Package ‘BaPreStoPro’

May 3, 2016

Type Package

Title Bayesian Prediction of Stochastic Processes

Version 0.1

Date 2016-04-25

Author Simone Hermann

Maintainer `<hermann@statistik.tu-dortmund.de>`

Description Bayesian inference and prediction for several kinds of stochastic processes, e.g. jump diffusion, (mixed) diffusion models.

License GPL (>= 2)

Depends mvtnorm,stats,methods

LazyData TRUE

R topics documented:

`BaPreStoPro-package` *Bayesian Prediction of Stochastic Processes*

Description

This package contains simulate, estimate and predict methods for jump diffusions, diffusions, mixed diffusions, hidden (mixed) diffusion models and regression models for comparison.

Details

Package:	BaPreStoPro
Type:	Package
Version:	1.0
Date:	2016-04-25
License:	GLP-2, GLP-3

An overview of how to use the package, including the most important functions

Author(s)

Simone Hermann <hermann@statistik.tu-dortmund.de>

References

Bayesian Prediction of Crack Growth Based on a Hierarchical Diffusion Model. S. Hermann, K. Ickstadt and C. Mueller, *appearing in: Applied Stochastic Models in Business and Industry 2016.*

Examples

```
model <- set.to.class("Diffusion", parameter = list(phi = 0.5, gamma2 = 0.01))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5, plot.series = TRUE)
est_diff <- estimate(model, t, data, 100) # better: 10000
plot(est_diff)
pred_diff <- predict(est_diff, b.fun.mat = function(phi, t, y) phi[,1])  # much faster
```

`ad.propSd` *Helping function*

Description

Adaptive MCMC

Usage

```
ad.propSd(chain, propSd, iteration, lower = 0.3, upper = 0.6,
  delta.n = function(n) min(0.05, 1/sqrt(n)))
```

Arguments

`chain`	Markov chain
`propSd`	current proposal standard deviation
`iteration`	current iteration (batch)
`lower`	lower bound
`upper`	upper bound
`delta.n`	function of batch number

Value

adjusted proposal standard deviation

`BinSearch` *Binary Search Algorithm*

Description

Binary Search Algorithm

Usage

```
BinSearch(Fun, len, candArea, grid = 1e-05, method = c("vector", "free"))
```

Arguments

`Fun`	cumulative distribution function
`len`	number of samples
`candArea`	candidate area
`grid`	fineness degree
`method`	vectorial ("vector") or not ("free")

Value

vector of samples

Examples

```
test <- BinSearch(function(x) pnorm(x, 5, 1), 1000, candArea = c(0, 10), method = "free")
plot(density(test))
curve(dnorm(x, 5, 1), col = 2, add = TRUE)
```

`class.to.list` *Builds a list from class*

Description

Class to list

Usage

```
class.to.list(cl)
```

Arguments

`cl`	class

Value

list

`diagnostic` *Calcucation of burn-in phase and thin rate*

Description

Proposal for burn-in and thin rate

Usage

```
diagnostic(chain, dependence = 0.8, m = 10)
```

Arguments

`chain`	vector of Markov chain samples
`dependence`	allowed dependence for the chain
`m`	number of blocks

`dNtoTimes` *Transformation of counting process to vector of event times*

Description

Transformation of vector of counting process to event times.

Usage

```
dNtoTimes(dN, t)
```

Arguments

`dN`	vector of differences of counting process
`t`	times of counting process

Value

vector of event times

Examples

```
t <- seq(0, 1, by = 0.01)
process <- simN(t, c(5, 0.5), len = 1)$N
times <- dNtoTimes(diff(process), t)
```

`drawSDE` *Function for simulating diffusion process*

Description

Simulation of process defined by $dYt = b(\phi, t, Y_t)dt + \gamma sigmaTilde(t, Y_t)dW_t$.

Usage

```
drawSDE(phi, gamma2, t, b, fODE, sigmaTilde, mw = 10,
  strictly.positive = TRUE)
```

Arguments

`phi`	parameter ϕ
`gamma2`	parameter γ^2
`t`	vector of time points
`b`	drift function
`fODE`	function for the starting point dependent on phi, for fixed y0: fODE = function(phi, t) y0
`sigmaTilde`	variance function $s(\gamma, t, y) = \gamma sigmaTilde(t, y)$
`mw`	mesh width to simulate the time-continuity
`strictly.positive`	if TRUE, only positive values for process Y_t

Value

data series in t

`estimate` *Estimation*

Description

Method for the S4 classes

Usage

```
estimate(model.class, ...)
```

Arguments

`model.class`	class
`...`	parameters dependent on the model class

`estimate,Diffusion-method` *Estimation for diffusion process*

Description

Bayesian estimation of the parameters of the stochastic process $dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t$.

Usage

```
## S4 method for signature 'Diffusion'
estimate(model.class, t, data, nMCMC)
```

Arguments

`model.class`	class of the respective model including all required information, see function set.to.class
`t`	vector of time points
`data`	vector or list or matrix of observation variables
`nMCMC`	length of Markov chain

Examples

```
model <- set.to.class("Diffusion", parameter = list(phi = 0.5, gamma2 = 0.01))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5, plot.series = TRUE)
est_diff <- estimate(model, t, data, 1000)
plot(est_diff)
```

estimate,hiddenDiffusion-method *Estimation for noisy / hidden diffusion process*

Description

Bayesian estimation of the model, $Z_i = Y_{t_i} + \epsilon_i, dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t$.

Usage

```
## S4 method for signature 'hiddenDiffusion'
estimate(model.class, t, data, nMCMC, Npart = 100)
```

Arguments

`model.class`	class of the respective model including all required information, see function set.to.class
`t`	vector of time points
`data`	vector or list or matrix of observation variables
`nMCMC`	length of Markov chain
`Npart`	number of particles in the particle Gibbs sampler

Examples

```
model <- set.to.class("hiddenDiffusion", y0.fun = function(phi, t) 0.5,
             parameter = list(phi = 5, gamma2 = 1, sigma2 = 0.1))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est <- estimate(model, t, data$Z, 100)  # nMCMC should be much larger!
plot(est)
## Not run:
# OU
b.fun <- function(phi, t, y) phi[1]-phi[2]*y
model <- set.to.class("hiddenDiffusion", y0.fun = function(phi, t) 0.5,
               parameter = list(phi = c(10, 5), gamma2 = 1, sigma2 = 0.1),
               b.fun = b.fun, sT.fun = function(t, x) 1)
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est <- estimate(model, t, data$Z, 1000)
plot(est)

## End(Not run)
```

`estimate,hiddenmixedDiffusion-method` *Estimation for noisy/hidden mixed diffusion process*

Description

Bayesian estimation of a stochastic process $Z_{ij} = Y_{t_{ij}} + \epsilon_{ij}, dY_t = b(\phi_j, t, Y_t)dt + s(\gamma, t, Y_t)dW_t, \phi_j\ N(\mu, \Omega)$.

Usage

```
## S4 method for signature 'hiddenmixedDiffusion'
estimate(model.class, t, data, nMCMC,
  Npart = 100)
```

Arguments

`model.class`	class of the respective model including all required information, see function set.to.class
`t`	vector of time points
`data`	vector or list or matrix of observation variables
`nMCMC`	length of Markov chain
`Npart`	number of particles in the particle Gibbs sampler

Examples

```
mu <- c(5, 1); Omega <- c(0.9, 0.04)
phi <- cbind(rnorm(21, mu[1], sqrt(Omega[1])), rnorm(21, mu[2], sqrt(Omega[2])))
y0.fun <- function(phi, t) phi[2]
model <- set.to.class("hiddenmixedDiffusion", y0.fun = y0.fun,
                b.fun = function(phi, t, y) phi[1],
             parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 1, sigma2 = 0.01))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
## Not run:
est <- estimate(model, t, data$Z[1:20,], 2000)
plot(est)

## End(Not run)
```

`estimate,jumpDiffusion-method` *Estimation for jump diffusion process*

Description

Bayesian estimation of a stochastic process $dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t + h(\eta, t, Y_t)dN_t$.

Usage

```
## S4 method for signature 'jumpDiffusion'
estimate(model.class, t, data, nMCMC)
```

Arguments

`model.class`	class of the respective model including all required information, see function set.to.class
`t`	vector of time points
`data`	vector or list or matrix of observation variables
`nMCMC`	length of Markov chain

Examples

```
model <- set.to.class("jumpDiffusion", Lambda = function(t, xi) (t/xi[2])^xi[1],
              parameter = list(theta = 0.1, phi = 0.05, gamma2 = 0.1, xi = c(3, 1/4)))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5, plot.series = TRUE)
est <- estimate(model, t, data, 1000)
plot(est)
```

estimate,Merton-method

Estimation for jump diffusion process

Description

Bayesian estimation of a stochastic process $Y_t = y_0 \exp(\phi t - \gamma 2/2t + \gamma W_t + \log(1+\theta)N_t)$.

Usage

```
## S4 method for signature 'Merton'
estimate(model.class, t, data, nMCMC)
```

Arguments

`model.class`	class of the respective model including all required information, see function set.to.class
`t`	vector of time points
`data`	vector or list or matrix of observation variables
`nMCMC`	length of Markov chain

Examples

```
model <- set.to.class("Merton", parameter = list(thetaT = 0.1, phi = 0.05, gamma2 = 0.1, xi = 10))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5, plot.series = TRUE)
est <- estimate(model, t, data, 1000)
plot(est)
## Not run:
est_hidden <- estimate(model, t, data$Y, 1000)
plot(est_hidden)

## End(Not run)
```

`estimate,mixedDiffusion-method`
Estimation for mixed diffusion process

Description

Bayesian estimation of a stochastic process $dY_t = b(\phi_j, t, Y_t)dt + s(\gamma, t, Y_t)dW_t, \phi_j\ N(\mu, \Omega)$.

Usage

```
## S4 method for signature 'mixedDiffusion'
estimate(model.class, t, data, nMCMC)
```

Arguments

`model.class`	class of the respective model including all required information, see function set.to.class
`t`	vector of time points
`data`	vector or list or matrix of observation variables
`nMCMC`	length of Markov chain

Examples

```
mu <- 2; Omega <- 0.4; phi <- matrix(rnorm(21, mu, sqrt(Omega)))
model <- set.to.class("mixedDiffusion",
             parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.1),
             b.fun = function(phi, t, x) phi*x, sT.fun = function(t, x) x)
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est <- estimate(model, t, data[1:20,], 100)  # nMCMC should be much larger
plot(est)
# OU
b.fun <- function(phi, t, y) phi[1]-phi[2]*y; y0.fun <- function(phi, t) phi[3]
mu <- c(10, 5, 0.5); Omega <- c(0.9, 0.01, 0.01)
phi <- sapply(1:3, function(i) rnorm(21, mu[i], sqrt(Omega[i])))
model <- set.to.class("mixedDiffusion",
               parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.1),
               y0.fun = y0.fun, b.fun = b.fun, sT.fun = function(t, x) 1)
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est <- estimate(model, t, data[1:20,], 100)  # nMCMC should be much larger
plot(est)
```

estimate,mixedRegression-method *Estimation for mixed regression model*

Description

Bayesian estimation of the parameter of the regression model $y_i = f(\phi_j, t_i) + \epsilon_i, \phi_j \ N(\mu, \Omega)$.

Usage

```
## S4 method for signature 'mixedRegression'
estimate(model.class, t, data, nMCMC)
```

Arguments

model.class	class of the respective model including all required information, see function set.to.class
t	vector of time points
data	vector or list or matrix of observation variables
nMCMC	length of Markov chain

Examples

```
mu <- c(10, 5); Omega <- c(0.9, 0.01)
phi <- cbind(rnorm(21, mu[1], sqrt(Omega[1])), rnorm(21, mu[2], sqrt(Omega[2])))
model <- set.to.class("mixedRegression",
                 parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.1),
                 fun = function(phi, t) phi[1]*t + phi[2], sT.fun = function(t) 1)
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est <- estimate(model, t, data[1:20,], 2000)
plot(est)
```

estimate,NHPP-method *Estimation for Poisson process*

Description

Bayesian estimation of a nonhomogeneous Poisson process.

