Package 'msde'

July 23, 2025

Title Bayesian Inference for Multivariate Stochastic Differential Equations

Version 1.0.5

Date 2021-12-16

Description Implements an MCMC sampler for the posterior distribution of arbitrary time-homogeneous multivariate stochastic differential equation (SDE) models with possibly latent components. The package provides a simple entry point to integrate user-defined models directly with the sampler's C++ code, and parallelizes large portions of the calculations when compiled with 'OpenMP'.

Depends R (>= 3.0.0)

Imports Rcpp (>= 0.12.7), methods, stats, tools, whisker

LinkingTo Rcpp, RcppProgress

Suggests knitr, rmarkdown, testthat, RcppProgress

VignetteBuilder knitr

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation yes

Author Martin Lysy [aut, cre],
Feiyu Zhu [aut],
JunYong Tong [aut],
Trevor Kitt [ctb],
Nigel Delaney [ctb]

Maintainer Martin Lysy <mlysy@uwaterloo.ca>

Repository CRAN

Date/Publication 2021-12-17 07:50:02 UTC

Contents

`msde-package` *msde: Bayesian Inference for Multivariate Stochastic Differential Equations*

Description

Implements an MCMC sampler for the posterior distribution of arbitrary time-homogeneous multivariate stochastic differential equation (SDE) models with possibly latent components. The package provides a simple entry point to integrate user-defined models directly with the sampler's C++ code, and parallelizes large portions of the calculations when compiled with 'OpenMP'.

Details

See package vignettes; `vignette("msde-quicktut")` for a tutorial and `vignette("msde-exmodels")` for several example models.

Author(s)

Maintainer: Martin Lysy <mlysy@uwaterloo.ca>

Authors:

- Feiyu Zhu
- JunYong Tong

Other contributors:

- Trevor Kitt [contributor]
- Nigel Delaney [contributor]

Examples

```
# Posterior inference for Heston's model

# compile model
hfile <- sde.examples("hest", file.only = TRUE)
param.names <- c("alpha", "gamma", "beta", "sigma", "rho")
data.names <- c("X", "Z")
hmod <- sde.make.model(ModelFile = hfile,
                       param.names = param.names,
                       data.names = data.names)
# or simply load pre-compiled version
hmod <- sde.examples("hest")

# Simulate data
X0 <- c(X = log(1000), Z = 0.1)
theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)
dT <- 1/252
nobs <- 1000
hest.sim <- sde.sim(model = hmod, x0 = X0, theta = theta,
                    dt = dT, dt.sim = dT/10, nobs = nobs)

# initialize MCMC sampler
# both components observed, no missing data between observations
init <- sde.init(model = hmod, x = hest.sim$data,
                 dt = hest.sim$dt, theta = theta)

# Initialize posterior sampling argument
nsamples <- 1e4
burn <- 1e3
hyper <- NULL # flat prior
hest.post <- sde.post(model = hmod, init = init, hyper = hyper,
                      nsamples = nsamples, burn = burn)

# plot the histogram for the sampled parameters
par(mfrow = c(2,3))
for(ii in 1:length(hmod$param.names)) {
  hist(hest.post$params[,ii],breaks=100, freq = FALSE,
       main = parse(text = hmod$param.names[ii]), xlab = "")
}
```

`mou.loglik` *Loglikelihood for multivariate Ornstein-Uhlenbeck process.*

Description

Computes the exact Euler loglikelihood for any amount of missing data using a Kalman filter.

Usage

```
mou.loglik(X, dt, nvar.obs, Gamma, Lambda, Phi, mu0, Sigma0)
```

Arguments

X	An nobs x ndims matrix of complete data.
dt	A scalar or length nobs-1 vector of interobservations times.
nvar.obs	A scalar or length nobs vector of integers between 0 and ndims denoting the number of observed SDE variables in each row of data. Defaults to ndims. See sde.init() for details.
Gamma	A ndims x ndims of linear-drift parameters. See Details.
Lambda	A length-ndims vector of constant-drift parameters. See Details.
Phi	A ndims x ndims positive definite variance matrix. See Details.
mu0, Sigma0	Mean and variance of marginal multivariate normal distribution of X[1,]. Defaults to iid standard normals for each component.

