# Package 'BaPreStoPro'

June 7, 2016

**Type** Package

**Title** Bayesian Prediction of Stochastic Processes

**Version** 0.1

**Date** 2016-06-07

**Author** Simone Hermann

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

**Description** Bayesian estimation and prediction for stochastic processes based on the Euler approximation. Considered processes are: jump diffusion, (mixed) diffusion models, hidden (mixed) diffusion models, non-homogeneous Poisson processes (NHPP), (mixed) regression models for comparison and a regression model including a NHPP.

**License** GPL (>= 2)

**Depends** stats, methods, graphics

**LazyData** TRUE

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2016-06-07 14:28:11

## R topics documented:

---

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

---

## Description

This package contains simulate, estimate and predict methods for non-homogeneous Poisson processes (NHPP), jump diffusions, (mixed) diffusions, hidden (mixed) diffusion models, regression model including a NHPP, and (mixed) regression models for comparison.

## Details

| | |
|---|---|
| Package: | BaPreStoPro |
| Type: | Package |
| Version: | 1.0 |
| Date: | 2016-06-07 |
| License: | GLP-2, GLP-3 |

Each of the models has its specific class, `"jumpDiffusion"`, `"Merton"`, `"Diffusion"`, `"mixedDiffusion"`, `"hiddenDiffusion"`, `"hiddenmixedDiffusion"`, `"jumpRegression"`, `"NHPP"`, `"Regression"`, `"mixedRegression"`, created with the function `set.to.class`. For each of the model classes, a method `simulate` and `estimate` are provided. The output of method `estimate` is a new class object with the prefix "est.". For the estimation classes, methods `plot` and `predict` are available. An overview of the package can be found in Hermann (2016a) and theoretical details to the prediction procedures in Hermann (2016b).

## jumpDiffusion

The model class `"jumpDiffusion"` contains all information about model defined by the stochastic differential equation (SDE) $dY_t = b(\phi, t, Y_t)dt + s(\gamma^2, t, Y_t)dW_t + h(\theta, t, Y_t)dN_t$ with $N_t \sim Pois(\Lambda(t, \xi))$ a non-homogeneous Poisson process and $W_t$ a Brownian motion. The SDE is approximated with the Euler Maruyama approximation, which leads, dependent on the Poisson process variables, to a normal likelihood. For more information how to build the class, see examples in `jumpDiffusion-class` and `estimate,jumpDiffusion-method`.

Estimation is done by a Metropolis-within-Gibbs sampler. For each of the parameters, a Metropolis-Hastings (MH) step is made, where the proposal density can be chosen between normal and log-normal. A proposal standard deviation can be chosen, which, if desired, is adapted after every 50 iterations, see Rosenthal (2011).

In the case of unobserved variables of the Poisson process, one step of the Gibbs sampler is filtering of the unobserved jump process. Details can be found in Hermann and Ruggeri (2016) or Hermann (2016a).

### Merton

Specific choices of functions $b$, $s$ and $h$ lead to an explicit solution of the process $Y_t = y_0 \exp(\phi t - \gamma^2/2t + \gamma W_t + \log(1+\theta)N_t)$. This model is well-known in the literature as Merton model. There are conjugate prior distributions available for $\phi, \gamma^2$ and $\log(1+\theta)$: $\phi$ and $\log(1+\theta)$ are assumed to have normal prior distribution with parameters `m.phi` (mean) and `v.phi` (variance), `m.thetaT` (mean) and `v.thetaT` (variance) respectively. $\gamma^2$ is assumed to have an inverse gamma distribution with parameters `alpha.gamma` and `beta.gamma`. An example how to build the model class can be found in `Merton-class` and `estimate,Merton-method`.

Estimation is similar to the jump diffusion process based on the Euler approximation. The difference is that for the parameters with conjugate priors, no MH step is necessary and drawing from the full conditional posterior is possible.

### Diffusion

The special case of $h(\theta, t, y) = 0$ leads to a general diffusion process $dY_t = b(\phi, t, Y_t)dt + s(\gamma^2, t, Y_t)dW_t$. We here restrict to the special case of $s(\gamma^2, t, y) = \sqrt{\gamma^2}\widetilde{s}(t, y)$, because a conjugate prior for $\gamma^2$ is available (the inverse gamma with parameters `alpha.gamma` and `beta.gamma`) in this case. For $\phi$, a normal prior with parameters `m.phi` (mean) and `v.phi` (variance) is assumed. An example can be found in `Diffusion-class` and `estimate,Diffusion-method`.

A Gibbs sampler with an MH step for $\phi$ and one step drawing from the full conditional of $\gamma^2$ is implemented. For $\phi$, the proposal density can be chosen, `"normal"` or `"lognormal"` and the proposal standard deviation, with the option to adapt, as well.

### mixedDiffusion

The diffusion process is extended to a hierarchical model with $\phi$ as random effect with normal mixture distribution with mean $\mu$ and variance $\Omega$ (diagonal matrix). For $\mu$, a normal prior (parameters: `m.mu` and `v.mu`) is conjugate. For each diagonal element of $\Omega$, an inverse gamma prior (parameters: `alpha.omega` and `beta.omega`) is conjugate. Further information can be found in `mixedDiffusion-class` and `estimate,mixedDiffusion-method`.

A Gibbs sampler with an MH step for each random effect and each one step drawing from the full conditionals of $\mu, \Omega$ and $\gamma^2$ is implemented.

### hiddenDiffusion

The same model as the diffusion above is taken, but with an added error: $Z_i = Y_{t_i} + \epsilon_i, dY_t = b(\phi, t, Y_t)dt + \gamma\widetilde{s}(t, Y_t)dW_t, \epsilon_i \sim N(0, \sigma^2), Y_{t_0} = y_0(\phi, t_0)$. The inverse gamma prior for $\sigma^2$ is conjugate and is, therefore, implemented with parameters `alpha.sigma` and `beta.sigma`. The diffusion process is a latent variable and has also to be estimated. A conditional sequential Monte Carlo (SMC) approach is implemented. For further details see Andrieu et al. (2010). Examples can be found in `hiddenDiffusion-class` and `estimate,hiddenDiffusion-method`

### hiddenmixedDiffusion

Here, the hidden diffusion model is extended to a hierarchical model with the random effect $\phi$, similar to the mixed diffusion model. In the Gibbs sampler, one step is filtering the unobserved diffusion process for each observed series. Based on this estimation, the random effects are estimated with an MH step. With conjugate priors, full conditionals for $\mu, \Omega, \gamma^2$ and $\sigma^2$ are available. Examples can be found in `hiddenmixedDiffusion-class` and `estimate,hiddenmixedDiffusion-method`.

### jumpRegression

We here consider the regression model: $y_i = f(t_i, N_{t_i}, \theta) + \epsilon_i$ dependent on the Poisson process $N_t \sim Pois(\Lambda(t, \xi))$ with $\epsilon_i \sim N(0, \gamma^2 \widetilde{s}(t))$.

Here, for the case of missing observations for the Poisson process variable, a filtering procedure based on the conditional SMC is implemented. But this is still work in progress.

See examples in `jumpRegression-class` and `estimate,jumpRegression-method`.

### NHPP

Some models base on the non-homogeneous Poisson process. Here, only the NHPP itself is considered. A simple MH algorithm is implemented. A proposal density can be chosen, normal or lognormal, and the proposal standard deviation as well, which is adapted, if desired.

See examples in `NHPP-class` and `estimate,NHPP-method`.

### Regression

For the case of a comparison of regression and diffusion model, as made, for example, in Hermann et al. (2016), estimation and prediction is also made for regression models. All notations are analogously to the diffusion model. See examples in `Regression-class` and `estimate,Regression-method`.

### mixedRegression

Analogous hierarchical regression model to the mixed diffusion model above. See examples in `mixedRegression-class` and `estimate,mixedRegression-method`.

### Author(s)

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

### References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

Hermann, S., K. Ickstadt and C. H. Mueller (2016). Bayesian Prediction of Crack Growth Based on a Hierarchical Diffusion Model. *Applied Stochastic Models in Business and Industry, DOI: 10.1002/asmb.2175*.