Usage

```
## S4 method for signature 'NHPP'
estimate(model.class, t, data, nMCMC)
```

Arguments

`model.class`	class of the respective model including all required information, see function set.to.class
`t`	vector of time points
`data`	vector or list or matrix of observation variables
`nMCMC`	length of Markov chain

Examples

```
model <- set.to.class("NHPP", parameter = list(xi = c(5, 1/2)),
                     Lambda = function(t, xi) (t/xi[2])^xi[1])
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est_NHPP <- estimate(model, t, data$Times, 10000)
plot(est_NHPP)
```

`estimate,Regression-method`

Estimation for regression model

Description

Bayesian estimation of the parameter of the regression model $y_i = f(\phi, t_i) + \epsilon_i$.

Usage

```
## S4 method for signature 'Regression'
estimate(model.class, t, data, nMCMC)
```

Arguments

`model.class`	class of the respective model including all required information, see function set.to.class
`t`	vector of time points
`data`	vector or list or matrix of observation variables
`nMCMC`	length of Markov chain

Examples

```
t <- seq(0,1, by = 0.01)
model <- set.to.class("Regression", fun = function(phi, t) phi[1]*t + phi[2],
                     parameter = list(phi = c(1,2), gamma2 = 0.1))
data <- simulate(model, t = t, plot.series = TRUE)
est <- estimate(model, t, data, 1000)
plot(est)
```

`estimate,reg_hiddenNHPP-method`
Estimation for regression model dependent on Poisson process

Description

Bayesian estimation of the parameter of the regression model $y_i = f(t_i, N_i, \theta) + \epsilon_i$.

Usage

```
## S4 method for signature 'reg_hiddenNHPP'
estimate(model.class, t, data, nMCMC)
```

Arguments

`model.class`	class of the respective model including all required information, see function set.to.class
`t`	vector of time points
`data`	vector or list or matrix of observation variables
`nMCMC`	length of Markov chain

Examples

```
t <- seq(0,1, by = 0.01)
model <- set.to.class("reg_hiddenNHPP", fun = function(t, N, theta) exp(theta[1]*t) + theta[2]*N,
                  parameter = list(theta = c(2, 2), gamma2 = 0.25, xi = c(3, 0.5)),
                  Lambda = function(t, xi) (t/xi[2])^xi[1])
data <- simulate(model, t = t)
est <- estimate(model, t, data, 1000)
plot(est)
## Not run:
est_hid <- estimate(model, t, data$Y, 1000)
plot(est_hid)

## End(Not run)
```

`estReg` *Bayesian estimation in mixed nonlinear regression models*

Description

Bayesian estimation of the random effects ϕ_j in the mixed nonlinear regression model $y_{ij} = f(\phi_j, t_{ij}) + \epsilon_{ij}, \epsilon_{ij}\ N(0, \gamma^2 * s^2(t_{ij})), \phi_j\ N(\mu, \Omega)$ and the parameters μ, Ω, γ^2.

Usage

```
estReg(t, y, prior, start, fODE, sVar, ipred = 1, cut, len = 1000,
  mod = c("Gompertz", "logistic", "Weibull", "Richards", "Paris", "Paris2"),
  propPar = 0.02)
```

Arguments

`t`	vector of observation times
`y`	matrix of the M trajectories
`prior`	list of prior parameters - list(m, v, priorOmega, alpha, beta)
`start`	list of starting values
`fODE`	regression function
`sVar`	variance function
`ipred`	which of the M trajectories is the one to be predicted
`cut`	the index how many of the ipred-th series are used for estimation
`len`	number of iterations of the MCMC algorithm
`mod`	model out of Gompertz, Richards, logistic, Weibull, only used instead of fODE
`propPar`	proposal standard deviation of phi is |start$mu|*propPar

Value

`phi`	samples from posterior of ϕ
`mu`	samples from posterior of μ
`Omega`	samples from posterior of Ω
`gamma2`	samples from posterior of γ^2

`estReg_single` *Bayesian stimation in nonlinear regression models*

Description

Bayesian estimation of the parameters of the mixed nonlinear regression model $y_j = f(\phi, t_j) + \epsilon_j, \epsilon_j\ N(0, \gamma^2 * s^2(t_j))$.

Usage

```
estReg_single(t, y, prior, start, fODE, sVar, len = 1000, mod = "Gompertz")
```

Arguments

`t`	vector of observation times
`y`	vector of the M trajectories
`prior`	list of prior parameters - list(mu, Omega, alpha, beta)
`start`	list of starting values
`fODE`	regression function
`sVar`	variance function
`len`	number of iterations of the MCMC algorithm
`mod`	model out of Gompertz, Richards, logistic, Weibull, only used instead of fODE

Value

`phi`	estimator of ϕ
`gamma2`	estimator of γ^2

estSDE — *Bayesian estimation in mixed stochastic differential equations*

Description

Bayesian estimation of the random effects ϕ_i in the mixed SDE $dY_i(t) = b(\phi_i, t, Y_i(t))dt + \gamma s(t, Y_i(t))dW_i(t), \phi_i\ N(\mu, \Omega), i = 1, ..., n$ and the parameters μ, Ω, γ^2.

Usage

```
estSDE(t, y, prior, start, y0.fun, bSDE, sVar, ipred = 1, cut, len = 1000,
  mod = c("Gompertz", "logistic", "Weibull", "Richards", "Paris", "Paris2"),
  propPar = 0.2)
```

Arguments

`t`	vector of observation times
`y`	matrix or list of the n trajectories
`prior`	list of prior parameters - list(m, v, alpha.omega, beta.omega, alpha.gamma, beta.gamma)
`start`	list of starting values
`y0.fun`	$y_0(\phi, t_1)$ function
`bSDE`	b(phi, t, x) drift function
`sVar`	variance function s^2
`ipred`	which of the n trajectories is the one to be predicted
`cut`	the index how many of the ipred-th series are used for estimation
`len`	number of iterations of the MCMC algorithm - chain length
`mod`	model out of Gompertz, Richards, logistic, Weibull, Paris, Paris2, only used instead of bSDE
`propPar`	proposal standard deviation of phi is |start$mu|*propPar

Details

Simulation from the posterior distribution of the random effect from n independent trajectories of the SDE (the Brownian motions $W1, ..., Wn$ are independent).

Value

`phi`	samples from posterior of ϕ
`mu`	samples from posterior of μ
`Omega`	samples from posterior of Ω
`gamma2`	samples from posterior of γ^2

References

Hermann et al. (2015)

`estSDE_single` *Bayesian estimation in stochastic differential equations*

Description

Bayesian estimation of the parameters in the SDE $dY(t) = b(\phi, t, Y(t))dt + \gamma s(t, Y(t))dW(t)$.

Usage

```
estSDE_single(t, X, prior, start, bSDE, sVar, len = 1000)
```

Arguments

`t`	vector of observation times
`X`	vector of the M trajectories
`prior`	list of prior parameters - list(mu, Omega, alpha, beta)
`start`	list of starting values
`bSDE`	drift function
`sVar`	variance function
`len`	number of iterations of the MCMC algorithm

Details

Simulation from the posterior distribution of the random effect from n independent trajectories of the SDE (the Brownian motions $W1, ..., Wn$ are independent).

Value

`phi`	estimator of ϕ
`gamma2`	estimator of γ^2

`est_JD_Euler` *Metropolis within Gibbs sampler*

Description

Bayesian estimation of the parameter of the jump diffusion process define by SDE $dXt = b(\phi, t, Xt)dt + s(\gamma, t, Xt)dWt + h(\theta, t, Xt)dNt$.

Usage

```
est_JD_Euler(X, N, t, n = 1000, start, b, s, h, priorRatio, Lambda,
  int = c("Weibull", "Exp"), rangeN = 2, propSd = 0.5, it.xi = 5)
```

Arguments

`X`	vector of observed variables
`N`	vector of Poisson process variables
`t`	vector of time points
`n`	length of Markov chain
`start`	list of starting values, list(phi, theta, gamma2)
`b`	drift function
`s`	variance function
`h`	jump high function
`priorRatio`	list of functions for the prior ratio of MH step, if missing: non-informative
`Lambda`	intensity rate function
`int`	if Lambda is missing, one of "Weibull" or "Exp"
`rangeN`	range for candidates for filtering N
`propSd`	starting value for proposal standard deviation
`it.xi`	number of iterations of MH inside the Gibbs sampler

Value

`phi`	estimator of ϕ
`gamma2`	estimator of $\gamma 2$
`theta`	estimator of θ
`xi`	estimator of ξ
`N`	estimator of latent variable N, if not observed
`prop`	storage of adapted proposal variances

`est_Merton` *Gibbs sampler*

Description

Bayesian estimation of the parameter of the jump diffusion process $X_t = x_0 \exp(\phi t - \gamma 2/2t + \gamma W_t + \log(1+\theta)N_t)$.

Usage

```
est_Merton(X, N, t, n = 1000, start, prior, Lambda, rangeN = 2,
  it.xi = 10)
```

Arguments

`X`	vector of observed variables
`N`	vector of Poisson process variables, optional: if missing, filtering
`t`	vector of time points
`n`	length of Markov chain
`start`	list of starting values, list(thetaT, gamma2)
`prior`	list of prior values, list(mu_phi, s_phi, mu_th, s_th, alpha, beta)
`Lambda`	intensity rate function
`rangeN`	range of candidates for filtering N
`it.xi`	number of iterations of MH inside the Gibbs sampler

Value

`phi`	estimator of ϕ
`gamma2`	estimator of $\gamma 2$
`thetaT`	estimator of $log(1+\theta)$
`xi`	estimator of ξ
`N`	estimator of latent variable N, if not observed

`est_NHPP` *Metropolis-Hastings sampler*

Description

Bayesian estimation of the parameter of the intensity rate.

Usage

```
est_NHPP(jumpTimes, Tend, start, n = 5000, int = c("Weibull", "Exp"),
  priorRatio, proposal = c("lognormal", "normal"), Lambda)
```

Arguments

`jumpTimes`	vector of jump (or event) times
`Tend`	last observation vector of the NHPP
`start`	starting value
`n`	length of Markov chain
`int`	one out of "Weibull" or "Exp"
`priorRatio`	function for prior ratio, if missing: non-informative
`proposal`	"lognormal" (for positive parameters, default) or "normal"
`Lambda`	intensity rate function

Value

p x n dimensional matrix of posterior samples

Examples

```
# exponential
Lambda <- function(t, xi){
xi[2]*exp(xi[1]*t)-xi[2]
}
jumpTimes <- simN(seq(0, 1, by = 0.001), c(4, 0.5), len = 1, Lambda = Lambda)$Times
chain <- est_NHPP(jumpTimes, 1, c(4, 0.5), Lambda = Lambda)
# chain <- est_NHPP(jumpTimes, 1, c(4, 0.5), int = "Exp")
plot(chain[1,], type="l"); abline(h = 4)
plot(chain[2,], type="l"); abline(h = 0.5)

# weibull
Lambda <- function(t, xi){
(t/xi[2])^xi[1]
}
jumpTimes <- simN(seq(0, 1, by = 0.001), c(3, 0.5), len = 1, Lambda = Lambda)$Times
chain <- est_NHPP(jumpTimes, 1, c(3, 0.5), Lambda = Lambda)
# chain <- est_NHPP(jumpTimes, 1, c(3, 0.5), int = "Weibull")
plot(chain[1,], type="l"); abline(h = 3)
plot(chain[2,], type="l"); abline(h = 0.5)
```

est_reg_hiddenNHPP *Gibbs sampler*

Description

Bayesian estimation of the parameter of the regression model $y_i = f(t_i, N_i, \theta) + \epsilon_i$.