Details

The p-dimensional multivariate Ornstein-Uhlenbeck (mOU) process $Y_t = (Y_{1t}, \ldots, Y_{dt})$ satisfies the SDE

$$dY_t = (\Gamma Y_t + \Lambda)dt + \Phi^{1/2} dB_t,$$

where $B_t = (B_{1t}, \ldots, B_{pt})$ is p-dimensional Brownian motion. Its Euler discretization is of the form

$$Y_{n+1} = Y_n + (\Gamma Y_n + \Lambda)\Delta_n + \Phi^{1/2}\Delta B_n,$$

where $Y_n = Y(t_n)$, $\Delta_n = t_{n+1} - t_n$ and

$$\Delta B_n = B(t_{n+1}) - B(t_n) \overset{\text{ind}}{\sim} \mathcal{N}(0, \Delta_n).$$

Thus, $Y_0, \ldots, Y_N$ is multivariate normal Markov chain for which the marginal distribution of any subset of timepoints and/or components can be efficiently calculated using the Kalman filter. This can be used to check the MCMC output of sde.post() as in the example.

Value

Scalar value of the loglikelihood. See Details.

Examples

```
# bivariate OU model
bmod <- sde.examples("biou")

# simulate some data

# true parameter values
Gamma0 <- .1 * crossprod(matrix(rnorm(4),2,2))
Lambda0 <- rnorm(2)
Phi0 <- crossprod(matrix(rnorm(4),2,2))
Psi0 <- chol(Phi0) # precompiled model uses the Cholesky scale
theta0 <- c(Gamma0, Lambda0, Psi0[c(1,3,4)])
names(theta0) <- bmod$param.names
# initial value
```

```
Y0 <- rnorm(2)
names(Y0) <- bmod$data.names

# simulation
dT <- runif(1, max = .1) # time step
nObs <- 10
bsim <- sde.sim(bmod, x0 = Y0, theta = theta0,
                dt = dT, dt.sim = dT, nobs = nObs)
YObs <- bsim$data

# inference via MCMC
binit <- sde.init(bmod, x = YObs, dt = dT, theta = theta0,
                  nvar.obs = 1) # second component is unobserved
# only Lambda1 is unknown
fixed.params <- rep(TRUE, bmod$nparams)
names(fixed.params) <- bmod$param.names
fixed.params["Lambda1"] <- FALSE
# prior on (Lambda1, Y_0)
hyper <- list(mu = c(0,0), Sigma = diag(2))
names(hyper$mu) <- bmod$data.names
dimnames(hyper$Sigma) <- rep(list(bmod$data.names), 2)

# posterior sampling
nsamples <- 1e5
burn <- 1e3
bpost <- sde.post(bmod, binit, hyper = hyper,
                  fixed.params = fixed.params,
                  nsamples = nsamples, burn = burn)
L1.mcmc <- bpost$params[,"Lambda1"]

# analytic posterior
L1.seq <- seq(min(L1.mcmc), max(L1.mcmc), len = 500)
L1.loglik <- sapply(L1.seq, function(l1) {
  lambda <- Lambda0
  lambda[1] <- l1
  mou.loglik(X = YObs, dt = dT, nvar.obs = 1,
             Gamma = Gamma0, Lambda = lambda, Phi = Phi0,
             mu0 = hyper$mu, Sigma0 = hyper$Sigma)
})
# normalize density
L1.Kalman <- exp(L1.loglik - max(L1.loglik))
L1.Kalman <- L1.Kalman/sum(L1.Kalman)/(L1.seq[2]-L1.seq[1])

# compare
hist(L1.mcmc, breaks = 100, freq = FALSE,
     main = expression(p(Lambda[1]*" | "*bold(Y)[1])),
     xlab = expression(Lambda[1]))
lines(L1.seq, L1.Kalman, col = "red")
legend("topright", legend = c("Analytic", "MCMC"),
       pch = c(NA, 22), lty = c(1, NA), col = c("red", "black"))
```

`mvn.hyper.check` *Argument checking for the default multivariate normal prior.*

Description

Argument checking for the default multivariate normal prior.

Usage

```
mvn.hyper.check(hyper, param.names, data.names)
```

Arguments

`hyper`	The normal prior's hyperparameters: `NULL`, or a list with elements `mu` and `Sigma`, corresponding to a named mean vector and variance matrix (see Details).
`param.names`	Vector of parameter names (see Details).
`data.names`	Vector of data names (see Details).