Hermann, S. and F. Ruggeri (2016). Modelling Wear in Cylinder Liners. SFB 823 discussion paper 06/16.

Hermann, S., K. Ickstadt, and C. H. Mueller (2015). Bayesian Prediction for a Jump Diffusion Process with Application to Crack Growth in Fatigue Experiments. SFB 823 discussion paper 30/15.

Heeke, G., S. Hermann, R. Maurer, K. Ickstadt, and C. H. Mueller (2015). Stochastic Modeling and Statistical Analysis of Fatigue Tests on Prestressed Concrete Beams under Cyclic Loadings. SFB 823 discussion paper 25/15.

Monte Carlo methods:

Robert, C. P. and G. Casella (2004). Monte Carlo Statistical Methods. Springer, New York.

Adaptive MCMC:

Rosenthal, J. S. (2011). Optimal Proposal Distributions and Adaptive MCMC. In: Handbook of Markov Chain Monte Carlo, pp. 93-112.

particel Gibbs / SMC:

Andrieu, C., A. Doucet and R. Holenstein (2010). Particle Markov Chain Monte Carlo Methods. Journal of the Royal Statistical Society B 72, pp. 269-342.

## Examples

```
model <- set.to.class("Diffusion", parameter = list(phi = 0.5, gamma2 = 0.01))
t <- seq(0, 1, by = 0.1)
data <- simulate(model, t = t, y0 = 0.5, plot.series = TRUE)
est <- estimate(model, t, data, 10) # better: 10000
plot(est)
pred <- predict(est)
```

---

`ad.propSd` *Adaptation of proposal standard deviation*

---

## Description

Adaptive MCMC: if acceptance rate of the chain is smaller than `lower` or larger than `upper`, the proposal standard deviation `propSd`=exp(l) is adapted with respect to function `delta.n`, that means, the new proposal standard deviation is equal to exp(l-`delta.n(batch)`), respectively exp(l+`delta.n(batch)`).

## Usage

```
ad.propSd(chain, propSd, batch, 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 |
| batch | number of batch (of chain) |
| lower | lower bound |
| upper | upper bound |
| delta.n | function of batch number |

### Value

adapted proposal standard deviation

### References

Rosenthal, J. S. (2011). Optimal Proposal Distributions and Adaptive MCMC. In: Handbook of Markov Chain Monte Carlo, pp. 93-112.

---

## class.to.list *Builds a list from class object*

---

### Description

Class slots are tranfered to list entries.

### Usage

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

### Arguments

| | |
|---|---|
| cl | class object |

### Examples

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

## diagnostic *Calculation of a proposal for burn-in phase and thinning rate*

### Description

The proposed burn-in is calculated by dividing the Markov chains into `m` blocks and calculate the 95% credibility intervals and the respective mean. Starting in the first one, the block is taken as burn-in as long as the mean of the current block is not in the credibility interval of the following block or vice versa. The thinning rate is proposed by the first lag which leads to a chain autocorrelation less than `dependence`. It is not easy to automate these choices, so it is highly recommended to verify the chains manually.

### 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 |

### Value

vector of burn-in and thinning

## Diffusion-class *S4 class of model informations for diffusion process*

### Description

Informations of model $dY_t = b(\phi, t, Y_t)dt + \gamma\widetilde{s}(t, Y_t)dW_t$.

### Slots

`phi` parameter $\phi$

`gamma2` parameter $\gamma^2$

`b.fun` function $b(\phi, t, y)$

`sT.fun` function $\widetilde{s}(t, y)$

`prior` list of prior parameters

`start` list of starting values for the Metropolis within Gibbs sampler

## Examples

```
parameter <- list(phi = 0.1, gamma2 = 0.01)
b.fun <- function(phi, t, y) phi * y
sT.fun <- function(t, y) y
start <- parameter
prior <- list(m.phi = parameter$phi, v.phi = parameter$phi^2,
   alpha.gamma = 3, beta.gamma = 2*parameter$gamma2)
model <- set.to.class("Diffusion", parameter, prior, start,
  b.fun = b.fun, sT.fun = sT.fun)
```

---

# dNtoTimes *Transformation of NHPP variables to event times*

---

## Description

Vector of Poisson process differences are translated to a vector of event times.

## Usage

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

## Arguments

| | |
|---|---|
| dN | vector of differences of counting process |
| t | times of counting process |

---

# estimate *Bayesian estimation*

---

## Description

Estimation method for the S4 classes.

## Usage

```
estimate(model.class, t, data, nMCMC, propSd, adapt = TRUE,
  proposal = c("normal", "lognormal"), ...)
```

### Arguments

| | |
|---|---|
| model.class | class object with model informations, see set.to.class |
| t | vector or list of time points |
| data | vector or list or matrix of observation variables |
| nMCMC | length of Markov chain |
| propSd | vector of proposal variances |
| adapt | if TRUE (default), proposal variance is adapted |
| proposal | proposal density: "normal" (default) or "lognormal" (for positive parameters) |
| ... | parameters dependent on the model class |

### Value

class object est.model.class containing Markov chains, data input and model informations

### References

Hermann, S. (2016). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Robert, C. P. and G. Casella (2004). Monte Carlo Statistical Methods. Springer, New York.

Rosenthal, J. S. (2011). Optimal Proposal Distributions and Adaptive MCMC. In: Handbook of Markov Chain Monte Carlo, pp. 93-112.

---

## estimate,Diffusion-method

*Estimation for diffusion process*

---

### Description

Bayesian estimation of the parameters $\phi$ and $\gamma^2$ of the stochastic process $dY_t = b(\phi, t, Y_t)dt + \gamma\tilde{s}(t, Y_t)dW_t$.

### Usage

```
## S4 method for signature 'Diffusion'
estimate(model.class, t, data, nMCMC, propSd,
  adapt = TRUE, proposal = c("normal", "lognormal"))
```

### Arguments

| | |
|---|---|
| model.class | class of the diffusion process model including all required information, see Diffusion-class |
| t | vector of time points |
| data | vector of observation variables |
| nMCMC | length of Markov chain |
| propSd | vector of proposal variances for $\phi$ |
| adapt | if TRUE (default), proposal variance is adapted |
| proposal | proposal density: "normal" (default) or "lognormal" (for positive parameters) |

### References

Hermann, S., K. Ickstadt and C. H. Mueller (2016). Bayesian Prediction of Crack Growth Based on a Hierarchical Diffusion Model. Applied Stochastic Models in Business and Industry, DOI: 10.1002/asmb.2175.

### 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 hidden diffusion process*

---

### Description

Bayesian estimation of the model $Z_i = Y_{t_i} + \epsilon_i, dY_t = b(\phi, t, Y_t)dt + \gamma\tilde{s}(t, Y_t)dW_t, \epsilon_i \sim N(0, \sigma^2), Y_{t_0} = y_0(\phi, t_0)$ with a particle Gibbs sampler.

### Usage

```
## S4 method for signature 'hiddenDiffusion'
estimate(model.class, t, data, nMCMC, propSd,
  adapt = TRUE, proposal = c("normal", "lognormal"), Npart = 100)
```

### Arguments

| | |
|---|---|
| model.class | class of the hidden diffusion model including all required information, see hiddenDiffusion-class |
| t | vector of time points |
| data | vector of observation variables |
| nMCMC | length of Markov chain |
| propSd | vector of proposal variances for $\phi$ |
| adapt | if TRUE (default), proposal variance is adapted |
| proposal | proposal density: "normal" (default) or "lognormal" (for positive parameters) |
| Npart | number of particles in the particle Gibbs sampler |

### References

Andrieu, C., A. Doucet and R. Holenstein (2010). Particle Markov Chain Monte Carlo Methods. Journal of the Royal Statistical Society B 72, pp. 269-342.

### 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, 1), 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 hierarchical (mixed) hidden diffusion process*

---

### Description

Bayesian estimation of the parameters in the hierarchical model: $Z_{ij} = Y_{t_{ij}} + \epsilon_{ij}, dY_t = b(\phi_j, t, Y_t)dt + \gamma\widetilde{s}(t, Y_t)dW_t, \phi_j \sim N(\mu, \Omega), Y_{t_0} = y_0(\phi, t_0), \epsilon_{ij} \sim N(0, \sigma^2)$ with the particle Gibbs sampler.

## Usage