Usage

```
est_reg_hiddenNHPP(Y, N, t, fun, n = 1000, start, prior, Lambda,
  int = c("Weibull", "Exp"), rangeN = 2)
```

Arguments

Y	vector of observation variables
N	vector of Poisson process variables
t	vector of time points
fun	regression function
n	length of Markov chain
start	list of starting values
prior	list of prior values
Lambda	intensity rate function
int	one out of "Weibull" or "Exp", if Lambda is missing
rangeN	range for candidates of filtering N, if unobserved

Value

`theta`	Markov chains of θ
`gamma2`	Markov chains of γ^2
`N`	if hidden: Markov chains of N
`xi`	Markov chains of ξ

`est_SEP` *Metropolis-Hastings sampler*

Description

Bayesian estimation of the parameter of the self-exciting process.

Usage

```
est_SEP(jumpTimes, Tend, start, n = 5000, s = 200, f_xi, priorRatio,
  proposals, Lmax = 35)
```

Arguments

`jumpTimes`	vector of event times
`Tend`	last observation time, if missing => last event time
`start`	starting value
`n`	length of Markov chain
`s`	stress range
`f_xi`	exp(-f_xi) intensity function
`priorRatio`	function(old, new) for prior ratio, if missing => noninformative
`proposals`	list of functions: draw(old) and ratio(new, old)
`Lmax`	maximal number of tension wires that can break

Value

p x n dimensional matrix of posterior samples, p length of vector xi

Examples

```
wt <- rexp(20, exp(- 1 + 0.25*200/(35-(0:19))))
jumpTimes <- cumsum(wt)
chain <- est_SEP(jumpTimes, start = c(1, 0.25), s = 200)
par(mfrow = c(2,1))
plot(chain[1,], type = "l")
abline(h = 1, col = 2)
plot(chain[2,], type = "l")
abline(h = 0.25, col = 2)

print("acceptance rate:")
length(unique(chain[1,]))/length(chain[1,])

# or:
```

```
s <- seq(100, 300, length = 10)
jumpTimes <- lapply(1:10, function(a){
wt <- rexp(20, exp(- 1 + 0.25*s[a]/(35-(0:19))))
cumsum(wt)
})
chain <- est_SEP(jumpTimes, start = c(1, 0.25), s = s)
```

`findCandidateArea` *Helping function*

Description

Finding suitable candidate area

Usage

```
findCandidateArea(VFun, start = 1, pos.support = TRUE, quasi.null = 1e-05)
```

Arguments

`VFun`	cumulative distribution function
`start`	starting point for search
`pos.support`	if TRUE: only positive support
`quasi.null`	size of values to be defined as zero, default: 10^-5

Value

adjusted proposal standard deviation

`getFun` *Yields the regression or drift function for the specific models*

Description

Gives out the regression function (ODE) or the drift function (SDE) for the specific model out of Gompertz, Richards, logistic, Weibull

Usage

```
getFun(model, mod)
```

Arguments

`model`	one out of SDE, ODE
`mod`	one out of Gompertz, Richards, logistic, Weibull

Value

function

getPrior *Builds list of prior parameters*

Description

Creation of list of parameters conditional on true values μ and γ^2

Usage

```
getPrior(parameter, model = c("jumpDiffusion", "Merton", "Diffusion",
  "mixedDiffusion", "hiddenDiffusion", "hiddenmixedDiffusion", "reg_hiddenNHPP",
  "NHPP", "Regression", "mixedRegression"))
```

Arguments

`parameter`	list of parameters
`model`	name of model

Value

list of prior values

jumpDiffusion-class *S4 class for the jump diffusion process*

Description

S4 class for the jump diffusion process

Slots

`theta`

`phi` ...

partFiltering *Estimation function*

Description

Bayesian estimation of the parameter of the model $Y_i = X_{t_i} + \epsilon_i$ with $dX_t = b(\phi, t, X_t)dt + s(\gamma, t, X_t)dW_t$.

Usage

```
partFiltering(t, y, prior, start, est = c("Bayes", "MLE"), len = 1000,
  sigmaTilde, Npart = 10, y0.fun = 1, b.fun = 1, parPropPhi = 5,
  maxIt = 10)
```

Arguments

`t`	vector of time points
`y`	vector of observation variables
`prior`	list of prior values
`start`	list of starting values
`est`	one out of "Bayes" or "MLE"
`len`	length of Markov chain
`sigmaTilde`	variance function $s(\gamma, t, y) = \gamma sigmaTilde(t, y)$
`Npart`	number of particles
`y0.fun`	function for starting point dependent on ϕ
`b.fun`	drift function
`parPropPhi`	parameter for proposal standard deviation
`maxIt`	maximal iteration of MH step of ϕ

Value

`phi`	Markov chain of ϕ
`gamma2`	Markov chain of $\gamma 2$
`sigma2`	Markov chain of $\sigma 2$
`X`	filtered process

Author(s)

Simone Hermann and Adeline Leclercq-Samson

`partFiltering_mixed` *Estimation function*

Description

Bayesian estimation of the parameter of the model $Y_{ij} = X_{t_{ij}} + \epsilon_{ij}$ with $dX_t = b(\phi_j, t, X_t)dt + s(\gamma, t, X_t)dW_t$.

Usage

```
partFiltering_mixed(t, y, prior, start, len = 1000, sigmaTilde, y0.fun = 1,
  b.fun = 1, Npart = 10, parPropPhi = 5, maxIt = 10)
```

Arguments

`t`	vector of time points
`y`	matrix of observation variables
`prior`	list of prior values
`start`	list of starting values
`len`	length of Markov chain

`sigmaTilde`	variance function $s(\gamma, t, y) = \gamma sigmaTilde(t, y)$
`y0.fun`	function for starting point dependent on ϕ
`b.fun`	drift function
`Npart`	number of particles
`parPropPhi`	parameter for proposal standard deviation
`maxIt`	maximal iteration of MH step of ϕ

Value

`phi`	Markov chain of ϕ
`mu`	Markov chain of μ
`Omega`	Markov chain of Ω
`gamma2`	Markov chain of γ^2
`sigma2`	Markov chain of σ^2
`X`	filtered process

Author(s)

Simone Hermann and Adeline Leclercq-Samson

`phi_ij` *Helping function*

Description

Returns the ij th matrix entry from a list

Usage

```
phi_ij(phi, i, j)
```

Arguments

`phi`	list with each entry a matrix
`i`	row
`j`	column

Value

vector of samples

plot,est.Diffusion-method

Plot method for the Bayesian estimation class object

Description

Plot method for the S4 class est.Diffusion

Usage

```
## S4 method for signature 'est.Diffusion'
plot(x, par.options, style = c("chains", "acf",
  "density"), par2plot, reduced = TRUE, thinning, burnIn, priorMeans = TRUE,
  col.priorMean = 2, lty.priorMean = 1, newwindow = FALSE, ...)
```

Arguments

`x`	est.Diffusion class
`par.options`	list of options for function par()
`style`	one out of "chains", "acf", "density"
`par2plot`	logical vector, which parameters to be plotted, order: (ϕ, γ^2)
`reduced`	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
`thinning`	thinning rate, if missing, the proposed one by the estimation procedure is taken
`burnIn`	burn-in phase, if missing, the proposed one by the estimation procedure is taken
`priorMeans`	logical(1), if TRUE (default), prior means are marked with a line
`col.priorMean`	color of the prior mean line, default 2
`lty.priorMean`	linetype of the prior mean line, default 1
`newwindow`	logical(1), if TRUE, a new window is opened for the plot
`...`	optional plot parameters

Examples

```
model <- set.to.class("Diffusion", b.fun = function(phi, t, y) phi[1]-phi[2]*y,
    parameter = list(phi = c(10, 1), gamma2 = 0.1))
data <- simulate(model, t = seq(0, 1, by = 0.01), y0 = 0.5, plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.01), data, 1000)  # nMCMC small for example
plot(est)
plot(est, burnIn = 100, thinning = 2)
plot(est, reduced = FALSE, par.options = list(mar = c(5, 4.5, 4, 2) + 0.1, mfrow = c(3,1)), xlab = "iteration"
plot(est, style = "acf", main = "", par2plot = c(TRUE, TRUE, FALSE))
plot(est, style = "density", lwd = 2, priorMean = FALSE)
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), main = "", par2plot = c(FALSE, FALSE, TRUE))
plot(est, priorMeans = FALSE, newwindow = TRUE)
```

`plot,est.hiddenDiffusion-method` *Plot method for the Bayesian estimation class object*

Description

Plot method for the S4 class est.hiddenDiffusion

Usage

```
## S4 method for signature 'est.hiddenDiffusion'
plot(x, par.options, style = c("chains",
  "acf", "density"), par2plot, reduced = TRUE, thinning, burnIn,
  priorMeans = TRUE, col.priorMean = 2, lty.priorMean = 1,
  newwindow = FALSE, ...)
```

Arguments

`x`	est.hiddenDiffusion class
`par.options`	list of options for function par()
`style`	one out of "chains", "acf", "density"
`par2plot`	logical vector, which parameters to be plotted, order: $(\phi, \gamma^2, \sigma^2, Y)$
`reduced`	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
`thinning`	thinning rate, if missing, the proposed one by the estimation procedure is taken
`burnIn`	burn-in phase, if missing, the proposed one by the estimation procedure is taken
`priorMeans`	logical(1), if TRUE (default), prior means are marked with a line
`col.priorMean`	color of the prior mean line, default 2
`lty.priorMean`	linetype of the prior mean line, default 1
`newwindow`	logical(1), if TRUE, a new window is opened for the plot
`...`	optional plot parameters

Examples

```
model <- set.to.class("hiddenDiffusion", b.fun = function(phi, t, y) phi[1]-phi[2]*y,
    parameter = list(phi = c(10, 1), gamma2 = 1, sigma2 = 0.1),
    y0 = function(phi, t) 0.5)
data <- simulate(model, t = seq(0, 1, by = 0.01), plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.01), data$Y, 100)  # nMCMC small for example
plot(est)
plot(est, par2plot = c(rep(FALSE, 3), TRUE, FALSE), ylim = c(0.001, 0.1), par.options = list())
plot(est, burnIn = 10, thinning = 2)
plot(est, reduced = FALSE, par.options = list(mar = c(5, 4.5, 4, 2) + 0.1, mfrow = c(3,1)), xlab = "iteration"
plot(est, style = "acf", main = "", par2plot = c(TRUE, TRUE, FALSE, FALSE))
plot(est, style = "density", lwd = 2, priorMean = FALSE)
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), main = "", par2plot = c(FALSE, FALSE, TRUE, TRUE))
plot(est, priorMeans = FALSE, newwindow = TRUE)
```

plot,est.hiddenmixedDiffusion-method

Plot method for the Bayesian estimation class object

Description

Plot method for the S4 class est.hiddenmixedDiffusion

Usage

```
## S4 method for signature 'est.hiddenmixedDiffusion'
plot(x, par.options, style = c("chains",
  "acf", "density", "int.phi"), par2plot, reduced = TRUE, thinning, burnIn,
  priorMeans = TRUE, col.priorMean = 2, lty.priorMean = 1, level = 0.05,
  phi, newwindow = FALSE, ...)
```

Arguments

`x`	est.hiddenmixedDiffusion class
`par.options`	list of options for function par()
`style`	one out of "chains", "acf", "density", "int.phi"
`par2plot`	logical vector, which parameters to be plotted, order: $(\mu, \Omega, \gamma^2, \sigma^2, Y)$
`reduced`	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
`thinning`	thinning rate, if missing, the proposed one by the estimation procedure is taken
`burnIn`	burn-in phase, if missing, the proposed one by the estimation procedure is taken
`priorMeans`	logical(1), if TRUE (default), prior means are marked with a line
`col.priorMean`	color of the prior mean line, default 2
`lty.priorMean`	linetype of the prior mean line, default 1
`level`	level for style = "int.phi"
`phi`	in the case of simulation study: known values for phi
`newwindow`	logical(1), if TRUE, a new window is opened for the plot
`...`	optional plot parameters