Details

This function is not meant to be called directly by the user, but rather to parse the hyper-parameters of a default multivariate normal prior distribution to be passed to the C++ code in `sde.prior()` and `sde.post()`. This default prior is multivariate normal on the elements of `(theta, x0)` specified by each of `names(mu)`, `rownames(Sigma)`, and `colnames(Sigma)`. The remaining components are given Lebesgue priors, or a full Lebesgue prior if `hyper == NULL`. If the names of `mu` and `Sigma` are inconsistent an error is thrown.

Value

A list with the following elements:

`mean` The mean vector.

`cholSd` The upper upper Cholesky factor of the variance matrix.

`thetaId` The index of the corresponding variables in `theta`.

`xId` The index of the corresponding variables in `x0`.

sde.diff *SDE diffusion function.*

Description

Computes the SDE model's diffusion function given data and parameter values.

Usage

```
sde.diff(model, x, theta)
```

Arguments

model	An sde.model object.
x	A vector or matrix of data with ndims columns.
theta	A vector or matrix of parameters with nparams columns.

Value

A matrix with ndims^2 columns containing the diffusion function evaluated at x and theta. Each row corresponds to the upper triangular Cholesky factor of the diffusion matrix. If either input contains invalid SDE data or parameters an error is thrown.

Examples

```
# load Heston's model
hmod <- sde.examples("hest")
#'
# single input
theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)
x0 <- c(X = log(1000), Z = 0.1)
sde.diff(model = hmod, x = x0, theta = theta)
#'
# multiple inputs
nreps <- 10
Theta <- apply(t(replicate(nreps, theta)), 2, jitter)
X0 <- apply(t(replicate(nreps, x0)), 2, jitter)
sde.diff(model = hmod, x = X0, theta = Theta)
#'
# mixed inputs
sde.diff(model = hmod, x = x0, theta = Theta)
```

sde.drift *SDE drift function.*

Description

Computes the SDE model's drift function given data and parameter values.

Usage

```
sde.drift(model, x, theta)
```

Arguments

model	An sde.model object.
x	A vector or matrix of data with ndims columns.
theta	A vector or matrix of parameters with nparams columns.

Value

A matrix with ndims columns containing the drift function evaluated at x and theta. If either input contains invalid SDE data or parameters an error is thrown.

Examples

```
# load Heston's model
hmod <- sde.examples("hest")

# single input
x0 <- c(X = log(1000), Z = 0.1)
theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)
sde.drift(model = hmod, x = x0, theta = theta)

# multiple inputs
nreps <- 10
Theta <- apply(t(replicate(nreps,theta)),2,jitter)
X0 <- apply(t(replicate(nreps,x0)),2,jitter)
sde.drift(model = hmod, x = X0, theta = Theta)
```

sde.examples *Example SDE models.*

Description

Provides sample C++ code for several SDE models.

Usage

```
sde.examples(
  model = c("hest", "pgnet", "lotvol", "biou", "eou"),
  file.only = FALSE
)
```

Arguments

model	Character string giving the name of a sample model. Possible values are: hest, pgnet, lotvol, biou, eou. See Details.
file.only	If TRUE returns only the path to the header file containing the sdeModel object implementation.

Details

All pre-compiled models are with the default prior and with OpenMP disabled. A full description of the example models can be found in the package vignette; to view it run `vignette("msde-exmodels")`.

Value

An sde.model object, or the path to the C++ model header file.

See Also

sde.make.model() for sde.model objects, mvn.hyper.check() for specification of the default prior.

Examples

```
# Heston's model
hmod <- sde.examples("hest") # load pre-compiled model

# inspect model's C++ code
hfile <- sde.examples("hest", file.only = TRUE)
cat(readLines(hfile), sep = "\n")

## Not run:
# compile it from scratch
param.names <- c("alpha", "gamma", "beta", "sigma", "rho")
data.names <- c("X", "Z")
hmod <- sde.make.model(ModelFile = hfile,
                       param.names = param.names,
                       data.names = data.names)

## End(Not run)
```

sde.init *MCMC initialization.*

Description

Specifies the observed SDE data, interobservation times, initial parameter and missing data values to be supplied to `sde.post()`.

