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 .................................................. 3
ad.propSd .............................................................. 6
class.to.list .......................................................... 7
diagnostic ............................................................... 8
Diffusion-class ......................................................... 8
dNtoTimes ............................................................... 9
estimate ................................................................. 9
estimate,Diffusion-method ........................................... 10
estimate,hiddenDiffusion-method ................................ 11
estimate,hiddenmixedDiffusion-method ......................... 12
estimate,jumpDiffusion-method ................................... 13
estimate,jumpRegression-method ................................. 15
estimate,Merton-method .......................... 16
estimate,mixedDiffusion-method .................... 17
estimate,mixedRegression-method .................. 18
estimate,NHPP-method .......................... 19
estimate,Regression-method ........................ 20
hiddenDiffusion-class .......................... 21
hiddenmixedDiffusion-class ........................ 22
InvMethod .................................. 23
jumpDiffusion-class .......................... 24
jumpRegression-class .......................... 25
Merton-class .................................. 25
mixedDiffusion-class .......................... 26
mixedRegression-class .......................... 27
NHPP-class .................................. 28
plot,est.Diffusion-method ......................... 28
plot,est.hiddenDiffusion-method .................. 29
plot,est.hiddenmixedDiffusion-method ............ 30
plot,est.jumpDiffusion-method .................... 31
plot,est.jumpRegression-method .................. 32
plot,est.Merton-method .......................... 33
plot,est.mixedDiffusion-method .................. 34
plot,est.mixedRegression-method .................. 36
plot,est.NHPP-method .......................... 37
plot,est.Regression-method ....................... 38
pred.base .................................. 39
predict,est.Diffusion-method ..................... 40
predict,est.hiddenDiffusion-method ............... 41
predict,est.hiddenmixedDiffusion-method ........ 42
predict,est.jumpDiffusion-method ................ 44
predict,est.jumpRegression-method ............... 45
predict,est.Merton-method ....................... 47
predict,est.mixedDiffusion-method ............... 48
predict,est.mixedRegression-method ............. 50
predict,est.NHPP-method ....................... 52
predict,est.Regression-method ................... 54
prediction.intervals ................................ 55
proposal .................................. 56
proposalRatio .................................. 56
Regression-class .................................. 57
RejSampling .................................. 57
set.to.class .................................. 57
simulate,Diffusion-method ......................... 58
simulate,hiddenDiffusion-method .................. 59
simulate,hiddenmixedDiffusion-method ............ 60
simulate,jumpDiffusion-method .................... 61
simulate,jumpRegression-method .................. 61
simulate,Merton-method .......................... 62
simulate,mixedDiffusion-method .................. 63
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

<table>
<thead>
<tr>
<th>Package:</th>
<th>BaPreStoPro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>1.0</td>
</tr>
<tr>
<td>Date:</td>
<td>2016-06-07</td>
</tr>
<tr>
<td>License:</td>
<td>GLP-2, GLP-3</td>
</tr>
</tbody>
</table>

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), $\gamma^2$ is assumed to have an inverse gamma distribution with parameters $\alpha_{\gamma^2}$ and $\beta_{\gamma^2}$. An example how to build the model class can be found in `mertonMclass` and `estimateLmertonMmethod`.

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 $s(\gamma^2,t,y) = \sqrt{\gamma^2} \tilde{s}(t,y)$, because a conjugate prior for $\gamma^2$ is available (the inverse gamma with parameters $\alpha_{\gamma^2}$ and $\beta_{\gamma^2}$) 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 `diffusionMclass` and `estimateLdiffusionMmethod`.

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 `mixedDiffusionMclass` and `estimateLmixedDiffusionMmethod`.

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: $Z_i = Y_i + \epsilon_i, dY_i = b(\phi,t,Y_i)dt + \gamma \tilde{s}(t,Y_i)dW_t, \epsilon_i \sim N(0,\sigma^2), Y_{i_0} = y_0(\phi,t_0)$. The inverse gamma prior for $\sigma^2$ is conjugate and is, therefore, implemented with parameters $alpha_{\sigma^2}$ and $beta_{\sigma^2}$. 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 `hiddenDiffusionMclass` and `estimateLhiddenDiffusionMmethod`
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 $N_{t_i} \sim \text{Pois}(\Lambda(t, \xi))$ with $\epsilon_i \sim N(0, \gamma^2 \tilde{s}(t))$.