Examples

```
## Not run:
mu <- c(10, 3, 1); Omega = c(1, 0.4, 0.01)
phi <- sapply(1:3, function(i) rnorm(20, mu[i], sqrt(Omega[i])))
model <- set.to.class("hiddenmixedDiffusion", b.fun = function(phi, t, y) phi[1]-phi[2]*y,
    parameter = list(mu = mu, Omega = Omega, phi = phi, gamma2 = 1, sigma2 = 0.1),
    y0 = function(phi, t) phi[3])
data <- simulate(model, t = seq(0, 1, by = 0.02), plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.02), data$Z, 1000)
plot(est, newwindow = TRUE)
plot(est, burnIn = 10, thinning = 2)
plot(est, reduced = FALSE, par.options = list(mar = c(5, 4.5, 4, 2) + 0.1, mfrow = c(2,1)), xlab = "iteration"
plot(est, style = "acf", main = "", par2plot = c(TRUE, TRUE, rep(FALSE, 7)))
plot(est, style = "density", lwd = 2, priorMean = FALSE, par2plot = c(rep(FALSE, 6), TRUE, TRUE, FALSE))
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), main = "", par2plot = c(rep(FALSE, 6), TRUE, TRUE))
plot(est, priorMeans = FALSE, newwindow = TRUE)
plot(est, style = "int.phi", phi = phi, par2plot = c(TRUE, FALSE, FALSE))

## End(Not run)
```

`plot,est.jumpDiffusion-method` *Plot method for the Bayesian estimation class object*

Description

Plot method for the S4 class est.jumpDiffusion

Usage

```
## S4 method for signature 'est.jumpDiffusion'
plot(x, par.options, style = c("chains", "acf",
  "density"), par2plot, reduced = TRUE, thinning, burnIn, priorMeans = TRUE,
  col.priorMean = 2, lty.priorMean = 1, newwindow = FALSE, ...)
```

Arguments

`x`	est.jumpDiffusion class
`par.options`	list of options for function par()
`style`	one out of "chains", "acf", "density"
`par2plot`	logical vector, which parameters to be plotted, order: $(\phi, \theta, \gamma^2, \xi, N)$
`reduced`	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
`thinning`	thinning rate, if missing, the proposed one by the estimation procedure is taken
`burnIn`	burn-in phase, if missing, the proposed one by the estimation procedure is taken
`priorMeans`	logical(1), if TRUE (default), prior means are marked with a line
`col.priorMean`	color of the prior mean line, default 2
`lty.priorMean`	linetype of the prior mean line, default 1
`newwindow`	logical(1), if TRUE, a new window is opened for the plot
`...`	optional plot parameters

Examples

```
model <- set.to.class("jumpDiffusion", Lambda = function(t, xi) (t/xi[2])^xi[1],
parameter = list(theta = 0.1, phi = 0.05, gamma2 = 0.1, xi = c(3, 1/4)))
data <- simulate(model, t = seq(0, 1, by = 0.01), y0 = 0.5, plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.01), data, 1000)  # nMCMC small for example
plot(est)
plot(est, burnIn = 100, thinning = 2)
plot(est, reduced = FALSE, par.options = list(mar = c(5, 4.5, 4, 2) + 0.1, mfrow = c(2, 3)), xlab = "iteration
# plot only for phi and xi ...
plot(est, style = "acf", main = "", par2plot = c(TRUE, FALSE, FALSE, TRUE, TRUE))
plot(est, style = "density", lwd = 2, priorMean = FALSE)
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), par2plot = c(TRUE, rep(FALSE, 4)), main = "")
plot(est, priorMeans = FALSE, newwindow = TRUE)
```

`plot,est.Merton-method` *Plot method for the Bayesian estimation class object*

Description

Plot method for the S4 class est.Merton

Usage

```
## S4 method for signature 'est.Merton'
plot(x, par.options, style = c("chains", "acf",
  "density"), par2plot, reduced = TRUE, thinning, burnIn, priorMeans = TRUE,
  col.priorMean = 2, lty.priorMean = 1, newwindow = FALSE, ...)
```

Arguments

`x`	est.Merton class
`par.options`	list of options for function par()
`style`	one out of "chains", "acf", "density"
`par2plot`	logical vector, which parameters to be plotted, order: $(\phi, \widetilde{\theta}, \gamma^2, \xi, N)$
`reduced`	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
`thinning`	thinning rate, if missing, the proposed one by the estimation procedure is taken
`burnIn`	burn-in phase, if missing, the proposed one by the estimation procedure is taken
`priorMeans`	logical(1), if TRUE (default), prior means are marked with a line
`col.priorMean`	color of the prior mean line, default 2
`lty.priorMean`	linetype of the prior mean line, default 1
`newwindow`	logical(1), if TRUE, a new window is opened for the plot
`...`	optional plot parameters

Examples

```
model <- set.to.class("Merton", Lambda = function(t, xi) (t/xi[2])^xi[1],
parameter = list(thetaT = 0.1, phi = 0.05, gamma2 = 0.1, xi = c(3, 1/4)))
data <- simulate(model, t = seq(0, 1, by = 0.01), y0 = 0.5, plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.01), data, 1000)  # nMCMC small for example
plot(est)
plot(est, burnIn = 100, thinning = 2)
plot(est, reduced = FALSE, par.options = list(mar = c(5, 4.5, 4, 2) + 0.1, mfrow = c(2, 3)), xlab = "iteration
# plot only for phi and xi ...
plot(est, style = "acf", main = "", par2plot = c(TRUE, FALSE, FALSE, TRUE, TRUE))
plot(est, style = "density", lwd = 2, priorMean = FALSE)
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), par2plot = c(TRUE, rep(FALSE, 4)), main = "")
plot(est, priorMeans = FALSE, newwindow = TRUE)
```

`plot,est.mixedDiffusion-method` *Plot method for the Bayesian estimation class object*

Description

Plot method for the S4 class est.mixedDiffusion

Usage

```
## S4 method for signature 'est.mixedDiffusion'
plot(x, par.options, style = c("chains", "acf",
  "density", "int.phi"), par2plot, reduced = TRUE, thinning, burnIn,
  priorMeans = TRUE, col.priorMean = 2, lty.priorMean = 1, level = 0.05,
  phi, newwindow = FALSE, ...)
```

Arguments

`x`	est.mixedDiffusion class
`par.options`	list of options for function par()
`style`	one out of "chains", "acf", "density", "int.phi"
`par2plot`	logical vector, which parameters to be plotted, order: (μ, Ω, γ^2)
`reduced`	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
`thinning`	thinning rate, if missing, the proposed one by the estimation procedure is taken
`burnIn`	burn-in phase, if missing, the proposed one by the estimation procedure is taken
`priorMeans`	logical(1), if TRUE (default), prior means are marked with a line
`col.priorMean`	color of the prior mean line, default 2
`lty.priorMean`	linetype of the prior mean line, default 1
`level`	level for style = "int.phi"
`phi`	in the case of simulation study: known values for phi
`newwindow`	logical(1), if TRUE, a new window is opened for the plot
`...`	optional plot parameters

Examples

```
mu <- c(10, 3, 1); Omega = c(1, 0.4, 0.01)
phi <- sapply(1:3, function(i) rnorm(20, mu[i], sqrt(Omega[i])))
model <- set.to.class("mixedDiffusion", b.fun = function(phi, t, y) phi[1]-phi[2]*y,
    parameter = list(mu = mu, Omega = Omega, phi = phi, gamma2 = 0.1),
    y0 = function(phi, t) phi[3], sT.fun = function(t, x) sqrt(abs(x)))
data <- simulate(model, t = seq(0, 1, by = 0.02), plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.02), data, 100)  # nMCMC small for example
plot(est, newwindow = TRUE)
plot(est, burnIn = 10, thinning = 2)
plot(est, reduced = FALSE, par.options = list(mar = c(5, 4.5, 4, 2) + 0.1, mfrow = c(2,1)), xlab = "iteration"
plot(est, style = "acf", main = "")
plot(est, style = "density", lwd = 2, priorMean = FALSE)
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), main = "", par2plot = c(rep(FALSE, 6), TRUE))
plot(est, priorMeans = FALSE, newwindow = TRUE)
plot(est, style = "int.phi", phi = phi, par2plot = c(TRUE, FALSE, FALSE))
```

---

plot,est.mixedRegression-method

Plot method for the Bayesian estimation class object

Description

Plot method for the S4 class est.mixedRegression

Usage

```
## S4 method for signature 'est.mixedRegression'
plot(x, par.options, style = c("chains",
  "acf", "density", "int.phi"), par2plot, reduced = TRUE, thinning, burnIn,
  priorMeans = TRUE, col.priorMean = 2, lty.priorMean = 1, level = 0.05,
  phi, newwindow = FALSE, ...)
```

Arguments

`x`	est.mixedRegression class
`par.options`	list of options for function par()
`style`	one out of "chains", "acf", "density", "int.phi"
`par2plot`	logical vector, which parameters to be plotted, order: (μ, Ω, γ^2)
`reduced`	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
`thinning`	thinning rate, if missing, the proposed one by the estimation procedure is taken
`burnIn`	burn-in phase, if missing, the proposed one by the estimation procedure is taken
`priorMeans`	logical(1), if TRUE (default), prior means are marked with a line
`col.priorMean`	color of the prior mean line, default 2
`lty.priorMean`	linetype of the prior mean line, default 1
`level`	level for style = "int.phi"
`phi`	in the case of simulation study: known values for phi
`newwindow`	logical(1), if TRUE, a new window is opened for the plot
`...`	optional plot parameters

Examples

```
mu <- c(1, 3); Omega = c(0.4, 0.01)
phi <- sapply(1:2, function(i) rnorm(20, mu[i], sqrt(Omega[i])))
model <- set.to.class("mixedRegression", fun = function(phi, t) phi[1]*t + phi[2],
    parameter = list(mu = mu, Omega = Omega, phi = phi, gamma2 = 0.1))
data <- simulate(model, t = seq(0, 1, by = 0.02), plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.02), data, 100)  # nMCMC small for example
plot(est, newwindow = TRUE)
plot(est, burnIn = 10, thinning = 2)
plot(est, reduced = FALSE, par.options = list(mar = c(5, 4.5, 4, 2) + 0.1, mfrow = c(2,1)), xlab = "iteration"
plot(est, style = "acf", main = "")
plot(est, style = "density", lwd = 2, priorMean = FALSE)
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), main = "", par2plot = c(rep(FALSE, 4), TRUE))
plot(est, priorMeans = FALSE, newwindow = TRUE)
plot(est, style = "int.phi", phi = phi, par2plot = c(TRUE, FALSE))
```

---

plot,est.NHPP-method *Plot method for the Bayesian estimation class object*

Description

Plot method for the S4 class est.NHPP

Usage

```
## S4 method for signature 'est.NHPP'
plot(x, par.options, style = c("chains", "acf",
  "density"), par2plot, reduced = TRUE, thinning, burnIn, priorMeans = TRUE,
  col.priorMean = 2, lty.priorMean = 1, newwindow = FALSE, ...)
```

Arguments

x	est.NHPP class
par.options	list of options for function par()
style	one out of "chains", "acf", "density"
par2plot	logical vector, which parameters to be plotted, order: $(\phi, \theta, \gamma^2, \xi, N)$
reduced	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
thinning	thinning rate, if missing, the proposed one by the estimation procedure is taken
burnIn	burn-in phase, if missing, the proposed one by the estimation procedure is taken
priorMeans	logical(1), if TRUE (default), prior means are marked with a line
col.priorMean	color of the prior mean line, default 2
lty.priorMean	linetype of the prior mean line, default 1
newwindow	logical(1), if TRUE, a new window is opened for the plot
...	optional plot parameters