Usage

```
sde.init(model, x, dt, m = 1, nvar.obs, theta)
```

Arguments

`model`	An `sde.model` object.
`x`	An `nobs x ndims` matrix of data.
`dt`	A scalar or length `nobs-1` vector of interobservations times.
`m`	Positive integer, such that `m-1` evenly-spaced missing data time points are placed between observations. See Details.
`nvar.obs`	A scalar or length `nobs` vector of integers between 0 and `ndims` denoting the number of observed SDE variables in each row of `data`. Defaults to `ndims`. See Details.
`theta`	A length `nparams` vector of parameter values.

Value

An `sde.init` object, corresponding to a list with elements:

`data` An `ncomp x ndims` matrix of complete data, where `ncomp = N_m = m * (nobs-1)+1`.

`dt.m` The complete data interobservation time, `dt_m = dt/m`.

`nvar.obs.m` The number of variables observed per row of `data`. Note that `nvar.obs.m[(i-1)*m+1] == nvar.obs[ii]`, and that `nvar.obs.m[i-1] == 0` if `i` is not a multiple of `m`.

`params` Parameter initial values.

Examples

```
# load Heston's model
hmod <- sde.examples("hest")

# generate some observed data
nObs <- 5
x0 <- c(X = log(1000), Z = 0.1)
X0 <- apply(t(replicate(nObs, x0)), 2, jitter)
dT <- .6
theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)
```

```
# no missing data
sde.init(model = hmod, x = X0, dt = dT, theta = theta)

# all but endpoint volatilities are missing
sde.init(model = hmod, x = X0, dt = dT, m = 1,
         nvar.obs = c(2, rep(1, nObs-2), 2), theta = theta)

# all volatilities missing,
# two completely missing SDE timepoints between observations
m <- 3 # divide each observation interval into m equally spaced timepoints
sde.init(model = hmod, x = X0, dt = dT,
         m = m, nvar.obs = 1, theta = theta)
```

`sde.loglik` *SDE loglikelihood function.*

Description

Evaluates the loglikelihood function given SDE data and parameter values.

Usage

```
sde.loglik(model, x, dt, theta, ncores = 1)
```

Arguments

`model`	An `sde.model` object.
`x`	A matrix or 3-d array of data with `dim(x)[1]` observations and `dim(x)[2]` == `ndims`.
`dt`	A scalar or vector of length `dim(x)[1]-1` of time intervals between observations.
`theta`	A vector or matrix of parameters with `nparams` columns.
`ncores`	If `model` is compiled with `OpenMP`, the number of cores to use for parallel processing. Otherwise, uses `ncores = 1` and gives a warning.

Value

A vector of loglikelihood evaluations, of the same length as the third dimension of `x` and/or first dimension of `theta`. If input contains invalid data or parameters an error is thrown.

Examples

```
# load Heston's model
hmod <- sde.examples("hest")

# Simulate data
nreps <- 10
nobs <- 100
```

```
theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)
Theta <- apply(t(replicate(nreps, theta)), 2, jitter)
x0 <- c(X = log(1000), Z = 0.1)
X0 <- apply(t(replicate(nreps,x0)), 2, jitter)
dT <- 1/252
hsim <- sde.sim(model = hmod, x0 = X0, theta = Theta,
                dt = dT, dt.sim = dT/10, nobs = nobs, nreps = nreps)

# single parameter, single data
sde.loglik(model = hmod, x = hsim$data[,,1], dt = dT, theta = theta)
# multiple parameters, single data
sde.loglik(model = hmod, x = hsim$data[,,1], dt = dT, theta = Theta)
# multiple parameters, multiple data
sde.loglik(model = hmod, x = hsim$data, dt = dT, theta = Theta)
```

`sde.make.model` *Create an SDE model object.*

Description

Compiles the C++ code for various SDE-related algorithms and makes the routines available within R.

