mu,
  servers = 1L,
  capacity = Inf,
  T_max = 10,
  x0 = 0L,
  n_paths = 1L,
  n_grid = NULL
)
```

Arguments

`lambda`	Positive numeric. Customer arrival rate (Poisson process).
`mu`	Positive numeric. Per-server service rate (exponential service times).
`servers`	Positive integer. Number of servers c. Default 1 (M/M/1 queue).

`capacity`	Either a positive integer K (maximum customers in the system, M/M/c/K) or `Inf` (no capacity limit, default).
`T_max`	Positive numeric. Simulation end time.
`x0`	Non-negative integer. Initial queue length (number in system). Default 0.
`n_paths`	Positive integer. Number of independent replications.
`n_grid`	Positive integer. Number of equally spaced time points for the returned path. Defaults to an adaptive value.

Details

The queue-length process $N(t)$ is a CTMC with:

- Arrival rate: $\lambda_n = \lambda$ for $n < K$ (0 at capacity for finite K).
- Service rate: $\mu_n = \min(n, c)\,\mu$ for $n \geq 1$.

The M/M/1 queue is stable when $\rho = \lambda/\mu < 1$. For M/M/c, stability requires $\rho = \lambda/(c\mu) < 1$. Finite-capacity queues are always stable. Theoretical steady-state values are available via `queue_stats`.

Value

A `new_stoch_path` object with non-negative integer values recording the queue length (number in system) over time. The `process` field names the queue type (e.g.\ `"M/M/1 Queue"`).

References

Kleinrock, L. (1975). *Queueing Systems, Volume 1: Theory*. Wiley.

Gross, D. & Harris, C. M. (1998). *Fundamentals of Queueing Theory*, 3rd ed. Wiley.

Examples

```
# M/M/1 queue: arrival rate 2, service rate 3
paths <- sim_queue(lambda = 2, mu = 3, T_max = 20, n_paths = 5)
plot_paths(paths)

# M/M/2 queue (two servers)
paths2 <- sim_queue(lambda = 4, mu = 3, servers = 2, T_max = 20, n_paths = 5)
plot_paths(paths2)

# M/M/1/5 queue (capacity 5)
paths_k <- sim_queue(lambda = 3, mu = 2, capacity = 5,
                     T_max = 20, n_paths = 10)
plot_paths(paths_k)
```

`sim_yule` *Simulate a Yule (Pure-Birth) Process*

Description

Simulates exact sample paths of a Yule process, also known as the Yule–Furry process or linear birth process. Each individual in the population gives birth independently at rate λ, giving a total birth rate of $n\lambda$ when the population is n.

Usage

```
sim_yule(lambda, T_max = 10, x0 = 1L, n_paths = 1L, n_grid = NULL)
```

Arguments

`lambda`	Positive numeric. Per-individual birth rate.
`T_max`	Positive numeric. Simulation end time.
`x0`	Positive integer. Initial population size (default 1).
`n_paths`	Positive integer. Number of independent paths.
`n_grid`	Positive integer. Grid resolution.

Details

The Yule process is a special case of a birth-death process with $\lambda_n = n\lambda$ and $\mu_n = 0$. Starting from $X(0) = x_0$, the mean and variance are:

$$E[X(t)] = x_0 e^{\lambda t}, \quad \mathrm{Var}[X(t)] = x_0 e^{\lambda t}(e^{\lambda t} - 1).$$

The marginal distribution of $X(t)$ given $X(0) = 1$ is geometric with success probability $e^{-\lambda t}$.

Value

A `new_stoch_path` object with positive integer values. The object's `params` list includes `mean_at_Tmax`, the theoretical mean $x_0 e^{\lambda T}$.

References

Yule, G. U. (1924). A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F.R.S. *Philosophical Transactions of the Royal Society B*, 213, 21–87.

Bailey, N. T. J. (1964). *The Elements of Stochastic Processes with Applications to the Natural Sciences*. Wiley.

Examples

```
paths <- sim_yule(lambda = 0.5, T_max = 6, x0 = 1, n_paths = 30)
plot_paths(paths)