```
## S4 method for signature 'hiddenmixedDiffusion'
estimate(model.class, t, data, nMCMC, propSd,
  adapt = TRUE, proposal = c("normal", "lognormal"), Npart = 100)
```

## Arguments

| | |
|---|---|
| model.class | class of the hierarchical hidden diffusion model including all required information, see hiddenmixedDiffusion-class |
| t | list or vector of time points |
| data | list or matrix of observation variables |
| nMCMC | length of Markov chain |
| propSd | vector of proposal variances for $\phi$ |
| adapt | if TRUE (default), proposal variance is adapted |
| proposal | proposal density: "normal" (default) or "lognormal" (for positive parameters) |
| Npart | number of particles in the particle Gibbs sampler |

## References

Andrieu, C., A. Doucet and R. Holenstein (2010). Particle Markov Chain Monte Carlo Methods. Journal of the Royal Statistical Society B 72, pp. 269-342.

## 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 $dY_t = b(\phi, t, Y_t)dt + s(\gamma^2, t, Y_t)dW_t + h(\theta, t, Y_t)dN_t$.

## Usage

```
## S4 method for signature 'jumpDiffusion'
estimate(model.class, t, data, nMCMC, propSd,
  adapt = TRUE, proposal = c("normal", "lognormal"), it.xi = 5)
```

## Arguments

| | |
|---|---|
| model.class | class of the jump diffusion model including all required information, see jumpDiffusion-class |
| t | vector of time points |
| data | vector of observation variables |
| nMCMC | length of Markov chain |
| propSd | vector of proposal variances for $(\phi, \theta, \gamma^2, \xi)$ |
| adapt | if TRUE (default), proposal variance is adapted |
| proposal | proposal density for phi, theta: "normal" (default) or "lognormal" (for positive parameters), see description below |
| it.xi | number of iterations for MH step for $\xi$ inside the Gibbs sampler |

## Proposal densities

For $\gamma^2$, always the lognormal density is taken, since the parameter is always positive. For $\theta$ and $\phi$, there is the possibility to choose "normal" or "lognormal" (for both together). The proposal density for $\xi$ depends on the starting value of $\xi$. If all components are positive, the proposal density is lognormal, and normal otherwise.

## Examples

```
# non-informative
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)

# informative
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)),
   priorDensity = list(phi = function(phi) dnorm(phi, 0.05, 0.01),
                       theta = function(theta) dgamma(1/theta, 10, 0.1*9),
                       gamma2 = function(gamma2) dgamma(1/gamma2, 10, 0.1*9),
                       xi = function(xi) dnorm(xi, c(3, 1/4), c(1,1))))
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,jumpRegression-method`
*Estimation for regression model dependent on Poisson process*

---

## Description

Bayesian estimation of the parameter of the regression model $y_i = f(t_i, N_{t_i}, \theta) + \epsilon_i$ with $N_t \sim Pois(\Lambda(t, \xi)), \epsilon_i \sim N(0, \gamma^2 \widetilde{s}(t))$.

## Usage

```
## S4 method for signature 'jumpRegression'
estimate(model.class, t, data, nMCMC, propSd,
  adapt = TRUE, proposal = c("normal", "lognormal"), it.xi = 10)
```

## Arguments

| | |
|---|---|
| `model.class` | class of the regression model based on the NHPP including all required information, see `jumpRegression-class` |
| `t` | vector of time points |
| `data` | vector of observation variables |
| `nMCMC` | length of Markov chain |
| `propSd` | vector of proposal variances for $(\theta, \xi)$ |
| `adapt` | if TRUE (default), proposal variance is adapted |
| `proposal` | proposal density for $\theta$: "normal" (default) or "lognormal" (for positive parameters) |
| `it.xi` | number of iterations for MH step for $\xi$ inside the Gibbs sampler |

## Proposal densities

For $\theta$, there is the possibility to choose "normal" or "lognormal". The proposal density for $\xi$ depends on the starting value of $\xi$. If all components are positive, the proposal density is lognormal, and normal otherwise.

## References

Heeke, G., S. Hermann, R. Maurer, K. Ickstadt, and C. H. Mueller (2015). Stochastic Modeling and Statistical Analysis of Fatigue Tests on Prestressed Concrete Beams under Cyclic Loadings. SFB 823 discussion paper 25/15.

## Examples

```
t <- seq(0,1, by = 0.01)
model <- set.to.class("jumpRegression", 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, plot.series = FALSE)
est <- estimate(model, t, data, 1000)
plot(est)
## Not run:
# work in progress
est_hid <- estimate(model, t, data$Y, 1000)
plot(est_hid)

## End(Not run)
```

---

`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, propSd, adapt = TRUE,
  proposal = c("normal", "lognormal"), it.xi = 10)
```

## Arguments

| | |
|---|---|
| `model.class` | class of the jump diffusion model including all required information, see `Merton-class` |
| `t` | vector of time points |
| `data` | vector of observation variables |
| `nMCMC` | length of Markov chain |
| `propSd` | vector of proposal variances for $\xi$ |
| `adapt` | if TRUE (default), proposal variance is adapted |
| `proposal` | proposal density for xi: "normal" (default) or "lognormal" |
| `it.xi` | number of iterations for MH step for $\xi$ inside the Gibbs sampler |

## References

Hermann, S. and F. Ruggeri (2016). Modelling Wear Degradation in Cylinder Liners. SFB 823 discussion paper 06/16.

Hermann, S., K. Ickstadt and C. H. Mueller (2015). Bayesian Prediction for a Jump Diffusion Process with Application to Crack Growth in Fatigue Experiments. SFB 823 discussion paper 30/15.

## 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 hierarchical (mixed) diffusion model*

---

## Description

Bayesian estimation of a model $dY_t = b(\phi_j, t, Y_t)dt + \gamma\tilde{s}(t, Y_t)dW_t, \phi_j \sim N(\mu, \Omega), Y_{t_0} = y_0(\phi, t_0)$.

## Usage

```
## S4 method for signature 'mixedDiffusion'
estimate(model.class, t, data, nMCMC, propSd,
  adapt = TRUE, proposal = c("normal", "lognormal"))
```

## Arguments

| | |
|---|---|
| `model.class` | class of the hierarchical diffusion model including all required information, see `mixedDiffusion-class` |
| `t` | list or vector of time points |
| `data` | list or matrix of observation variables |
| `nMCMC` | length of Markov chain |
| `propSd` | vector of proposal variances for $\phi$ |
| `adapt` | if TRUE (default), proposal variance is adapted |
| `proposal` | proposal density: "normal" (default) or "lognormal" (for positive parameters) |

## References

Hermann, S., K. Ickstadt and C. H. Mueller (2016). Bayesian Prediction of Crack Growth Based on a Hierarchical Diffusion Model. Applied Stochastic Models in Business and Industry, DOI: 10.1002/asmb.2175.

## 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)

##
t.list <- list()
for(i in 1:20) t.list[[i]] <- t
t.list[[21]] <- t[1:50]
data.list <- list()
for(i in 1:20) data.list[[i]] <- data[i,]
data.list[[21]] <- data[21, 1:50]
est <- estimate(model, t.list, data.list, 100)
pred <- predict(est, t = t[50:101], which.series = "current", ind.pred = 21,
   b.fun.mat = function(phi, t, y) phi[,1]-phi[,2]*y)
```

---

`estimate,mixedRegression-method`

*Estimation for the hierarchical (mixed) regression model*

---

## Description

Bayesian estimation of the parameter of the hierarchical regression model $y_{ij} = f(\phi_j, t_{ij}) + \epsilon_{ij}, \phi_j \sim N(\mu, \Omega), \epsilon_{ij} \sim N(0, \gamma^2 \widetilde{s}(t_{ij}))$.