Usage

```
sde.make.model(
  ModelFile,
  PriorFile = "default",
  data.names,
  param.names,
  hyper.check,
  OpenMP = FALSE,
  ...
)
```

Arguments

`ModelFile`	Path to the header file where the SDE model is defined.
`PriorFile`	Path to the header file where the SDE prior is defined. See `sde.prior()` for details.
`data.names`	Vector of names for the SDE components. Defaults to `X1,...,Xd`.
`param.names`	Vector of names for the SDE parameters. Defaults to `theta1,...,thetap`.
`hyper.check`	A function with arguments `hyper`, `param.names`, and `data.names` used for passing the model hyper parameters to the C++ code. See `mvn.hyper.check()` for details.
`OpenMP`	Logical; whether the model is compiled with `OpenMP` for C++ level parallelization.
`...`	additional arguments to `Rcpp::sourceCpp()` for compiling the C++ code.

Value

An sde.model object, consisting of a list with the following elements:

ptr Pointer to C++ sde object (sdeRobj) implementing the member functions: drift/diffusion, data/parameter validators, loglikelihood, prior distribution, forward simulation, MCMC algorithm for Bayesian inference.

ndims, nparams The number of SDE components and parameters.

data.names, param.names The names of the SDE components and parameters.

omp A logical flag for whether or not the model was compiled for multicore functionality with OpenMP.

See Also

sde.drift(), sde.diff(), sde.valid(), sde.loglik(), sde.prior(), sde.sim(), sde.post().

Examples

```
# header (C++) file for Heston's model
hfile <- sde.examples("hest", file.only = TRUE)
cat(readLines(hfile), sep = "\n")


# compile the model
param.names <- c("alpha", "gamma", "beta", "sigma", "rho")
data.names <- c("X", "Z")
hmod <- sde.make.model(ModelFile = hfile,
                       param.names = param.names,
                       data.names = data.names)

hmod
```

---

sde.post *MCMC sampler for the SDE posterior.*

Description

A Metropolis-within-Gibbs sampler for the Euler-Maruyama approximation to the true posterior density.

Usage

```
sde.post(
  model,
  init,
  hyper,
  nsamples,
```

```
  burn,
  mwg.sd = NULL,
  adapt = TRUE,
  loglik.out = FALSE,
  last.miss.out = FALSE,
  update.data = TRUE,
  data.out,
  update.params = TRUE,
  fixed.params,
  ncores = 1,
  verbose = TRUE
)
```

Arguments

model	An sde.model object constructed with sde.make.model().
init	An sde.init object constructed with sde.init().
hyper	The hyperparameters of the SDE prior. See sde.prior().
nsamples	Number of MCMC iterations.
burn	Integer number of burn-in samples, or fraction of nsamples to prepend as burn-in.
mwg.sd	Standard deviation jump size for Metropolis-within-Gibbs on parameters and missing components of first SDE observation (see Details).
adapt	Logical or list to specify adaptive Metropolis-within-Gibbs sampling (see Details).
loglik.out	Logical, whether to return the loglikelihood at each step.
last.miss.out	Logical, whether to return the missing sde components of the last observation.
update.data	Logical, whether to update the missing data.
data.out	A scalar, integer vector, or list of three integer vectors determining the subset of data to be returned (see Details).
update.params	Logical, whether to update the model parameters.
fixed.params	Logical vector of length nparams indicating which parameters are to be held fixed in the MCMC sampler.
ncores	If model is compiled with OpenMP, the number of cores to use for parallel processing. Otherwise, uses ncores = 1 and gives a warning.
verbose	Logical, whether to periodically output MCMC status.

Details

The Metropolis-within-Gibbs (MWG) jump sizes can be specified as a scalar, a vector or length nparams + ndims, or a named vector containing the elements defined by sde.init$nvar.obs.m[1] (the missing variables in the first SDE observation) and fixed.params (the SDE parameters which are not held fixed). The default jump sizes for each MWG random variable are .25 * |initial_value| when |initial_value| > 0, and 1 otherwise.

adapt == TRUE implements an adaptive MCMC proposal by Roberts and Rosenthal (2009). At step n of the MCMC, the jump size of each MWG random variable is increased or decreased by $\delta(n)$, depending on whether the cumulative acceptance rate is above or below the optimal value of 0.44. If σ_n is the size of the jump at step n, then the next jump size is determined by

$$\log(\sigma_{n+1}) = \log(\sigma_n) \pm \delta(n), \qquad \delta(n) = \min(.01, 1/n^{1/2}).$$

When adapt is not logical, it is a list with elements max and rate, such that delta(n) = min(max, 1/n^rate). These elements can be scalars or vectors in the same manner as mwg.sd.