# Theoretical mean overlay
t_grid <- paths$times
E_t <- exp(0.5 * t_grid)   # x0 = 1
cat("Simulated mean at T:", mean(paths$values[nrow(paths$values), ]),
    " Theoretical:", E_t[length(E_t)], "\\n")
```

`vr_antithetic` *Antithetic Variates Estimator*

Description

Reduces variance by pairing each simulation with its "mirror" simulation constructed from negated uniform random numbers. This induces negative correlation between the pair, lowering the variance of the average.

Usage

```
vr_antithetic(sim_fn, h = NULL, n_paths = 1000L, ...)
```

Arguments

`sim_fn`	A StochSimR simulation function (e.g., `sim_gbm`).
`h`	Function applied to the path values matrix, returning a numeric vector of length `n_paths`. Default: terminal value of each path.
`n_paths`	Positive integer. Number of antithetic pairs. The total number of function evaluations is `2 * n_paths`.
`...`	Additional arguments passed to `sim_fn`.

Details

The implementation works by simulating `n_paths` paths with a fixed random seed, then re-simulating with the same seed. For GBM and other processes driven by Brownian motion, the antithetic path is constructed by reflecting the log-returns around the drift, which is equivalent to negating the Brownian increments.

Value

A list with components:

estimate Numeric. Antithetic estimator of $E[h(X)]$.

se Numeric. Standard error of the antithetic estimator.

ci Numeric vector of length 2. 95% confidence interval.

raw_estimate Numeric. Crude Monte Carlo estimate (no VR).

raw_se Numeric. Standard error of the crude estimator.

reduction_factor Numeric. Variance reduction ratio (crude variance / antithetic variance).

method Character. `"antithetic_variates"`.

n_samples Integer. Total samples used.

References

Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*. Springer. Chapter 4.

Examples

```
h_call <- function(vals) pmax(vals[nrow(vals), ] - 100, 0)
result <- vr_antithetic(sim_gbm, h = h_call, n_paths = 5000,
  T_max = 1, n_steps = 252, mu = 0.05, sigma = 0.2, x0 = 100)
cat("Estimate:", result$estimate, " SE:", result$se,
    " Reduction:", round(result$reduction_factor, 2), "x\\n")
```

---

vr_control_variate *Control Variate Estimator*

Description

Uses a correlated control variate with known expectation to reduce the variance of a Monte Carlo estimator.

Usage

```
vr_control_variate(sim_fn, h, g, E_g, n_paths = 1000L, ...)
```

Arguments

`sim_fn`	A StochSimR simulation function.
`h`	Target function applied to path values matrix. Returns a numeric vector of length `n_paths`.
`g`	Control variate function applied to path values matrix. Returns a numeric vector of length `n_paths`.
`E_g`	Numeric. Known expectation of `g(X)`.
`n_paths`	Positive integer. Number of paths.
`...`	Additional arguments to `sim_fn`.

Details

The control variate estimator is

$$\hat{\mu}_{CV} = \bar{Y} + c^*(\bar{C} - E[C])$$

where $c^* = -\mathrm{Cov}(Y, C)/\mathrm{Var}(C)$ is the optimal coefficient. The theoretical variance reduction factor is $1/(1 - \rho_{YC}^2)$.

Value

A list with components:

estimate Numeric. Control variate estimator.

se Numeric. Standard error.

ci Numeric vector of length 2. 95% confidence interval.

c_star Numeric. Optimal control coefficient.

correlation Numeric. Correlation between `h` and `g`.

raw_estimate Numeric. Crude MC estimate.

raw_se Numeric. Crude MC standard error.

reduction_factor Numeric. Variance reduction ratio.

method Character. `"control_variate"`.

n_samples Integer. Total samples used.

References

Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*. Springer. Section 4.1.

Examples