Examples

```
model <- set.to.class("NHPP", parameter = list(xi = c(5, 1/2)),
  Lambda = function(t, xi) (t/xi[2])^xi[1])
data <- simulate(model, t = seq(0, 1, by = 0.01), plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.01), data$Times, 10000)  # nMCMC small for example
plot(est)
plot(est, burnIn = 1000, thinning = 20)
plot(est, reduced = FALSE, xlab = "iteration")
plot(est, style = "acf", main = "", par2plot = c(TRUE, FALSE), par.options = list(mfrow = c(1, 1)))
plot(est, style = "density", lwd = 2, priorMean = FALSE)
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), par2plot = c(FALSE, TRUE), main = "")
plot(est, priorMeans = FALSE, newwindow = TRUE)
```

plot,est.Regression-method

Plot method for the Bayesian estimation class object

Description

Plot method for the S4 class est.Regression

Usage

```
## S4 method for signature 'est.Regression'
plot(x, par.options, style = c("chains", "acf",
  "density"), par2plot, reduced = TRUE, thinning, burnIn, priorMeans = TRUE,
  col.priorMean = 2, lty.priorMean = 1, newwindow = FALSE, ...)
```

Arguments

`x`	est.Regression class
`par.options`	list of options for function par()
`style`	one out of "chains", "acf", "density"
`par2plot`	logical vector, which parameters to be plotted, order: (ϕ, γ^2)
`reduced`	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
`thinning`	thinning rate, if missing, the proposed one by the estimation procedure is taken
`burnIn`	burn-in phase, if missing, the proposed one by the estimation procedure is taken
`priorMeans`	logical(1), if TRUE (default), prior means are marked with a line
`col.priorMean`	color of the prior mean line, default 2
`lty.priorMean`	linetype of the prior mean line, default 1
`newwindow`	logical(1), if TRUE, a new window is opened for the plot
`...`	optional plot parameters

Examples

```
model <- set.to.class("Regression", fun = function(phi, t) phi[1]*t + phi[2],
    parameter = list(phi = c(1, 2), gamma2 = 0.1))
data <- simulate(model, t = seq(0, 1, by = 0.01), plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.01), data, 1000)  # nMCMC small for example
plot(est)
plot(est, burnIn = 100, thinning = 2)
plot(est, reduced = FALSE, par.options = list(mar = c(5, 4.5, 4, 2) + 0.1, mfrow = c(3,1)), xlab = "iteration"
plot(est, style = "acf", main = "", par2plot = c(TRUE, TRUE, FALSE))
plot(est, style = "density", lwd = 2, priorMean = FALSE)
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), main = "", par2plot = c(FALSE, FALSE, TRUE))
plot(est, priorMeans = FALSE, newwindow = TRUE)
```

`plot,est.reg_hiddenNHPP-method` *Plot method for the Bayesian estimation class object*

Description

Plot method for the S4 class est.reg_hiddenNHPP

Usage

```
## S4 method for signature 'est.reg_hiddenNHPP'
plot(x, par.options, style = c("chains", "acf",
  "density"), par2plot, reduced = TRUE, thinning, burnIn, priorMeans = TRUE,
  col.priorMean = 2, lty.priorMean = 1, newwindow = FALSE, ...)
```

Arguments

`x`	est.reg_hiddenNHPP class
`par.options`	list of options for function par()
`style`	one out of "chains", "acf", "density"
`par2plot`	logical vector, which parameters to be plotted, order: $(\phi, \theta, \gamma^2, \xi, N)$
`reduced`	logical (1), if TRUE, the chains are thinned and burn-in phase is dropped
`thinning`	thinning rate, if missing, the proposed one by the estimation procedure is taken
`burnIn`	burn-in phase, if missing, the proposed one by the estimation procedure is taken
`priorMeans`	logical(1), if TRUE (default), prior means are marked with a line
`col.priorMean`	color of the prior mean line, default 2
`lty.priorMean`	linetype of the prior mean line, default 1
`newwindow`	logical(1), if TRUE, a new window is opened for the plot
`...`	optional plot parameters

Examples

```
model <- set.to.class("reg_hiddenNHPP", fun = function(t, N, theta) exp(theta[1]*t) + theta[2]*N,
  parameter = list(theta = c(2, 2), gamma2 = 0.25, xi = c(3, 0.5)),
  Lambda = function(t, xi) (t/xi[2])^xi[1])
data <- simulate(model, t = seq(0, 1, by = 0.01), plot.series = TRUE)
est <- estimate(model, t = seq(0, 1, by = 0.01), data, 1000)  # nMCMC small for example
plot(est)
plot(est, burnIn = 100, thinning = 2)
plot(est, reduced = FALSE, par.options = list(mar = c(5, 4.5, 4, 2) + 0.1, mfrow = c(2, 3)), xlab = "iteration
plot(est, style = "acf", main = "", par2plot = c(TRUE, FALSE, FALSE, TRUE, TRUE))
plot(est, style = "density", lwd = 2, priorMean = FALSE)
plot(est, style = "density", col.priorMean = 1, lty.priorMean = 2, main = "posterior")
plot(est, style = "acf", par.options = list(), par2plot = c(TRUE, rep(FALSE, 4)), main = "")
plot(est, priorMeans = FALSE, newwindow = TRUE)
```

plotQuantiles *Plot function for credibility or prediction intervals*

Description

Plots intervals.

Usage

```
plotQuantiles(input, xlab = "", ylab = "", main = "", l = 0.025,
  u = 0.975, color = 1)
```

Arguments

`input`	list or matrix of samples from posterior or predictive distribution
`xlab`	a title for the x axis
`ylab`	a title for the y axis
`main`	an overall title for the plot
`l`	lower bound
`u`	upper bound
`color`	color of the lines to be drawn

postmu *Posterior for mu*

Description

Posterior for parameters μ

Usage

```
postmu(phi, m, v, Omega)
```

Arguments

`phi`	matrix of random effects
`m`	prior mean
`v`	prior variance
`Omega`	variance of the random effects

Value

one sample of posterior

postOmega *Posterior*

Description

Posterior for parameters Ω

Usage

```
postOmega(alpha, beta, phi, mu)
```

Arguments

alpha	vector of prior variables
beta	vector of prior variables
phi	matrix of random effects
mu	mean of random effects

Value

one sample of posterior

postOmega_matrix *Posterior*

Description

Posterior for parameters Ω

Usage

```
postOmega_matrix(R, phi, mu)
```

Arguments

R	prior matrix of wishart distribution
phi	matrix of random effects
mu	mean of random effects

Value

one sample of posterior

pred.base *Prediction function*

Description

Bayesian prediction of the parameter of a stochastic process $dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t + h(\eta, t, Y_t)dN_t$.

Usage

```
pred.base(samples, VFun, dens, len = 100, x0, method = c("vector", "free"),
  pred.alg = c("Distribution", "Trajectory"), sampling.alg = c("RejSamp",
  "BinSearch"), candArea, grid = 0.001)
```

Arguments

samples	MCMC samples
VFun	cumulative distribution function
dens	density function
len	number of samples to be drawn
x0	starting point
method	vectorial ("vector") or not ("free")
pred.alg	prediction algorithm, "Distribution" or "Trajectory"
sampling.alg	sampling algorithm, rejection sampling ("RejSamp") or inversion method ("BinSearch")
candArea	candidate area
grid	fineness degree

Value

vector of samples from prediction

predict,est.Diffusion-method *Prediction for diffusion process*

Description

Bayesian prediction of a stochastic process $dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t$.

Usage

```
## S4 method for signature 'est.Diffusion'
predict(object, t, Euler.interval = FALSE,
  level = 0.05, burnIn, thinning, b.fun.mat, which.series = c("new",
  "current"), y.start, M2pred = 10, cand.length = 1000,
  pred.alg = c("Distribution", "Trajectory", "simpleTrajectory",
  "simpleBayesTrajectory"), method = c("vector", "free"),
  sampling.alg = c("BinSearch", "RejSamp"), sample.length, grid,
  plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.Diffusion"
`t`	vector of time points to make predictions for
`Euler.interval`	if TRUE: simple prediction intervals with Euler are made (in one step each)
`level`	level of the prediction intervals
`burnIn`	burn-in period
`thinning`	thinning rate
`b.fun.mat`	matrix-wise definition of drift function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")
`y.start`	optional, if missing, first (which.series = "new") or last observation variable ("current") is taken
`M2pred`	optional, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`cand.length`	length of candidate samples (if method = "vector")
`pred.alg`	prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory"
`method`	vectorial ("vector") or not ("free")
`sampling.alg`	sampling algorithm, inversion method ("BinSearch") or rejection sampling ("RejSamp")
`sample.length`	length of samples to be drawn
`grid`	fineness degree of approximation
`plot.prediction`	if TRUE, result are plotted

Examples

```
model <- set.to.class("Diffusion", parameter = list(phi = 0.5, gamma2 = 0.01))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5, plot.series = TRUE)
est_diff <- estimate(model, t, data, 1000) # better: 10000
plot(est_diff)
## Not run:
pred_diff <- predict(est_diff, t = seq(0, 1, by = 0.1))
pred_diff <- predict(est_diff, b.fun.mat = function(phi, t, y) phi[,1])  # much faster
pred_diff2 <- predict(est_diff, which.series = "current", b.fun.mat = function(phi, t, y) phi[,1])
pred_diff3 <- predict(est_diff, which.series = "current", y.start = data[51],
               t = t[seq(51, 100, by = 5)], b.fun.mat = function(phi, t, y) phi[,1])

## End(Not run)
pred_diff <- predict(est_diff, Euler.interval = TRUE, b.fun.mat = function(phi, t, y) phi[,1])
# one step Euler approximation
pred_diff <- predict(est_diff, pred.alg = "simpleTrajectory", sample.length = 100)
for(i in 1:100) lines(t[-1], pred_diff[i,], col = "grey")
pred_diff <- predict(est_diff, pred.alg = "simpleBayesTrajectory")
```

predict,est.hiddenDiffusion-method *Prediction for noisy / hidden diffusion process*

Description

Bayesian prediction of the model, $Z_i = Y_{t_i} + \epsilon_i, dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t$.

Usage

```
## S4 method for signature 'est.hiddenDiffusion'
predict(object, t, burnIn, thinning, b.fun.mat,
  which.series = c("new", "current"), M2pred = 10, cand.length = 1000,
  pred.alg = c("Distribution", "Trajectory", "simpleTrajectory",
  "simpleBayesTrajectory"), sample.length, grid, plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.hiddenDiffusion"
`t`	vector of time points to make predictions for
`burnIn`	burn-in period
`thinning`	thinning rate
`b.fun.mat`	matrix-wise definition of drift function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")
`M2pred`	optional, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`cand.length`	length of candidate samples (if method = "vector")
`pred.alg`	prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory"
`sample.length`	length of samples to be drawn
`grid`	fineness degree of approximation
`plot.prediction`	if TRUE, result are plotted

Examples

```
model <- set.to.class("hiddenDiffusion", parameter = list(phi = 5, gamma2 = 1, sigma2 = 0.1))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est_hiddiff <- estimate(model, t, data$Z, 100)  # nMCMC should be much larger!
plot(est_hiddiff)
## Not run:
pred_hiddiff <- predict(est_hiddiff, t = seq(0, 1, by = 0.1))
pred_hiddiff2 <- predict(est_hiddiff, which.series = "current")

## End(Not run)
pred_hiddiff <- predict(est_hiddiff, pred.alg = "simpleTrajectory", sample.length = 100)
pred_hiddiff <- predict(est_hiddiff, pred.alg = "simpleBayesTrajectory")
```

predict,est.hiddenmixedDiffusion-method *Prediction for noisy/hidden mixed diffusion process*

Description

Bayesian prediction of a stochastic process $Z_{ij} = Y_{t_{ij}} + \epsilon_{ij}, dY_t = b(\phi_j, t, Y_t)dt + s(\gamma, t, Y_t)dW_t, \phi_j\ N(\mu, \Omega)$.