For SDE models with thousands of latent variables, data.out can be used to thin the MCMC missing data output. An integer vector or scalar returns specific or evenly-spaced posterior samples from the ncomp x ndims complete data matrix. A list with elements isamples, icomp, and idims determines which samples, time points, and SDE variables to return. The first of these can be a scalar or vector with the same meaning as before.

Value

A list with elements:

params An nsamples x nparams matrix of posterior parameter draws.

data A 3-d array of posterior missing data draws, for which the output dimensions are specified by data.out.

init The sde.init object which initialized the sampler.

data.out A list of three integer vectors specifying which timepoints, variables, and MCMC iterations correspond to the values in the data output.

mwg.sd A named vector of Metropolis-within-Gibbs standard devations used at the last posterior iteration.

hyper The hyperparameter specification.

loglik If loglik.out == TRUE, the vector of nsamples complete data loglikelihoods calculated at each posterior sample.

last.iter A list with elements data and params giving the last MCMC sample. Useful for resuming the MCMC from that point.

last.miss If last.miss.out == TRUE, an nsamples x nmissN matrix of all posterior draws for the missing data in the final observation. Useful for SDE forecasting at future timepoints.

accept A named list of acceptance rates for the various components of the MCMC sampler.

References

Roberts, G.O. and Rosenthal, J.S. "Examples of adaptive MCMC." *Journal of Computational and Graphical Statistics* 18.2 (2009): 349-367. http://www.probability.ca/jeff/ftpdir/adaptex.pdf.

Examples

```
# Posterior inference for Heston's model
hmod <- sde.examples("hest") # load pre-compiled model
```

```
# Simulate data
X0 <- c(X = log(1000), Z = 0.1)
theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)
dT <- 1/252
nobs <- 1000
hest.sim <- sde.sim(model = hmod, x0 = X0, theta = theta,
                    dt = dT, dt.sim = dT/10, nobs = nobs)

# initialize MCMC sampler
# both components observed, no missing data between observations
init <- sde.init(model = hmod, x = hest.sim$data,
                 dt = hest.sim$dt, theta = theta)

# Initialize posterior sampling argument
nsamples <- 1e4
burn <- 1e3
hyper <- NULL # flat prior
hest.post <- sde.post(model = hmod, init = init, hyper = hyper,
                      nsamples = nsamples, burn = burn)

# plot the histogram for the sampled parameters
par(mfrow = c(2,3))
for(ii in 1:length(hmod$param.names)) {
  hist(hest.post$params[,ii],breaks=100, freq = FALSE,
       main = parse(text = hmod$param.names[ii]), xlab = "")
}
```

`sde.prior` *SDE prior function.*

Description

Evaluates the SDE prior given data, parameter, and hyperparameter values.

Usage

```
sde.prior(model, theta, x, hyper)
```

Arguments

`model`	An `sde.model` object.
`theta`	A vector or matrix of parameters with `nparams` columns.
`x`	A vector or matrix of data with `ndims` columns.
`hyper`	The hyperparameters of the SDE prior. See Details.

Details

The prior is constructed at the C++ level by defining a function (i.e., public member)

```
double logPrior(double *theta, double *x)
```

within the sdePrior class. At the R level, the hyper.check argument of sde.make.model() is a function with arguments hyper, param.names, data.names used to convert hyper into a list of NULL or double-vectors which get passed on to the C++ code. This function can also be used to throw R-level errors to protect the C++ code from invalid inputs, as is done for the default prior in mvn.hyper.check(). For a full example see the "Custom Prior" section in vignette("msde-quicktut").

Value

A vector of log-prior densities evaluated at the inputs.

Examples

```
hmod <- sde.examples("hest") # load Heston's model

# setting prior for 3 parameters
rv.names <- c("alpha","gamma","rho")
mu <- rnorm(3)
Sigma <- crossprod(matrix(rnorm(9),3,3))
names(mu) <- rv.names
colnames(Sigma) <- rv.names
rownames(Sigma) <- rv.names
hyper <- list(mu = mu, Sigma = Sigma)

# Simulate data
nreps <- 10
theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)
x0 <- c(X = log(1000), Z = 0.1)
Theta <- apply(t(replicate(nreps,theta)),2,jitter)
X0 <- apply(t(replicate(nreps,x0)),2,jitter)

sde.prior(model = hmod, x = X0, theta = Theta, hyper = hyper)
```

---

sde.sim *Simulation of multivariate SDE trajectories.*

Description

Simulates a discretized Euler-Maruyama approximation to the true SDE trajectory.