## Usage

```
## S4 method for signature 'mixedRegression'
estimate(model.class, t, data, nMCMC, propSd,
  adapt = TRUE, proposal = c("normal", "lognormal"))
```

### Arguments

| | |
|---|---|
| model.class | class of the hierarchical regression model including all required information, see mixedRegression-class |
| t | list or vector of time points |
| data | list or matrix of observation variables |
| nMCMC | length of Markov chain |
| propSd | vector of proposal variances for $\phi$ |
| adapt | if TRUE (default), proposal variance is adapted |
| proposal | proposal density: "normal" (default) or "lognormal" (for positive parameters) |

### References

Hermann, S., K. Ickstadt, and C. H. Mueller (2016). Bayesian Prediction of Crack Growth Based on a Hierarchical Diffusion Model. Applied Stochastic Models in Business and Industry, DOI: 10.1002/asmb.2175.

### 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 = FALSE)
est <- estimate(model, t, data[1:20,], 1000)
plot(est)
```

---

## estimate,NHPP-method *Estimation for a non-homogeneous Poisson process*

---

### Description

Bayesian estimation of a non-homogeneous Poisson process (NHPP) with cumulative intensity function $\Lambda(t, \xi)$.

### Usage

```
## S4 method for signature 'NHPP'
estimate(model.class, t, data, nMCMC, propSd, adapt = TRUE,
  proposal = c("normal", "lognormal"))
```

### Arguments

| | |
|---|---|
| model.class | class of the NHPP model including all required information, see NHPP-class |
| t | vector of time points |
| data | vector of observation variables |
| nMCMC | length of Markov chain |
| propSd | vector of proposal variances for $\xi$ |
| adapt | if TRUE (default), proposal variance is adapted |
| proposal | proposal density: "normal" (default) or "lognormal" (for positive parameters) |

### References

Hermann, S., K. Ickstadt and C. H. Mueller (2015). Bayesian Prediction for a Jump Diffusion Process with Application to Crack Growth in Fatigue Experiments. SFB 823 discussion paper 30/15.

### 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 <- estimate(model, t, data$Times, 10000, proposal = "lognormal")
plot(est)

##
model <- set.to.class("NHPP", parameter = list(xi = 5),
                  Lambda = function(t, xi) t*xi)
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, plot.series = TRUE)
est <- estimate(model, t, data$N, 10000)
plot(est, par.options = list(mfrow = c(1,1)))
```

---

## estimate,Regression-method

*Estimation for regression model*

---

### Description

Bayesian estimation of the parameter of the regression model $y_i = f(\phi, t_i) + \epsilon_i, \epsilon_i \sim N(0, \gamma^2 \widetilde{s}(t_i))$.

### Usage

```
## S4 method for signature 'Regression'
estimate(model.class, t, data, nMCMC, propSd,
  adapt = TRUE, proposal = c("normal", "lognormal"))
```

### Arguments

| | |
|---|---|
| model.class | class of the regression model including all required information, see Regression-class |
| t | vector of time points |
| data | vector of observation variables |
| nMCMC | length of Markov chain |
| propSd | vector of proposal variances for $\phi$ |
| adapt | if TRUE (default), proposal variance is adapted |
| proposal | proposal density: "normal" (default) or "lognormal" (for positive parameters) |

### References

Hermann, S., K. Ickstadt, and C. H. Mueller (2016). Bayesian Prediction of Crack Growth Based on a Hierarchical Diffusion Model. Applied Stochastic Models in Business and Industry, DOI: 10.1002/asmb.2175.

### 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)
```

---

## hiddenDiffusion-class *S4 class of model informations for hidden diffusion process*

---

### Description

Informations of model $Z_i = Y_{t_i} + \epsilon_i, dY_t = b(\phi, t, Y_t)dt + \gamma\widetilde{s}(t, Y_t)dW_t, \epsilon_i \sim N(0, \sigma^2), Y_{t_0} = y_0(\phi, t_0)$.

### Slots

phi parameter $\phi$

gamma2 parameter $\gamma^2$

sigma2 parameter $\sigma^2$

y0.fun function $y_0(\phi, t)$

b.fun function $b(\phi, t, y)$

sT.fun function $\widetilde{s}(t, y)$

prior list of prior parameters

start list of starting values for the Metropolis within Gibbs sampler

## Examples

```
parameter <- list(phi = c(2, 1), gamma2 = 0.1, sigma2 = 0.1)
b.fun <- function(phi, t, y) phi[1] * y
sT.fun <- function(t, y) y
y0.fun <- function(phi, t) phi[2]
start <- parameter
prior <- list(m.phi = parameter$phi, v.phi = parameter$phi^2, alpha.gamma = 3,
   beta.gamma = parameter$gamma2*2, alpha.sigma=3, beta.sigma=parameter$sigma2*2)
model <- set.to.class("hiddenDiffusion", parameter, prior, start,
  b.fun = b.fun, sT.fun = sT.fun, y0.fun = y0.fun)
```

---

`hiddenmixedDiffusion-class` *S4 class of model informations for hierarchical (mixed) hidden diffusion process*

---

## Description

Informations of model $Z_{ij} = Y_{t_{ij}} + \epsilon_{ij}, dY_t = b(\phi_j, t, Y_t)dt + \gamma\widetilde{s}(t, Y_t)dW_t, \phi_j \sim N(\mu, \Omega), Y_{t_0} = y_0(\phi, t_0), \epsilon_{ij} \sim N(0, \sigma^2)$.

## Slots

`phi` parameter $\phi$

`mu` parameter $\mu$

`Omega` parameter $\Omega$

`gamma2` parameter $\gamma^2$

`sigma2` parameter $\sigma^2$

`y0.fun` function $y_0(\phi, t)$

`b.fun` function $b(\phi, t, y)$

`sT.fun` function $\widetilde{s}(t, y)$

`prior` list of prior parameters

`start` list of starting values for the Metropolis within Gibbs sampler

## Examples

```
mu <- c(2, 1); Omega <- c(1, 0.04)
phi <- sapply(1:2, function(i) rnorm(21, mu[i], sqrt(Omega[i])))
parameter <- list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.1, sigma2 = 0.1)
b.fun <- function(phi, t, y) phi[1] * y
sT.fun <- function(t, y) y
y0.fun <- function(phi, t) phi[2]
start <- parameter
prior <- list(m.mu = parameter$mu, v.mu = parameter$mu^2,
   alpha.omega = rep(3, length(parameter$mu)), beta.omega = parameter$Omega*2,
```

```
   alpha.gamma = 3, beta.gamma = parameter$gamma2*2,
   alpha.sigma = 3, beta.sigma = parameter$sigma2*2)
model <- set.to.class("hiddenmixedDiffusion", parameter, prior, start,
  b.fun = b.fun, sT.fun = sT.fun, y0.fun = y0.fun)
```

---

`InvMethod` *Inversion Method*

---

## Description

Algorithm to sample from cumulative distribution function, if no inverse function is analytically available.

## Usage

```
InvMethod(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") |

## References

Devroye, L. (1986). Non-Uniform Random Variate Generation. New York: Springer.

## Examples

```
test <- InvMethod(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)
```

`jumpDiffusion-class` *S4 class of model informations for the jump diffusion process*

## Description

Informations of model $dY_t = b(\phi, t, Y_t)dt + s(\gamma^2, t, Y_t)dW_t + h(\theta, t, Y_t)dN_t$ with $N_t \sim Pois(\Lambda(t, \xi))$.