```
# Price European call; control = terminal stock price
h_call <- function(vals) pmax(vals[nrow(vals), ] - 100, 0)
g_term <- function(vals) vals[nrow(vals), ]
E_g_val <- 100 * exp(0.05)
result <- vr_control_variate(sim_gbm, h = h_call, g = g_term,
  E_g = E_g_val, n_paths = 5000,
  T_max = 1, n_steps = 252, mu = 0.05, sigma = 0.2, x0 = 100)
cat("Estimate:", result$estimate, " Reduction:",
    round(result$reduction_factor, 2), "x\\n")
```

`vr_importance` *Importance Sampling Estimator*

Description

Tilts the simulation measure via an exponential change of drift and reweights using the Girsanov likelihood ratio. Particularly useful for rare-event estimation.

Usage

```
vr_importance(sim_fn, h, drift_shift = 0.5, n_paths = 1000L, ...)
```

Arguments

`sim_fn`	A StochSimR simulation function for a process driven by Brownian motion (e.g., `sim_gbm`).
`h`	Target function applied to path values. Returns a numeric vector.
`drift_shift`	Numeric. Additive shift to the drift parameter `mu`. Positive values tilt the process upward.
`n_paths`	Positive integer. Number of paths.
`...`	Additional arguments to `sim_fn`. Must include `mu`, `sigma`, and `T_max` (or their defaults are used).

Details

Under the tilted measure Q with drift $\mu + \delta$, the Girsanov likelihood ratio for GBM is

$$L = \frac{dP}{dQ} = \exp\Big(-\frac{\delta}{\sigma} W_T - \frac{\delta^2}{2\sigma^2} T\Big)$$

where W_T is the terminal Brownian motion value recovered from the simulated path.

Value

A list with components:

estimate Numeric. Importance sampling estimator.

se Numeric. Standard error.

ci Numeric vector of length 2. 95% confidence interval.

ess Numeric. Effective sample size.

raw_estimate Numeric. Crude MC estimate (separate simulation).

raw_se Numeric. Crude MC standard error.

reduction_factor Numeric. Variance reduction ratio.

method Character. `"importance_sampling"`.

drift_shift Numeric. The applied drift shift.

n_samples Integer. Total samples used.

References

Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*. Springer. Chapter 4.6.

Examples

```
# Estimate P(S_T > 150) with upward drift tilt
h_tail <- function(vals) as.numeric(vals[nrow(vals), ] > 150)
result <- vr_importance(sim_gbm, h = h_tail, drift_shift = 0.4,
  n_paths = 10000, T_max = 1, n_steps = 252,
  mu = 0.05, sigma = 0.2, x0 = 100)
cat("P(S_T > 150):", result$estimate, " ESS:", round(result$ess), "\\n")
```

---

vr_stratified *Stratified Sampling Estimator*

Description

Partitions the probability space into strata and allocates samples proportionally, reducing variance by ensuring balanced coverage.

Usage

```
vr_stratified(sim_fn, h, n_strata = 10L, n_paths = 1000L, ...)
```

Arguments

`sim_fn`	A StochSimR simulation function.
`h`	Target function applied to path values. Returns a numeric vector.
`n_strata`	Positive integer. Number of strata.
`n_paths`	Positive integer. Total number of paths.
`...`	Additional arguments to `sim_fn`.

Details

The stratification is performed by running independent batches (one per stratum) and averaging their means. This is a simple form of stratified sampling that reduces variance when the target function has different behaviour across the strata.

Value

A list with components:

estimate Numeric. Stratified estimator.

se Numeric. Standard error.

ci Numeric vector of length 2. 95% confidence interval.

strata_means Numeric vector. Per-stratum means.

strata_vars Numeric vector. Per-stratum variances.

raw_estimate Numeric. Crude MC estimate.

raw_se Numeric. Crude MC standard error.

reduction_factor Numeric. Variance reduction ratio.

method Character. `"stratified_sampling"`.

n_strata Integer. Number of strata used.

n_samples Integer. Total samples used.

References

Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*. Springer. Section 4.3.

Examples

```
h_call <- function(vals) pmax(vals[nrow(vals), ] - 100, 0)
result <- vr_stratified(sim_gbm, h = h_call, n_strata = 10,
  n_paths = 5000, T_max = 1, n_steps = 252,
  mu = 0.05, sigma = 0.2, x0 = 100)
cat("Estimate:", result$estimate, " SE:", result$se, "\\n")
```

Index