Usage

```
sde.sim(
  model,
  x0,
  theta,
  dt,
  dt.sim,
  nobs,
  burn = 0,
  nreps = 1,
  max.bad.draws = 5000,
  verbose = TRUE
)
```

Arguments

model	An sde.model object.
x0	A vector or a matrix of size nreps x ndims of the SDE values at time 0.
theta	A vector or matrix of size nreps x nparams of SDE parameters.
dt	Scalar interobservation time.
dt.sim	Scalar interobservation time for simulation. That is, interally the interobservation time is dt.sim but only one out of every dt/dt.sim simulation steps is kept in the output.
nobs	The number of SDE observations per trajectory to generate.
burn	Scalar burn-in value. Either an integer giving the number of burn-in steps, or a value between 0 and 1 giving the fraction of burn-in relative to nobs.
nreps	The number of SDE trajectories to generate.
max.bad.draws	The maximum number of times that invalid forward steps are proposed. See Details.
verbose	Whether or not to display information on the simulation.

Details

The simulation algorithm is a Markov process with $Y_0 = x_0$ and

$$Y_{t+1} \sim \mathcal{N}(Y_t + \mathrm{dr}(Y_t, \theta)dt_{\mathrm{sim}}, \mathrm{df}(Y_t, \theta)dt_{\mathrm{sim}}),$$

where $\mathrm{dr}(y, \theta)$ is the SDE drift function and $\mathrm{df}(y, \theta)$ is the diffusion function on the **variance** scale. At each step, a while-loop is used until a valid SDE draw is produced. The simulation algorithm terminates after nreps trajectories are drawn or once a total of max.bad.draws are reached.

Value

A list with elements:

data An array of size nobs x ndims x nreps containing the simulated SDE trajectories.

`params` The vector or matrix of parameter values used to generate the data.

`dt, dt.sim` The actual and internal interobservation times.

`nbad` The total number of bad draws.

Examples

```
# load pre-compiled model
hmod <- sde.examples("hest")

# initial values
x0 <- c(X = log(1000), Z = 0.1)
theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)

# simulate data
dT <- 1/252
nobs <- 2000
burn <- 500
hsim <- sde.sim(model = hmod, x0 = x0, theta = theta,
                dt = dT, dt.sim = dT/10,
                nobs = nobs, burn = burn)

par(mfrow = c(1,2))
plot(hsim$data[,"X"], type = "l", xlab = "Time", ylab = "",
     main = expression(X[t]))
plot(hsim$data[,"Z"], type = "l", xlab = "Time", ylab = "",
     main = expression(Z[t]))
```

---

sde.valid *SDE data and parameter validators.*

Description

Checks whether input SDE data and parameters are valid.

Usage

```
sde.valid.data(model, x, theta)

sde.valid.params(model, theta)
```

Arguments

`model`	An `sde.model` object.
`x`	A length-`ndims` vector or `ndims`-column matrix of SDE data.
`theta`	A length-`nparams` vector or `nparams`-column of SDE parameter values.

Value

A logical scalar or vector indicating whether the given data/parameter pair is valid.

Examples

```
# Heston's model
# valid data is: Z > 0
# valid parameters are: gamma, sigma > 0, |rho| < 1, beta > .5 * sigma^2
hmod <- sde.examples("hest") # load model

theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)

# valid data
x0 <- c(X = log(1000), Z = 0.1)
sde.valid.data(model = hmod, x = x0, theta = theta)

# invalid data
x0 <- c(X = log(1000), Z = -0.1)
sde.valid.data(model = hmod, x = x0, theta = theta)

# valid parameters
theta <- c(alpha = 0.1, gamma = 1, beta = 0.8, sigma = 0.6, rho = -0.8)
sde.valid.params(model = hmod, theta = theta)

# invalid parameters
theta <- c(alpha = 0.1, gamma = -4, beta = 0.8, sigma = 0.6, rho = -0.8)
sde.valid.params(model = hmod, theta = theta)
```

Index