## Slots

`theta` parameter $\theta$

`phi` parameter $\phi$

`gamma2` parameter $\gamma^2$

`xi` parameter $\xi$

`b.fun` function $b(\phi, t, y)$

`s.fun` function $s(\gamma^2, t, y)$

`h.fun` function $b(\theta, t, y)$

`Lambda` function $\Lambda(t, \xi)$

`priorDensity` list of prior density functions, default is a non-informative approach

`start` list of starting values for the Metropolis within Gibbs sampler

## Examples

```
parameter <- list(phi = 0.01, theta = 0.1, gamma2 = 0.01, xi = c(2, 0.2))
b.fun <- function(phi, t, y) phi * y
s.fun <- function(gamma2, t, y) sqrt(gamma2) * y
h.fun <- function(theta, t, y) theta * y
Lambda <- function(t, xi) (t / xi[2])^xi[1]
priorDensity <- list(
  phi = function(phi) 1,
  theta = function(theta) dnorm(theta, 0.1, 0.001),
  gamma2 = function(gamma2) dgamma(1/gamma2, 3, 0.01*2),
  xi = function(xi) dgamma(xi, c(2, 0.2), 1)
)
start <- parameter
model <- set.to.class("jumpDiffusion", parameter, start = start,
  b.fun = b.fun, s.fun = s.fun, h.fun = h.fun, Lambda = Lambda,
  priorDensity = priorDensity)
```

## jumpRegression-class *S4 class of model informations for the jump regression model*

### Description

Informations of model $y_i = f(t_i, N_{t_i}, \theta) + \epsilon_i$ with $N_t \sim Pois(\Lambda(t, \xi)), \epsilon_i \sim N(0, \gamma^2 \widetilde{s}(t))$.

### Slots

`theta` parameter $\theta$

`gamma2` parameter $\gamma^2$

`xi` parameter $\xi$

`fun` function $f(t, N, \theta)$

`sT.fun` function $\widetilde{s}(t)$

`Lambda` function $\Lambda(t, \xi)$

`prior` list of prior parameters

`start` list of starting values for the Metropolis within Gibbs sampler

### Examples

```
parameter <- list(theta = c(3, 1), gamma2 = 0.1, xi = c(2, 0.2))
fun <- function(t, N, theta) theta[1]*t + theta[2]*N
sT.fun <- function(t) t
Lambda <- function(t, xi) (t / xi[2])^xi[1]
prior <- list(m.theta = parameter$theta, v.theta = parameter$theta^2,
   alpha.gamma = 3, beta.gamma = parameter$gamma2*2)
start <- parameter
model <- set.to.class("jumpRegression", parameter, prior, start = start,
  fun = fun, sT.fun = sT.fun, Lambda = Lambda)
```

## Merton-class *S4 class of model informations for a special jump diffusion process, called Merton model*

### Description

Informations of model $dY_t = \phi Y_t dt + \gamma^2 Y_t dW_t + \theta Y_t dN_t$ with $N_t \sim Pois(\Lambda(t, \xi))$. The explicit solution of the SDE is given by $Y_t = y_0 \exp(\phi t - \gamma^2/2t + \gamma W_t + \log(1 + \theta) N_t)$.

## Slots

`thetaT` parameter $\widetilde{\theta} = \log(1+\theta)$

`phi` parameter $\phi$

`gamma2` parameter $\gamma^2$

`xi` parameter $\xi$

`Lambda` function $\Lambda(t, \xi)$

`prior` list of prior parameters for $\phi, \widetilde{\theta}, \gamma^2$

`priorDensity` list of prior density function for $\xi$

`start` list of starting values for the Metropolis within Gibbs sampler

## Examples

```
parameter <- list(phi = 0.01, thetaT = 0.1, gamma2 = 0.01, xi = c(2, 0.2))
Lambda <- function(t, xi) (t / xi[2])^xi[1]
# prior density for xi:
priorDensity <- function(xi) dgamma(xi, c(2, 0.2), 1)
# prior parameter for phi (normal), thetaT (normal) and gamma2 (inverse gamma):
prior <- list(m.phi = parameter$phi, v.phi = parameter$phi, m.thetaT = parameter$thetaT,
   v.thetaT = parameter$thetaT, alpha.gamma = 3, beta.gamma = parameter$gamma2*2)
start <- parameter
model <- set.to.class("Merton", parameter, prior, start, Lambda = Lambda,
   priorDensity = priorDensity)
summary(class.to.list(model))
# default:
model <- set.to.class("Merton", parameter, Lambda = Lambda)
```

---

`mixedDiffusion-class` *S4 class of model informations for hierarchical (mixed) diffusion process model*

---

## Description

Informations of model $dY_t = b(\phi_j, t, Y_t)dt + \gamma\widetilde{s}(t, Y_t)dW_t, \phi_j \sim N(\mu, \Omega), Y_{t_0} = y_0(\phi, t_0)$.

## Slots

`phi` parameter $\phi$

`mu` parameter $\mu$

`Omega` parameter $\Omega$

`gamma2` parameter $\gamma^2$

`y0.fun` function $y_0(\phi, t)$

`b.fun` function $b(\phi, t, y)$

`sT.fun` function $\widetilde{s}(t, y)$

`prior` list of prior parameters

`start` list of starting values for the Metropolis within Gibbs sampler

## Examples

```
mu <- c(2, 1); Omega <- c(1, 0.04)
phi <- sapply(1:2, function(i) rnorm(21, mu[i], sqrt(Omega[i])))
parameter <- list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.01)
b.fun <- function(phi, t, y) phi[1] * y
sT.fun <- function(t, y) y
y0.fun <- function(phi, t) phi[2]
start <- parameter
prior <- list(m.mu = parameter$mu, v.mu = parameter$mu^2,
   alpha.omega = rep(3, length(parameter$mu)), beta.omega = parameter$Omega*2,
   alpha.gamma = 3, beta.gamma = parameter$gamma2*2)
model <- set.to.class("mixedDiffusion", parameter, prior, start,
  b.fun = b.fun, sT.fun = sT.fun, y0.fun = y0.fun)
```

---

# mixedRegression-class *S4 class of model informations for the hierarchical (mixed) regression model*

---

## Description

Informations of model $y_{ij} = f(\phi_j, t_{ij}) + \epsilon_{ij}, \phi_j \sim N(\mu, \Omega), \epsilon_{ij} \sim N(0, \gamma^2 \widetilde{s}(t_{ij}))$.

## Slots

`phi` parameter $\phi$

`mu` parameter $\mu$

`Omega` parameter $\Omega$

`gamma2` parameter $\gamma^2$

`fun` function $f(\phi, t)$

`sT.fun` function $\widetilde{s}(t)$

`prior` list of prior parameters

`start` list of starting values for the Metropolis within Gibbs sampler

## Examples

```
mu <- c(2, 1); Omega <- c(1, 0.04)
phi <- sapply(1:2, function(i) rnorm(21, mu[i], sqrt(Omega[i])))
parameter <- list(phi = phi, mu = mu, Omega = Omega, gamma2 = 0.01)
fun <- function(phi, t) phi[1] + phi[2]*t
sT.fun <- function(t) t
prior <- list(m.mu = parameter$mu, v.mu = parameter$mu^2,
   alpha.omega = rep(3, length(parameter$mu)), beta.omega = parameter$Omega*2,
   alpha.gamma = 3, beta.gamma = parameter$gamma2*2)
start <- parameter
model <- set.to.class("mixedRegression", parameter, prior, start, fun = fun, sT.fun = sT.fun)
```

## NHPP-class *S4 class of model informations for non-homogeneous Poisson process*

### Description

Informations of NHPP with cumulative intensity function $\Lambda(t, \xi)$.

### Slots

`xi` parameter $\xi$

`Lambda` function $\Lambda(t, \xi)$

`priorDensity` prior density function for $\xi$

`start` list of starting values for the Metropolis within Gibbs sampler

### Examples

```
parameter <- list(xi = c(2, 0.2))
Lambda <- function(t, xi) (t / xi[2])^xi[1]
priorDensity <- function(xi) dgamma(xi, c(2, 0.2), 1)
start <- parameter
model <- set.to.class("NHPP", parameter, start = start, Lambda = Lambda,
   priorDensity = priorDensity)
```

## plot,est.Diffusion-method *Plot method for the Bayesian estimation results*

### Description

Plot method for the estimation results of the diffusion model.

### Usage

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

### Arguments

| | |
|---|---|
| x | est.Diffusion class, created with method estimate,Diffusion-method |
| par.options | list of options for function par() |
| style | one out of "chains", "acf", "density" |
| par2plot | logical vector, which parameters to be plotted, order: $(\phi, \gamma^2)$ |
| 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 |
| ... | 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, reduced = TRUE)
plot(est, 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.hiddenDiffusion-method

*Plot method for the Bayesian estimation results*

---

### Description

Plot method for the estimation results of the hidden diffusion model.