Usage

```
## S4 method for signature 'est.hiddenmixedDiffusion'
predict(object, t, burnIn, thinning,
  b.fun.mat, which.series = c("new", "current"), ind.pred, M2pred = 10,
  cand.length = 1000, pred.alg = c("Distribution", "Trajectory",
  "simpleTrajectory", "simpleBayesTrajectory"), sample.length, grid,
  plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.hiddenmixedDiffusion"
`t`	vector of time points to make predictions for
`burnIn`	burn-in period
`thinning`	thinning rate
`b.fun.mat`	matrix-wise definition of drift function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")
`ind.pred`	index of series to be predicted, optional, if which.series = "current" and ind.pred missing, the last series is taken
`M2pred`	optional, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`cand.length`	length of candidate samples (if method = "vector")
`pred.alg`	prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory"
`sample.length`	length of samples to be drawn
`grid`	fineness degree of approximation
`plot.prediction`	if TRUE, result are plotted

Examples

```
mu <- c(5, 1); Omega <- c(0.9, 0.04)
phi <- cbind(rnorm(21, mu[1], sqrt(Omega[1])), rnorm(21, mu[2], sqrt(Omega[2])))
y0.fun <- function(phi, t) phi[2]
model <- set.to.class("hiddenmixedDiffusion", y0.fun = y0.fun,
     b.fun = function(phi, t, y) phi[1],
     parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 1, sigma2 = 0.01))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
```

```
## Not run:
est_hidmixdiff <- estimate(model, t, data$Z[1:20,], 2000)
plot(est_hidmixdiff)
pred1 <- predict(est_hidmixdiff, b.fun.mat = function(phi, t, y) phi[,1])
pred2 <- predict(est_hidmixdiff, pred.alg = "Trajectory", b.fun.mat = function(phi, t, y) phi[,1])
pred3 <- predict(est_hidmixdiff, pred.alg = "simpleTrajectory", sample.length = nrow(pred1$Y))
lines(t, apply(pred1$Z, 2, quantile, 0.025), col = 3)
lines(t, apply(pred1$Z, 2, quantile, 0.975), col = 3)
lines(t, apply(pred2$Z, 2, quantile, 0.025), col = 4)
lines(t, apply(pred2$Z, 2, quantile, 0.975), col = 4)
pred4 <- predict(est_hidmixdiff, pred.alg = "simpleBayesTrajectory")

## End(Not run)
```

`predict,est.jumpDiffusion-method`
Prediction for jump diffusion process

Description

Bayesian prediction of a stochastic process $dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t + h(\eta, t, Y_t)dN_t$.

Usage

```
## S4 method for signature 'est.jumpDiffusion'
predict(object, t, burnIn, thinning, Lambda.mat,
  which.series = c("new", "current"), M2pred = 10, cand.length = 1000,
  pred.alg = c("Trajectory", "Distribution", "simpleTrajectory",
  "simpleBayesTrajectory"), pred.alg.N = c("Trajectory", "Distribution"),
  candN = 0:5, sample.length, plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.jumpDiffusion"
`t`	vector of time points to make predictions for
`burnIn`	burn-in period
`thinning`	thinning rate
`Lambda.mat`	matrix-wise definition of intensity rate function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")
`M2pred`	default 10, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`cand.length`	length of candidate samples (if method = "vector"), for jump diffusion
`pred.alg`	prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory"
`pred.alg.N`	prediction algorithm, "Distribution", "Trajectory"
`candN`	vector of candidate area for differences of N, only if pred.alg.N = "Distribution"
`sample.length`	length of samples to be drawn
`plot.prediction`	if TRUE, result are plotted

Examples

```
model <- set.to.class("jumpDiffusion",
         parameter = list(theta = 0.1, phi = 0.05, gamma2 = 0.1, xi = c(3, 1/4)),
         Lambda = function(t, xi) (t/xi[2])^xi[1])
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5, plot.series = TRUE)
est_jd <- estimate(model, t, data, 2000)
plot(est_jd)
## Not run:
pred_jd <- predict(est_jd, Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
pred_jd2 <- predict(est_jd, pred.alg = "Distribution", pred.alg.N = "Distribution", Lambda.mat = function(t,
est <- estimate(model, t[1:81], data = list(N = data$N[1:81], Y = data$Y[1:81]), 2000)
pred <- predict(est, t = t[81:101], which.series = "current",
                     Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
lines(t, data$Y, type="l", lwd = 2)

## End(Not run)
pred_jd4 <- predict(est_jd, pred.alg = "simpleTrajectory", sample.length = 100)
for(i in 1:100) lines(t[-1], pred_jd4$Y[i,], col = "grey")
```

---

predict,est.Merton-method

Prediction for jump diffusion process

Description

Bayesian prediction of a stochastic process $Y_t = y_0 \exp(\phi t - \gamma 2/2t + \gamma W_t + \log(1+\theta)N_t)$.

Usage

```
## S4 method for signature 'est.Merton'
predict(object, t, burnIn, thinning, Lambda.mat,
  which.series = c("new", "current"), M2pred = 10, only.interval = TRUE,
  level = 0.05, cand.length = 1000, pred.alg = c("Distribution",
  "Trajectory", "simpleTrajectory", "simpleBayesTrajectory"), sample.length,
  plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.Merton"
`t`	vector of time points to make predictions for
`burnIn`	burn-in period
`thinning`	thinning rate
`Lambda.mat`	matrix-wise definition of intensity rate function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")
`M2pred`	default 10, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`only.interval`	if TRUE: only calculation of prediction intervals (only for pred.alg = "Distribution")

`level`	level of the prediction intervals
`cand.length`	length of candidate samples (if method = "vector"), for jump diffusion
`pred.alg`	prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory"
`sample.length`	length of samples to be drawn
`plot.prediction`	if TRUE, result are plotted

Examples

```
cl <- set.to.class("Merton",
               parameter = list(thetaT = 0.1, phi = 0.05, gamma2 = 0.1, xi = c(3, 1/4)),
               Lambda = function(t, xi) (t/xi[2])^xi[1])
t <- seq(0, 1, by = 0.01)
data <- simulate(cl, t = t, y0 = 0.5, plot.series = TRUE)
est <- estimate(cl, t, data, 1000)
plot(est)
## Not run:
pred1 <- predict(est, Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
pred2 <- predict(est, Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1], pred.alg = "Trajectory")
pred3 <- predict(est, pred.alg = "simpleTrajectory")
pred4 <- predict(est, pred.alg = "simpleBayesTrajectory")

## End(Not run)
```

---

predict,est.mixedDiffusion-method

Prediction for mixed diffusion process

Description

Bayesian prediction of a stochastic process $dY_t = b(\phi_j, t, Y_t)dt + s(\gamma, t, Y_t)dW_t, \phi_j\ N(\mu, \Omega)$.

Usage

```
## S4 method for signature 'est.mixedDiffusion'
predict(object, t, Euler.interval = FALSE,
  level = 0.05, burnIn, thinning, b.fun.mat, which.series = c("new",
  "current"), y.start, ind.pred, M2pred = 10, cand.length = 1000,
  pred.alg = c("Distribution", "Trajectory", "simpleTrajectory",
  "simpleBayesTrajectory"), sample.length, grid, plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.mixedDiffusion"
`t`	vector of time points to make predictions for
`Euler.interval`	if TRUE: simple prediction intervals with Euler are made (in one step each)
`level`	level of the prediction intervals
`burnIn`	burn-in period
`thinning`	thinning rate

`b.fun.mat`	matrix-wise definition of drift function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")
`y.start`	optional, if missing, first (which.series = "new") or last observation variable ("current") is taken
`ind.pred`	index of series to be predicted, optional, if which.series = "current" and ind.pred missing, the last series is taken
`M2pred`	optional, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`cand.length`	length of candidate samples (if method = "vector")
`pred.alg`	prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory"
`sample.length`	length of samples to be drawn
`grid`	fineness degree of approximation
`plot.prediction`	if TRUE, result are plotted

Examples

```
mu <- 2; Omega <- 0.4; phi <- matrix(rnorm(21, mu, sqrt(Omega)))
model <- set.to.class("mixedDiffusion",
         parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.1),
         b.fun = function(phi, t, x) phi*x, sT.fun = function(t, x) x)
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est_mixdiff <- estimate(model, t, data[1:20,], 100) # nMCMC should be much larger
plot(est_mixdiff)
## Not run:
pred_mixdiff <- predict(est_mixdiff, b.fun.mat = function(phi, t, y) phi[,1]*y)
lines(t, data[21,], lwd = 2)
mean(apply(pred_mixdiff$Y, 2, quantile, 0.025) <= data[21, ] &
apply(pred_mixdiff$Y, 2, quantile, 0.975) >= data[21, ])
mean(sapply(1:20, function(i){
   mean(apply(pred_mixdiff$Y, 2, quantile, 0.025) <= data[i, ] &
   apply(pred_mixdiff$Y, 2, quantile, 0.975) >= data[i, ])}))
pred_mixdiff2 <- predict(est_mixdiff, b.fun.mat = function(phi, t, y) phi[,1]*y,
     which.series = "current")
pred_mixdiff3 <- predict(est_mixdiff, b.fun.mat = function(phi, t, y) phi[,1]*y,
     which.series = "current", y.start = data[20, 51], t = t[51:101])

## End(Not run)
pred_mixdiff <- predict(est_mixdiff, Euler.interval = TRUE,
     b.fun.mat = function(phi, t, y) phi[,1]*y); lines(t, data[21,], lwd = 2)
     # one step Euler approximation
pred_mixdiff <- predict(est_mixdiff, pred.alg = "simpleTrajectory", sample.length = 100)
for(i in 1:100) lines(t, pred_mixdiff$Y[i,], col = "grey")
pred_mixdiff <- predict(est_mixdiff, pred.alg = "simpleBayesTrajectory")

# OU
## Not run:
b.fun <- function(phi, t, y) phi[1]-phi[2]*y; y0.fun <- function(phi, t) phi[3]
mu <- c(10, 1, 0.5); Omega <- c(0.9, 0.01, 0.01)
phi <- sapply(1:3, function(i) rnorm(21, mu[i], sqrt(Omega[i])))
```

```
cl <- set.to.class("mixedDiffusion",
           parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.1),
           y0.fun = y0.fun, b.fun = b.fun, sT.fun = function(t, x) 1)
t <- seq(0, 1, by = 0.01)
data <- simulate(cl, t = t, plot.series = TRUE)
est <- estimate(cl, t, data[1:20,], 2000)
plot(est)
pred_mixdiff <- predict(est, t = seq(0, 1, length = 21), b.fun.mat = function(phi, t, y) phi[,1]-phi[,2]*y)
lines(t, data[21,], lwd = 2)
mean(apply(pred_mixdiff$Y, 2, quantile, 0.025) <= data[21, seq(1, length(t), length = 21)] &
    apply(pred_mixdiff$Y, 2, quantile, 0.975) >= data[21, seq(1, length(t), length = 21)])
mean(sapply(1:20, function(i){
   mean(apply(pred_mixdiff$Y, 2, quantile, 0.025) <= data[i, seq(1, length(t), length = 21)] &
   apply(pred_mixdiff$Y, 2, quantile, 0.975) >= data[i, seq(1, length(t), length = 21)])}))

## End(Not run)
```

predict,est.mixedRegression-method

Prediction for mixed regression model

Description

Bayesian prediction of the regression model $y_i = f(\phi_j, t_i) + \epsilon_i, \phi_j ~ N(\mu, \Omega)$.