### Usage

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

### Arguments

| | |
|---|---|
| x | est.hiddenDiffusion class, created with method estimate,hiddenDiffusion-method |
| 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 |
| ... | 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, reduced = TRUE)
plot(est, 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.hiddenmixedDiffusion-method

*Plot method for the Bayesian estimation results*

---

### Description

Plot method for the estimation results of the hidden hierarchical diffusion model.

### Usage

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

### Arguments

| | |
|---|---|
| x | est.hiddenmixedDiffusion class, created with method estimate,hiddenmixedDiffusion-method |
| 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 |
| ... | 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, burnIn = 10, thinning = 2, reduced = TRUE)
plot(est, 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, style = "int.phi", phi = phi, par2plot = c(TRUE, FALSE, FALSE))

## End(Not run)
```

---

## plot,est.jumpDiffusion-method

*Plot method for the Bayesian estimation results*

---

### Description

Plot method for the estimation results of the jump diffusion model.

## Usage

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

## Arguments

| | |
|---|---|
| x | est.jumpDiffusion class, created with method estimate,jumpDiffusion-method |
| 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 |
| ... | 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, reduced = TRUE)
plot(est, 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.jumpRegression-method

*Plot method for the Bayesian estimation results*

---

## Description

Plot method for the estimation results of the jump regression model.

## Usage

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

## Arguments

| | |
|---|---|
| x | est.jumpRegression class, created with method estimate,jumpRegression-method |
| 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 |
| ... | optional plot parameters |

## Examples

```
model <- set.to.class("jumpRegression", 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, reduced = TRUE)
plot(est, 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.Merton-method

*Plot method for the Bayesian estimation results*

---

## Description

Plot method for the estimation results of the Merton model.

## Usage

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

## Arguments

| | |
|---|---|
| x | est.Merton class, created with method estimate,Merton-method |
| 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 |
| ... | 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, reduced = TRUE)
plot(est, 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.mixedDiffusion-method

*Plot method for the Bayesian estimation results*

---

## Description

Plot method for the estimation results of the hierarchical (mixed) diffusion model.

## Usage

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

## Arguments

| | |
|---|---|
| x | est.mixedDiffusion class, created with method estimate,mixedDiffusion-method |
| 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: $(\mu, \Omega, \gamma^2)$ |
| 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 |
| ... | 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, burnIn = 10, thinning = 2, reduced = TRUE)
plot(est, 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, style = "int.phi", phi = phi, par2plot = c(TRUE, FALSE, FALSE))
```

`plot,est.mixedRegression-method`

*Plot method for the Bayesian estimation results*

## Description

Plot method for the estimation results of the hierarchical (mixed) regression model.

## Usage

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

## Arguments

| | |
|---|---|
| `x` | est.mixedRegression class, created with method `estimate,mixedRegression-method` |
| `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: $(\mu, \Omega, \gamma^2)$ |
| `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 |
| `...` | 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, burnIn = 10, thinning = 2, reduced = TRUE)
plot(est, 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, style = "int.phi", phi = phi, par2plot = c(TRUE, FALSE))
```

---

`plot,est.NHPP-method` *Plot method for the Bayesian estimation results*

---

## Description

Plot method for the estimation results of the NHPP.

## Usage

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

## Arguments

| | |
|---|---|
| `x` | est.NHPP class, created with method `estimate,NHPP-method` |
| `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 |
| `...` | 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, reduced = TRUE)
plot(est, 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.Regression-method

*Plot method for the Bayesian estimation results*

## Description

Plot method for the estimation results of the regression model.

## Usage

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

## Arguments

| | |
|---|---|
| x | est.Regression class, created with method estimate,Regression-method |
| par.options | list of options for function par() |
| style | one out of "chains", "acf", "density" |
| par2plot | logical vector, which parameters to be plotted, order: $(\phi, \gamma^2)$ |
| 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 |
| ... | 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, reduced = TRUE)
plot(est, 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))
```

pred.base *Bayesian prediction function*

## Description

Drawing from predictive distribution based on distribution function `Fun(x, x0, samples)` or density `dens(x, x0, samples)`. Samples should contain samples from the posterior distribution of the parameters.

## Usage

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

## Arguments

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

## Value

vector of samples from prediction

## References

Hermann, S. (2016). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

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

## Description

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

## 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("InvMethod", "RejSamp"), sample.length, grid,
  plot.prediction = TRUE)
```

## Arguments

| | |
|---|---|
| `object` | class object of MCMC samples: "est.Diffusion", created with method `estimate,Diffusion-method` |
| `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 be predicted and t missing, `M2pred` variables will be predicted with the observation time distances |
| `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 ("InvMethod") or rejection sampling ("RejSamp") |
| `sample.length` | number of samples to be drawn, default is the number of posterior samples |
| `grid` | fineness degree of sampling approximation |
| `plot.prediction` | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## 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)
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 a hidden diffusion process*

---

## Description

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

## 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", created with method `estimate,hiddenDiffusion-method` |
| `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 be predicted and t missing, M2pred variables will be predicted with the observation time distances |
| cand.length | length of candidate samples (if method = "vector") |
| pred.alg | prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory" |
| sample.length | number of samples to be drawn, default is the number of posterior samples |
| grid | fineness degree of sampling approximation |
| plot.prediction | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## Examples

```
## Not run:
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)
est_hiddiff <- estimate(model, t, data$Z, 100)  # nMCMC should be much larger!
plot(est_hiddiff)

pred_hiddiff <- predict(est_hiddiff, t = seq(0, 1, by = 0.1))
pred_hiddiff2 <- predict(est_hiddiff, which.series = "current")

pred_hiddiff <- predict(est_hiddiff, pred.alg = "simpleTrajectory", sample.length = 100)
pred_hiddiff <- predict(est_hiddiff, pred.alg = "simpleBayesTrajectory")

## End(Not run)
```

---

# predict,est.hiddenmixedDiffusion-method

*Prediction for a hierarchical (mixed) hidden diffusion process model*

---

## Description

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

## 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", created with method estimate,hiddenmixedDiffusion-method |
| 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 be predicted and t missing, M2pred variables will be predicted with the observation time distances |
| cand.length | length of candidate samples (if method = "vector") |
| pred.alg | prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory" |
| sample.length | number of samples to be drawn, default is the number of posterior samples |
| grid | fineness degree of sampling approximation |
| plot.prediction | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## 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)

## Not run:
est_hidmixdiff <- estimate(model, t, data$Z[1:20,], 200)
plot(est_hidmixdiff)
pred1 <- predict(est_hidmixdiff, b.fun.mat = function(phi, t, y) phi[,1])
pred2 <- predict(est_hidmixdiff, pred.alg = "Trajectory", b.fun.mat = function(phi, t, y) phi[,1])
pred3 <- predict(est_hidmixdiff, pred.alg = "simpleTrajectory", sample.length = nrow(pred1$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 a 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", created with method `estimate,jumpDiffusion-me` |
| `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` | optional, if current series to be predicted and t missing, `M2pred` variables will be predicted with the observation time distances |
| `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 | number of samples to be drawn, default is the number of posterior samples |
| plot.prediction | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## 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)
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, xi) (t/xi[,2])^xi[,1])
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")
pred_jd5 <- predict(est_jd, pred.alg = "simpleBayesTrajectory", sample.length = 100)
```

---

# predict,est.jumpRegression-method

*Prediction for a regression model dependent on a Poisson process*

---

## Description

Bayesian prediction of a regression model $y_i = f(t_i, N_{t_i}, \theta) + \epsilon_i$ with $N_t \sim Pois(\Lambda(t, \xi)), \epsilon_i \sim N(0, \gamma^2 \widetilde{s}(t))$.

## Usage

```
## S4 method for signature 'est.jumpRegression'
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.jumpRegression", created with method estimate,jumpRegression-method |
| 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 | optional, if current series to be predicted and t missing, M2pred variables will be predicted with the observation time distances |
| cand.length | length of candidate samples (if method = "vector"), for jump diffusion |
| pred.alg | prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleTrajectory" |
| sample.length | number of samples to be drawn, default is the number of posterior samples |
| grid | fineness degree of sampling approximation |
| plot.prediction | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## Examples

```
t <- seq(0,1, by = 0.01)
cl <- set.to.class("jumpRegression", 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)
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)
```

---

`predict,est.Merton-method`

*Prediction for a 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", created with method `estimate,Merton-method` |
| `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` | optional, if current series to be predicted and t missing, `M2pred` variables will be predicted with the observation time distances |
| `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` | number of samples to be drawn, default is the number of posterior samples |
| `plot.prediction` | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## 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)
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 a hierarchical (mixed) diffusion process model*

---

## Description

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

## 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", created with method estimate,mixedDiffusion-method |
| 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 be predicted and t missing, M2pred variables will be predicted with the observation time distances |
| cand.length | length of candidate samples (if method = "vector") |
| pred.alg | prediction algorithm, "Distribution", "Trajectory", "simpleTrajectory" or "simpleBayesTrajectory" |
| sample.length | number of samples to be drawn, default is the number of posterior samples |
| grid | fineness degree of sampling approximation |
| plot.prediction | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## 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)
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])))
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)
est <- estimate(model, t, data[1:20,], 2000)
plot(est)
pred <- 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$Y, 2, quantile, 0.025) <= data[21, seq(1, length(t), length = 21)] &
     apply(pred$Y, 2, quantile, 0.975) >= data[21, seq(1, length(t), length = 21)])
mean(sapply(1:20, function(i){
    mean(apply(pred$Y, 2, quantile, 0.025) <= data[i, seq(1, length(t), length = 21)] &
    apply(pred$Y, 2, quantile, 0.975) >= data[i, seq(1, length(t), length = 21)])}))

## End(Not run)
```

---

`predict,est.mixedRegression-method`

*Prediction for a mixed regression model*

---

## Description

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

## 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,
  method = c("vector", "free"), sampling.alg = c("InvMethod", "RejSamp"),
  sample.length, grid, plot.prediction = TRUE)
```

## Arguments

| | |
|---|---|
| object | class object of MCMC samples: "est.mixedRegression", created with method estimate,mixedRegression-method |
| 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 be predicted and t missing, M2pred variables will be predicted with the observation time distances |
| cand.length | length of candidate samples (if method = "vector") |
| method | vectorial ("vector") or not ("free") |
| sampling.alg | sampling algorithm, inversion method ("InvMethod") or rejection sampling ("RejSamp") |
| sample.length | number of samples to be drawn, default is the number of posterior samples |
| grid | fineness degree of sampling approximation |
| plot.prediction | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## 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)
est <- estimate(model, 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)
```

```
t.list <- list()
for(i in 1:20) t.list[[i]] <- t
t.list[[21]] <- t[1:50]
data.list <- list()
for(i in 1:20) data.list[[i]] <- data[i,]
data.list[[21]] <- data[21, 1:50]
est <- estimate(model, t.list, data.list, 100)
pred <- predict(est, t = t[50:101], which.series = "current", ind.pred = 21,
   fun.mat = function(phi, t) phi[,1]*t + phi[,2])
```

---

`predict,est.NHPP-method`

*Prediction for a non-homogeneous Poisson process*

---

## Description

Bayesian prediction of a non-homogeneous Poisson process with cumulative intensity function $\Lambda(t, \xi)$.

## 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", created with method `estimate,NHPP-method` |
| `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 be predicted and t missing, `M2pred` variables will be predicted with the observation time distances |

| | |
|---|---|
| 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 | number of samples to be drawn, default is the number of posterior samples |
| grid | fineness degree of sampling approximation |
| plot.prediction | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## 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)
est <- estimate(model, t, data$Times, 1000)  # nMCMC should be much larger!
plot(est)
pred <- predict(est, Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1],
   variable = "PoissonProcess", pred.alg = "Distribution")

## Not run:
pred_NHPP <- predict(est, Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
pred_NHPP <- predict(est, variable = "PoissonProcess",
   Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
pred_NHPP2 <- predict(est, which.series = "current",
   Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
pred_NHPP3 <- predict(est, variable = "PoissonProcess", which.series = "current",
                      Lambda.mat = function(t, xi) (t/xi[,2])^xi[,1])
pred_NHPP4 <- predict(est, pred.alg = "simpleTrajectory", M2pred = length(data$Times))

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

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

## Description

Bayesian prediction of regression model $y_i = f(\phi, t_i) + \epsilon_i, \epsilon_i \sim N(0, \gamma^2 \widetilde{s}(t_i))$.

## 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("InvMethod", "RejSamp"), sample.length, grid,
  plot.prediction = TRUE)
```

## Arguments

| | |
|---|---|
| `object` | class object of MCMC samples: "est.Regression", created with method `estimate,Regression-method` |
| `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 be predicted and t missing, `M2pred` variables will be predicted with the observation time distances |
| `cand.length` | length of candidate samples (if method = "vector") |
| `method` | vectorial ("vector") or not ("free") |
| `sampling.alg` | sampling algorithm, inversion method ("InvMethod") or rejection sampling ("RejSamp") |
| `sample.length` | number of samples to be drawn, default is the number of posterior samples |
| `grid` | fineness degree of sampling approximation |
| `plot.prediction` | if TRUE, prediction intervals are plotted |

## References

Hermann, S. (2016a). BaPreStoPro: an R Package for Bayesian Prediction of Stochastic Processes. SFB 823 discussion paper 28/16.

Hermann, S. (2016b). Bayesian Prediction for Stochastic Processes based on the Euler Approximation Scheme. SFB 823 discussion paper 27/16.

## 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)
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)
```

---

# prediction.intervals *Bayesian prediction interval function*

---

## Description

Calculation of quantiles `level/2` and 1-`level/2` from predictive distribution based on distribution function `Fun(x, x0, samples)`. `samples` should contain samples from the posterior distribution of the parameters.

## Usage

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

## Arguments

| | |
|---|---|
| `samples` | posterior samples |
| `Fun` | cumulative distribution function |
| `x0` | starting point |
| `level` | level of prediction intervals |
| `candArea` | candidate area |
| `grid` | fineness degree for the binary search |

## Value

prediction interval

## proposal *Sampling from lognormal proposal density*

### Description

Drawing one sample from the lognormal distribution with mean `parOld` and standard deviation `propSd`. Used in Metropolis Hastings algorithms.

### Usage

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

### Arguments

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

### Examples

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

## proposalRatio *Proposal ratio of lognormal proposal density*

### Description

Calculation of proposal ratio, see also `proposal`.

### Usage

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

### Arguments

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

### Examples

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

## Regression-class *S4 class of model informations for the regression model*

### Description

Informations of model $y_i = f(\phi, t_i) + \epsilon_i, \epsilon_i \sim N(0, \gamma^2 \widetilde{s}(t_i))$.

### Slots

`phi` parameter $\phi$

`gamma2` parameter $\gamma^2$

`fun` function $f(\phi, t)$

`sT.fun` function $\widetilde{s}(t)$

`prior` list of prior parameters

`start` list of starting values for the Metropolis within Gibbs sampler

### Examples

```
parameter <- list(phi = c(3, 1), gamma2 = 0.1)
fun <- function(phi, t) phi[1] + phi[2]*t
sT.fun <- function(t) t
prior <- list(m.phi = parameter$phi, v.phi = parameter$phi^2,
   alpha.gamma = 3, beta.gamma = 2*parameter$gamma2)
start <- parameter
model <- set.to.class("Regression", parameter, prior, start, fun = fun, sT.fun = sT.fun)
```

## RejSampling *Rejection Sampling Algorithm*

### Description

Algorithm to sample from an arbitrary density function.

### 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") |

## References

Devroye, L. (1986). Non-Uniform Random Variate Generation. New York: Springer.

## Examples