Usage

```
## S4 method for signature 'est.mixedRegression'
predict(object, t, only.interval = TRUE,
  level = 0.05, burnIn, thinning, fun.mat, which.series = c("new",
  "current"), ind.pred, M2pred = 10, cand.length = 1000, sample.length,
  grid, plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.mixedRegression"
`t`	vector of time points to make predictions for
`only.interval`	if TRUE: only calculation of prediction intervals
`level`	level of the prediction intervals
`burnIn`	burn-in period
`thinning`	thinning rate
`fun.mat`	matrix-wise definition of drift function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")
`ind.pred`	index of series to be predicted, optional, if which.series = "current" and ind.pred missing, the last series is taken
`M2pred`	optional, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`cand.length`	length of candidate samples (if method = "vector")

`sample.length`	length of samples to be drawn
`grid`	fineness degree of approximation
`plot.prediction`	if TRUE, result are plotted

Examples

```
mu <- c(10, 5); Omega <- c(0.9, 0.01)
phi <- cbind(rnorm(21, mu[1], sqrt(Omega[1])), rnorm(21, mu[2], sqrt(Omega[2])))
cl <- set.to.class("mixedRegression",
         parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.1),
         fun = function(phi, t) phi[1]*t + phi[2], sT.fun = function(t) 1)
t <- seq(0, 1, by = 0.01)
data <- simulate(cl, t = t, plot.series = TRUE)
est <- estimate(cl, t, data[1:20,], 2000)
plot(est)
pred <- predict(est, fun.mat = function(phi, t) phi[,1]*t + phi[,2])
points(t, data[21,], pch = 20)
```

`predict,est.NHPP-method`
Prediction for Poisson process

Description

Bayesian prediction of a nonhomogeneous Poisson process.

Usage

```
## S4 method for signature 'est.NHPP'
predict(object, variable = c("eventTimes",
  "PoissonProcess"), t, burnIn, thinning, Lambda.mat, which.series = c("new",
  "current"), Tstart, M2pred = 10, rangeN = c(0, 5), cand.length = 1000,
  pred.alg = c("Trajectory", "Distribution", "simpleTrajectory",
  "simpleBayesTrajectory"), sample.length, grid = 1e-05,
  plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.NHPP"
`variable`	if prediction of event times ("eventTimes") or of Poisson process variables ("PoissonProcess")
`t`	vector of time points to make predictions for (only for variable = "PoissonProcess")
`burnIn`	burn-in period
`thinning`	thinning rate
`Lambda.mat`	matrix-wise definition of drift function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")

`Tstart`	optional, if missing, first (which.series = "new") or last observation variable ("current") is taken
`M2pred`	optional, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`rangeN`	vector of candidate area for differences of N, only if pred.alg = "Distribution" and variable = "PoissonProcess"
`cand.length`	length of candidate samples (if method = "vector")
`pred.alg`	prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory"
`sample.length`	length of samples to be drawn
`grid`	fineness degress of approximation
`plot.prediction`	if TRUE, result are plotted

Examples

```
model <- set.to.class("NHPP", parameter = list(xi = c(5, 1/2)),
               Lambda = function(t, xi) (t/xi[2])^xi[1])
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est_NHPP <- estimate(model, t, data$Times, 1000)  # nMCMC should be much larger!
plot(est_NHPP)
pred <- predict(est_NHPP, Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1], variable = "PoissonProcess", pred.a
## Not run:
pred_NHPP <- predict(est_NHPP, Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
pred_NHPP <- predict(est_NHPP, variable = "PoissonProcess", Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
pred_NHPP2 <- predict(est_NHPP, which.series = "current", Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
pred_NHPP3 <- predict(est_NHPP, variable = "PoissonProcess", which.series = "current", Lambda.mat = function(
pred_NHPP4 <- predict(est_NHPP, pred.alg = "simpleTrajectory", M2pred = length(data$Times))

## End(Not run)
pred_NHPP <- predict(est_NHPP, variable = "PoissonProcess", pred.alg = "simpleTrajectory",
                     M2pred = length(data$Times))
pred_NHPP <- predict(est_NHPP, variable = "PoissonProcess", pred.alg = "simpleBayesTrajectory",
                     M2pred = length(data$Times), sample.length = 100)
```

`predict,est.Regression-method`
Prediction for regression model

Description

Bayesian prediction of regression model $y_i = f(\phi, t_i) + \epsilon_i$.

Usage

```
## S4 method for signature 'est.Regression'
predict(object, t, only.interval = TRUE,
  level = 0.05, burnIn, thinning, fun.mat, which.series = c("new",
  "current"), M2pred = 10, cand.length = 1000, method = c("vector",
  "free"), sampling.alg = c("BinSearch", "RejSamp"), sample.length, grid,
  plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.Regression"
`t`	vector of time points to make predictions for
`only.interval`	if TRUE: only calculation of prediction intervals
`level`	level of the prediction intervals
`burnIn`	burn-in period
`thinning`	thinning rate
`fun.mat`	matrix-wise definition of drift function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")
`M2pred`	optional, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`cand.length`	length of candidate samples (if method = "vector")
`method`	vectorial ("vector") or not ("free")
`sampling.alg`	sampling algorithm, inversion method ("BinSearch") or rejection sampling ("RejSamp")
`sample.length`	length of samples to be drawn
`grid`	fineness degree of approximation
`plot.prediction`	if TRUE, result are plotted

Examples

```
t <- seq(0,1, by = 0.01)
cl <- set.to.class("Regression", fun = function(phi, t) phi[1]*t + phi[2],
                   parameter = list(phi = c(1,2), gamma2 = 0.1))
data <- simulate(cl, t = t, plot.series = TRUE)
est <- estimate(cl, t, data, 1000)
plot(est)
pred <- predict(est, fun.mat = function(phi, t) phi[,1]*t + phi[,2])
## Not run:
pred2 <- predict(est, fun.mat = function(phi, t) phi[,1]*t + phi[,2], only.interval = FALSE)
plot(density(pred2[,10]))

## End(Not run)
```

`predict,est.reg_hiddenNHPP-method` *Prediction for regression model dependent on Poisson process*

Description

Bayesian prediction of a regression model $y_i = f(t_i, N_i, \theta) + \epsilon_i$.

Usage

```
## S4 method for signature 'est.reg_hiddenNHPP'
predict(object, t, only.interval = TRUE,
  level = 0.05, burnIn, thinning, Lambda.mat, fun.mat,
  which.series = c("new", "current"), M2pred = 10, cand.length = 1000,
  pred.alg = c("Distribution", "simpleTrajectory", "simpleBayesTrajectory"),
  sample.length, grid = 1e-05, plot.prediction = TRUE)
```

Arguments

`object`	class object of MCMC samples: "est.reg_hiddenNHPP"
`t`	vector of time points to make predictions for
`only.interval`	if TRUE: only calculation of prediction intervals
`level`	level of the prediction intervals
`burnIn`	burn-in period
`thinning`	thinning rate
`Lambda.mat`	matrix-wise definition of intensity rate function (makes it faster)
`fun.mat`	matrix-wise definition of regression function (makes it faster)
`which.series`	which series to be predicted, new one ("new") or further development of current one ("current")
`M2pred`	default 10, if current series to predicted and t missing, M2pred variables will be predicted with dt of observation time points
`cand.length`	length of candidate samples (if method = "vector"), for jump diffusion
`pred.alg`	prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleTrajectory"
`sample.length`	length of samples to be drawn
`grid`	fineness degress of approximation, for Poisson process
`plot.prediction`	if TRUE, result are plotted

Examples

```
t <- seq(0,1, by = 0.01)
cl <- set.to.class("reg_hiddenNHPP", fun = function(t, N, theta) theta[1]*t + theta[2]*N,
            parameter = list(theta = c(1,2), gamma2 = 0.1, xi = c(3, 1/4)),
            Lambda = function(t, xi) (t/xi[2])^xi[1])
data <- simulate(cl, t = t, plot.series = TRUE)
est <- estimate(cl, t, data, 1000)
plot(est)
## Not run:
pred <- predict(est, Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1],
                 fun.mat = function(t, N, theta) theta[,1]*t + theta[,2]*N)

## End(Not run)
pred <- predict(est, pred.alg = "simpleTrajectory", sample.length = 100)
```

`prediction.intervals` *Prediction interval function*

Description

Bayesian prediction of the parameter of a stochastic process $dY_t = b(phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t + h(\eta, t, Y_t)dN_t$.

Usage

```
prediction.intervals(samples, Fun, x0, level = 0.05, candArea, grid = 0.001)
```

Arguments

`samples`	MCMC samples
`Fun`	cumulative distribution function
`x0`	starting point
`level`	level of prediction intervals
`candArea`	candidate area
`grid`	fineness degree

Value

prediction interval

`predPhi` *Prediction Of The Random Effects In Mixed Stochastic Differential Equations*

Description

Prediction of the random effects ϕ $N(\mu, \Omega)$

Usage

```
predPhi(samples, cand)
```

Arguments

`samples`	output of estSDE or estREg
`cand`	candidates for phi (matrix with p columns)

Value

matrix phi

predReg *Bayesian prediction in mixed nonlinear regression models*

Description

Bayesian prediction in the mixed nonlinear regression model $y_{ij} = f(\phi_j, t_{ij}) + \epsilon_{ij}, \epsilon_{ij}\ N(0, \gamma^2 * s^2(t_{ij}), \phi_j\ N(\mu, \Omega)$.

Usage

```
predReg(t, samples, fODE, sVar, cand, len = 1000, mod = "Gompertz")
```

Arguments

`t`	vector of times which are predicted
`samples`	list of samples from the posterior
`fODE`	regression function
`sVar`	variance function
`cand`	vector of candidates for trajection sampling
`len`	number of samples from the predictive distribution
`mod`	model out of Gompertz, Richards, logistic, Weibull, only used instead of fODE

Value

matrix of predictions in t

proposal *Sampling from proposal density for strictly positive parameters*

Description

Used in Metropolis Hastings algorithms.

Usage

```
proposal(parOld, propSd)
```

Arguments

`parOld`	the parameter from the last iteration step
`propSd`	proposal standard deviation

Value

candidate for MH ratio

Examples

```
plot(replicate(100, proposal(1, 0.1)), type = "l")
```

`proposalRatio` *Sampling from proposal density for strictly positive parameters*

Description

Used in Metropolis Hastings algorithms.

Usage

```
proposalRatio(parOld, parNew, propSd)
```

Arguments

`parOld`	the parameter from the last iteration step
`parNew`	drawn candidate
`propSd`	proposal standard deviation

Value

MH ratio

Examples

```
cand <- proposal(1, 0.01)
proposalRatio(1, cand, 0.01)
```

`RejSampling` *Rejection Sampling Algorithm*

Description

Rejection Sampling

Usage

```
RejSampling(Fun, dens, len, cand, grid = 0.001, method = c("vector",
  "free"))
```

Arguments

`Fun`	cumulative distribution function
`dens`	density
`len`	number of samples
`cand`	candidate area
`grid`	fineness degree
`method`	vectorial ("vector") or not ("free")

Value

vector of samples

Examples

```
plot(density(RejSampling(Fun = function(x) pnorm(x, 5, 1), dens = function(x) dnorm(x, 5, 1), len = 500, cand
lines(density(RejSampling(function(x) pnorm(x, 5, 1), function(x) dnorm(x, 5, 1), 500, cand = seq(2, 9, by = 0
curve(dnorm(x, 5, 1), from = 2, to = 8, add = TRUE, col = 3)
```

---

scoreRule *Calculation of interval score*

Description

Scoring rule of Raftery and Gneiting (??).

Usage