```
plot(density(RejSampling(dens = function(x) dnorm(x, 5, 1),
   len = 500, cand = seq(2, 9, by = 0.001), method = "free")))
lines(density(RejSampling(dens = function(x) dnorm(x, 5, 1), len = 500,
      cand = seq(2, 9, by = 0.001), method = "vector")), col=2)
curve(dnorm(x, 5, 1), from = 2, to = 8, add = TRUE, col = 3)
```

---

# set.to.class *Building of model classes*

---

## Description

Definition of the model classes.

## Usage

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

## 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 |
| priorDensity | list of functions for prior densities, if missing: non-informative estimation |

## Details

set.to.class is the central function to define a S4 model class, where the simulate and the estimate methods build up. Main input parameter is class.name, which is one out of "jumpDiffusion", "Merton", "Diffusion", "mixedDiffusion", "hiddenDiffusion", "hiddenmixedDiffusion", "jumpRegression", "NHPP", "Regression" and "mixedRegression", which is the name of the class object containing all information of the model. If you write set.to.class(class.name) without any further input parameter, the function tells you which entries the list parameter has to contain. This is the second central input parameter. If input parameter start is missing, it is set to parameters. If input parameter prior, which is a list of prior parameters, is missing, they are calculated from parameter in that way, that prior mean and standard deviation is equal to the entries of parameter. Functions b.fun, s.fun, h.fun can be seen in the model definition of the jump diffusion $dY_t = b(\phi, t, Y_t)dt + s(\gamma^2, t, Y_t)dW_t + h(\theta, t, Y_t)dN_t$. In the case of a continuous diffusion, one out of "Diffusion", "mixedDiffusion", "hiddenDiffusion" or "hiddenmixedDiffusion", variance function $s(\gamma^2, t, y)$ is restricted to the case $s(\gamma^2, t, y) = \gamma\widetilde{s}(t, y)$. sT.fun stands for $\widetilde{s}(t, y)$. In the case of a regression model, "Regression" or "mixedRegression", sT.fun means the variance function dependent on t of the regression error $\epsilon_i \sim N(0, \sigma^2\widetilde{s}(t))$. In both cases, default value is sT.fun = function(t, y) 1. y0.fun is for the models, where the starting value depends on the parameter phi, "mixedDiffusion", "hiddenDiffusion" or "hiddenmixedDiffusion". Default value is a constant function in 1. fun is the regression function for the models "Regression", "mixedRegression" and "jumpRegression". In the first two cases, this is $f(\phi, t)$ and in the third $f(t, N_t, \theta)$. Function Lambda is the cumulative intensity function in the models including the non-homogeneous Poisson process. Input parameter priorDensity is for the model class jumpDiffusion-class a list of functions for the prior density functions. For the model classes NHPP-class and Merton-class, priorDensity is the density of the intensity rate parameter of the Poisson process. Default is a non-informative approach for all cases.

## Examples

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

---

# simulate,Diffusion-method

*Simulation of diffusion process*

---

## Description

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

## 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 trajectories to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| y0 | starting point of the process |
| mw | mesh width for finer Euler approximation to simulate time-continuity |
| 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 hidden diffusion process*

---

### Description

Simulation of a hidden stochastic process model $Z_i = Y_{t_i} + \epsilon_i, dY_t = b(\phi, t, Y_t)dt + \gamma\widetilde{s}(t, Y_t)dW_t, \epsilon_i \sim N(0, \sigma^2), Y_{t_0} = y_0(\phi, t_0)$.

### 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 trajectories to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| mw | mesh width for finer Euler approximation to simulate time-continuity |
| 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 hierarchical (mixed) hidden diffusion model*

### Description

Simulation of a stochastic process $Z_{ij} = Y_{t_{ij}} + \epsilon_{ij}, dY_t = b(\phi_j, t, Y_t)dt + \gamma\widetilde{s}(t, Y_t)dW_t, \phi_j \sim N(\mu, \Omega), Y_{t_0} = y_0(\phi, t_0), \epsilon_{ij} \sim N(0, \sigma^2)$.

### 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 data sets to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| mw | mesh width for finer Euler approximation to simulate time-continuity |
| 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(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)
```

## 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,
  start = c(0, 0), mw = 1, plot.series = TRUE)
```

## Arguments

| | |
|---|---|
| object | class object of parameters: "jumpDiffusion" |
| nsim | number of trajectories to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| y0 | starting point of process |
| start | vector: start[1] starting point time, start[2] starting point for Poisson process |
| mw | mesh width for finer Euler approximation to simulate time-continuity |
| 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)),
   Lambda = function(t, xi) (t/xi[2])^xi[1])
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t, y0 = 0.5)
```

---

# simulate,jumpRegression-method

*Simulation of regression model dependent on Poisson process*

---

## Description

Simulation of of the regression model $y_i = f(t_i, N_{t_i}, \theta) + \epsilon_i$ with $N_t \sim Pois(\Lambda(t, \xi)), \epsilon_i \sim N(0, \gamma^2 \widetilde{s}(t))$.

## Usage

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

## Arguments

| | |
|---|---|
| object | class object of parameters: "jumpRegression" |
| nsim | number of trajectories to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| plot.series | logical(1), if TRUE, simulated series are depicted grafically |

## Examples

```
model <- set.to.class("jumpRegression", fun = function(t, N, theta) theta[1]*t + theta[2]*N,
   parameter = list(theta = c(1,2), gamma2 = 0.1, xi = 10))
t <- seq(0, 1, by = 0.01)
data <- simulate(model, t = t)
```

---

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,
  start = c(0, 0), plot.series = TRUE)
```

## Arguments

| | |
|---|---|
| object | class object of parameters: "Merton" |
| nsim | number of trajectories to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| y0 | starting point of process |
| start | vector: start[1] starting point time, start[2] starting point for Poisson 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 hierarchical (mixed) diffusion model*

---

## Description

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

## 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 data sets to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| mw | mesh width for finer Euler approximation to simulate time-continuity |
| 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", y0.fun = function(phi, t) 0.5,
  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)
```

---

simulate,mixedRegression-method

*Simulation of hierarchical (mixed) regression model*

---

## Description

Simulation of regression model $y_{ij} = f(\phi_j, t_{ij}) + \epsilon_{ij}, \phi_j \sim N(\mu, \Omega), \epsilon_{ij} \sim N(0, \gamma^2\widetilde{s}(t_{ij}))$.

## 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 data sets to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| 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),
   fun = function(phi, t) phi*t, sT.fun = function(t) t)
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 non-homogeneous Poisson process with cumulative intensity function $\Lambda(t, \xi)$.

### 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 trajectories to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| 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 $y_i = f(\phi, t_i) + \epsilon_i, \epsilon_i \sim N(0, \gamma^2 \widetilde{s}(t_i))$.

## 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 trajectories to simulate. Default is 1. |
| seed | optional: seed number for random number generator |
| t | vector of time points |
| 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)
```

TimestoN *Transformation of event times to NHPP*

## Description

Transformation of vector of event times to the corresponding counting process variables.

## Usage

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

## Arguments

| | |
|---|---|
| times | vector of event times |
| t | times of counting process |

Virkler *Crack Growth*

## Description

68 measurement series in 164 time points

## Usage

```
Virkler
```

## Format

Sixty-eight replicate constant amplitude tests in aluminum alloy were carried out to investigate the fatigue crack propagation. In each of these tests, the number of cycles that leads to fixed crack lengths was observed. Against the natural assumption that something is observed at fixed times, here the time is the dependent variable and the crack length is the independent variable. Therefore, from the mathematical viewpoint the crack length will here be treated as time vector $t$.

The Virkler data comes as a data-frame of 164 rows and 69 columns where the first column contains the crack lengths (in mm) and the 68 following the series of observed times in load cycles up to a fixed crack length.

We want to thank Eric J. Tuegel for providing us the data that were collected by Prof. B. M. Hillberry, published in Virkler et al. (1979).

## Source

*Eric J. Tuegel*

## References

*Virkler, D. A., Hillberry, B. M. and Goel, P. K. (1979). The Statistical Nature of Fatigue Crack Propagation. Journal of Engineering Materials and Technology 101, 148–153.*

## Examples

```
data(Virkler)

Y <- t(Virkler[,-1]/10000)
t <- Virkler[,1]

plot(t, Y[1,], type = 'l', ylim = range(Y), xlab = "crack length in mm",
   ylab = "time in load cycles / 10000")
for (i in 2:nrow(Y)){
 lines(t, Y[i,])
}
```

# Index