```
scoreRule(l, u, x, alpha = 0.05)
```

Arguments

`l`	lower bound
`u`	upper bound
`x`	true value
`alpha`	level

Value

interval score

---

set.to.class *Builds classes*

Description

Defines classes

Usage

```
set.to.class(class.name = c("jumpDiffusion", "Merton", "Diffusion",
  "mixedDiffusion", "hiddenDiffusion", "hiddenmixedDiffusion", "reg_hiddenNHPP",
   "NHPP", "Regression", "mixedRegression"), parameter, prior, start, b.fun,
   s.fun, h.fun, sT.fun, y0.fun, fun, Lambda, priorRatio)
```

Arguments

`class.name`	name of model class
`parameter`	list of parameter values
`prior`	optional list of prior parameters
`start`	optional list of starting values
`b.fun`	drift function b
`s.fun`	variance function s
`h.fun`	jump high function h
`sT.fun`	variance function $\widetilde{s}$
`y0.fun`	function for the starting point, if dependent on parameter
`fun`	regression function
`Lambda`	intensity rate of Poisson process
`priorRatio`	list of functions for prior ratios, only for jumpDiffusion, is missing: non-informative estimation

Value

class

Examples

```
set.to.class("jumpDiffusion")
cl_jd <- set.to.class("jumpDiffusion", parameter = list(theta = 0.1, phi = 0.01, gamma2 = 0.1, xi = 3))
summary(class.to.list(cl_jd))
```

`simN` *Simulation of counting process*

Description

Simulation of counting process and event times.

Usage

```
simN(t, xi, len, start = c(0, 0), Lambda, int = c("Weibull", "Exp"))
```

Arguments

`t`	vector of times
`xi`	parameter vector ξ
`len`	number of samples to be drawn
`start`	vector: start[1] starting point time, start[2] starting point for Poisson process
`Lambda`	intensity rate function
`int`	if no Lambda: one out of "Weibull" or "Exp" for intensity function

Value

`N`	Poisson process
`Times`	event times

simulate,Diffusion-method

Simulation of diffusion process

Description

Simulation of a stochastic process $dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t$.

Usage

```
## S4 method for signature 'Diffusion'
simulate(object, nsim = 1, seed = NULL, t, y0,
  mw = 1, plot.series = TRUE)
```

Arguments

object	class object of parameters: "Diffusion"
nsim	number of response vectors to simulate. Defaults to 1
seed	optional: seed number for random number generator
t	vector of time points to make predictions for
y0	starting point of the process
mw	mesh width for finer Euler approximation
plot.series	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
model <- set.to.class("Diffusion", parameter = list(phi = 0.5, gamma2 = 0.01))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5, plot.series = TRUE)
```

simulate,hiddenDiffusion-method

Simulation of diffusion process

Description

Simulation of a hidden stochastic process $Z_i = Y_{t_i} + \epsilon_i, dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t$.

Usage

```
## S4 method for signature 'hiddenDiffusion'
simulate(object, nsim = 1, seed = NULL, t,
  mw = 10, plot.series = TRUE)
```

Arguments

`object`	class object of parameters: "hiddenDiffusion"
`nsim`	number of response vectors to simulate. Defaults to 1
`seed`	optional: seed number for random number generator
`t`	vector of time points to make predictions for
`mw`	mesh width for finer Euler approximation
`plot.series`	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
model <- set.to.class("hiddenDiffusion", parameter = list(phi = 0.5, gamma2 = 0.01, sigma2 = 0.1))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
```

`simulate,hiddenmixedDiffusion-method`
Simulation of hidden mixed diffusion process

Description

Simulation of a stochastic process $Z_{ij} = Y_{t_{ij}} + \epsilon_{ij}, dY_t = b(\phi_j, t, Y_t)dt + s(\gamma, t, Y_t)dW_t, \phi_j \, N(\mu, \Omega)$.

Usage

```
## S4 method for signature 'hiddenmixedDiffusion'
simulate(object, nsim = 1, seed = NULL, t,
  mw = 10, plot.series = TRUE)
```

Arguments

`object`	class object of parameters: "hiddenmixedDiffusion"
`nsim`	number of response vectors to simulate. Defaults to 1
`seed`	optional: seed number for random number generator
`t`	vector of time points to make predictions for
`mw`	mesh width for finer Euler approximation
`plot.series`	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
mu <- c(5, 1); Omega <- c(0.9, 0.04); phi <- cbind(rnorm(21, mu[1], sqrt(Omega[1])), rnorm(21, mu[2], sqrt(Om
y0.fun <- function(phi, t) phi[2]
model <- set.to.class("hiddenmixedDiffusion", y0.fun = y0.fun, b.fun = function(phi, t, y) phi[1], parameter
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t)
```

simulate,jumpDiffusion-method

Simulation of jump diffusion process

Description

Simulation of jump diffusion process $dY_t = b(\phi, t, Y_t)dt + s(\gamma, t, Y_t)dW_t + h(\eta, t, Y_t)dN_t$.

Usage

```
## S4 method for signature 'jumpDiffusion'
simulate(object, nsim = 1, seed = NULL, t, y0,
  mw = 1, plot.series = TRUE)
```

Arguments

`object`	class object of parameters: "jumpDiffusion"
`nsim`	number of response vectors to simulate. Defaults to 1
`seed`	optional: seed number for random number generator
`t`	vector of time points to make predictions for
`y0`	starting point of process
`mw`	mesh width for finer Euler approximation
`plot.series`	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
model <- set.to.class("jumpDiffusion", parameter = list(theta = 0.1, phi = 0.05, gamma2 = 0.1, xi = c(3, 1/4))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5)
```

simulate,Merton-method

Simulation of jump diffusion process

Description

Simulation of jump diffusion process $Y_t = y_0 \exp(\phi t - \gamma 2/2t + \gamma W_t + \log(1 + \theta)N_t)$.

Usage

```
## S4 method for signature 'Merton'
simulate(object, nsim = 1, seed = NULL, t, y0,
  plot.series = TRUE)
```

Arguments

`object`	class object of parameters: "Merton"
`nsim`	number of response vectors to simulate. Defaults to 1
`seed`	optional: seed number for random number generator
`t`	vector of time points to make predictions for
`y0`	starting point of process
`plot.series`	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
model <- set.to.class("Merton", parameter = list(thetaT = 0.1, phi = 0.05, gamma2 = 0.1, xi = 10))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5)
```

`simulate,mixedDiffusion-method`

Simulation of diffusion process

Description

Simulation of a stochastic process $dY_t = b(\phi_j, t, Y_t)dt + s(\gamma, t, Y_t)dW_t, \phi_j\ N(\mu, \Omega)$.

Usage

```
## S4 method for signature 'mixedDiffusion'
simulate(object, nsim = 1, seed = NULL, t,
  mw = 1, plot.series = TRUE)
```

Arguments

`object`	class object of parameters: "mixedDiffusion"
`nsim`	number of response vectors to simulate. Defaults to 1
`seed`	optional: seed number for random number generator
`t`	vector of time points to make predictions for
`mw`	mesh width for finer Euler approximation
`plot.series`	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
mu <- 2; Omega <- 0.4; phi <- matrix(rnorm(21, mu, sqrt(Omega)))
model <- set.to.class("mixedDiffusion", parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.1), y0.
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
```

simulate,mixedRegression-method *Simulation of mixed regression model*

Description

Simulation of regression model $y_i = f(\phi_j, t_i) + \epsilon_i, \phi_j \sim N(\mu, \Omega)$.

Usage

```
## S4 method for signature 'mixedRegression'
simulate(object, nsim = 1, seed = NULL, t,
  plot.series = TRUE)
```

Arguments

`object`	class object of parameters: "mixedRegression"
`nsim`	number of response vectors to simulate. Defaults = 1.
`seed`	optional: seed number for random number generator
`t`	vector of time points to make predictions for
`plot.series`	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
mu <- 2; Omega <- 0.4; phi <- matrix(rnorm(21, mu, sqrt(Omega)))
model <- set.to.class("mixedRegression", parameter = list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.1), fu
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
```

simulate,NHPP-method *Simulation of Poisson process*

Description

Simulation of Poisson process.

Usage

```
## S4 method for signature 'NHPP'
simulate(object, nsim = 1, seed = NULL, t,
  plot.series = TRUE)
```

Arguments

`object`	class object of parameters: "NHPP"
`nsim`	number of response vectors to simulate. Defaults to 1
`seed`	optional: seed number for random number generator
`t`	vector of time points to make predictions for
`plot.series`	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
model <- set.to.class("NHPP", parameter = list(xi = c(5, 1/2)), Lambda = function(t, xi) (t/xi[2])^xi[1])
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t)
```

`simulate,Regression-method`
Simulation of regression model

Description

Simulation of the regression model $y_i = f(\phi, t_i) + \epsilon_i$.

Usage

```
## S4 method for signature 'Regression'
simulate(object, nsim = 1, seed = NULL, t,
  plot.series = TRUE)
```

Arguments

`object`	class object of parameters: "Diffusion"
`nsim`	number of response vectors to simulate. Defaults to 1
`seed`	optional: seed number for random number generator
`t`	vector of time points to make predictions for
`plot.series`	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
model <- set.to.class("Regression", parameter = list(phi = 5, gamma2 = 0.1), fun = function(phi, t) phi*t)
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
```

`simulate,reg_hiddenNHPP-method`
Simulation of regression model dependent on Poisson process

Description

Simulation of of the regression model $y_i = f(t_i, N_i, \theta) + \epsilon_i$.

Usage

```
## S4 method for signature 'reg_hiddenNHPP'
simulate(object, nsim = 1, seed = NULL, t,
  plot.series = TRUE)
```

Arguments

`object`	class object of parameters: "reg_hiddenNHPP"
`nsim`	number of response vectors to simulate. Defaults to 1
`seed`	optional: seed number for random number generator
`t`	vector of time points to make predictions for
`plot.series`	logical(1), if TRUE, simulated series are depicted grafically

Examples

```
model <- set.to.class("reg_hiddenNHPP", fun = function(t, N, theta) theta[1]*t + theta[2]*N, parameter = list
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t)
```

simY *Simulation of Jump diffusion process*

Description

Simulation of Jump diffusion process.

Usage

```
simY(t, phi, thetaT, gamma2, start, N)
```

Arguments

`t`	vector of times
`phi`	parameter ϕ
`thetaT`	parameter $\widetilde{\theta}$
`gamma2`	parameter γ^2
`start`	starting point ofprocess y_0
`N`	Poisson process variables in t

Value

matrix or vector

`sim_JD_Euler`	*Simulation of Jump diffusion process*

Description

Simulation of Jump diffusion process.

Usage

```
sim_JD_Euler(t, phi, theta, gamma2, b.fun, s.fun, h.fun, start, N, mw = 1)
```

Arguments

`t`	vector of times
`phi`	parameter ϕ
`theta`	parameter θ
`gamma2`	parameter γ^2
`b.fun`	drift function
`s.fun`	variance function
`h.fun`	jump high function
`start`	starting point y_0
`N`	Poisson process variables in t
`mw`	mesh width for finer Euler approximation

Value

matrix or vector

`sim_reg_hiddenNHPP`	*Simulation of regression model including the NHPP*

Description

Simulation.

Usage

```
sim_reg_hiddenNHPP(t, N, fun, theta, gamma2)
```

Arguments

`t`	vector of times
`N`	vector of Poisson process
`fun`	regression function
`theta`	parameter θ
`gamma2`	parameterγ^2

Value

matrix or vector

TimestoN — *Transformation of vector of event times to counting process*

Description

Transformation of vector of event times to counting process.

Usage

```
TimestoN(times, t)
```

Arguments

`times`	vector of event times
`t`	times of counting process

Value

vector of counting process observations in t

Examples

```
t <- seq(0, 1, by = 0.01)
times <- simN(t, c(5, 0.5), len = 1)$Times
process <- TimestoN(times, t)
```

Index