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


Monte Carlo methods:


Adaptive MCMC:


particle Gibbs / SMC:


Examples

```r
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**

```r
ad.propSd(chain, propSd, batch, lower = 0.3, upper = 0.6,
delta.n = function(n) min(0.05, 1/sqrt(n)))
```
class.to.list

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


class.to.list          Builds a list from class object

Description

Class slots are transferred 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))
**Description**

The proposed burn-in is calculated by dividing the Markov chains into \( m \) blocks and calculating 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**

\[
\text{diagnostic}(\text{chain}, \text{dependence} = 0.8, m = 10)
\]

**Arguments**

- \textit{chain}: vector of Markov chain samples
- \textit{dependence}: allowed dependence for the chain
- \textit{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 \( \text{d}Y_t = b(\phi, t, Y_t)\text{d}t + \gamma \tilde{s}(t, Y_t)\text{d}W_t \).

**Slots**

- \textit{phi}: parameter \( \phi \)
- \textit{gamma2}: parameter \( \gamma^2 \)
- \textit{b.fun}: function \( b(\phi, t, y) \)
- \textit{sT.fun}: function \( \tilde{s}(t, y) \)
- \textit{prior}: list of prior parameters
- \textit{start}: list of starting values for the Metropolis within Gibbs sampler
**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**


**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**

```r
## 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
proposition  proposal density: "normal" (default) or "lognormal" (for positive parameters)

References


Examples

```r
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, 10000)
plot(est_diff)
```

Description

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

Usage

```r
## S4 method for signature 'hiddenDiffusion'
estimate(model.class, t, data, nMCMC, propSd,
adapt = TRUE, proposition = 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


### Examples

```r
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(0, 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)
```

### Description

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

```r
## 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


Examples

```r
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)
```

Description

Bayesian estimation of a stochastic process $dY_t = b(\phi, t, Y_t)dt + s(\gamma^2, t, Y_t)dW_t + h(\theta, t, Y_t)dN_t$. 

Estimation for jump diffusion process
Usage

```r
## 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

```r
# 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)
```
estimate jumperegression-method

plot(est_hidden)
## End(Not run)

---

### estimate, jumperegression-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 $N_{t_i} \sim \text{Pois}(\Lambda(t, \xi)), \epsilon_i \sim N(0, \gamma^2 \tilde{s}(t))$.

**Usage**

```r
## S4 method for signature 'jumperegression'
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**

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)

Estimation for jump diffusion process

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

Estimation for hierarchical (mixed) diffusion model

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

Usage

### S4 method for signature 'mixedDiffusion'
```r
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

estimate.mixedRegression-method

Examples

```r
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)
```

Description

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

Usage

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

```r
## 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 ξ

adapt     if TRUE (default), proposal variance is adapted

proposal   proposal density: "normal" (default) or "lognormal" (for positive parameters)

References


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 \( y_i = f(\phi, t_i) + \epsilon_i, \epsilon_i \sim N(0, \gamma^2 s(t_i)) \).

Usage

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

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


Examples

```r
  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 $Z_t = Y_{t_i} + \epsilon_i$, $dY_t = b(\phi, t, Y_t)dt + \gamma \bar{s}(t, Y_t)dW_t, \epsilon_i \sim N(0, \sigma^2), Y_{t_0} = y_0(\phi, t_0)$.

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 $\bar{s}(t, y)$
- prior: list of prior parameters
- start: list of starting values for the Metropolis within Gibbs sampler
hiddenmixedDiffusion-class

S4 class of model informations for hierarchical (mixed) hidden diffusion process

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

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 $\tilde{s}(t, y)$
- `prior` list of prior parameters
- `start` list of starting values for the Metropolis within Gibbs sampler

Examples
```r
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,
               phi = phi)
```
InvMethod

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


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)

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)
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 \(N_t \sim \text{Pois}(\Lambda(t, \xi))\).

Slots

- `theta` parameter \(\theta\)
- `phi` parameter \(\phi\)
- `gamma2` parameter \(\gamma^2\)
- `xi` parameter \(\xi\)
- `bNfun` function \(b(\phi, t, y)\)
- `sNfun` function \(s(\gamma^2, t, y)\)
- `hNfun` 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

```r
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 \( N_t \sim Pois(\Lambda(t, \xi)) \), \( \epsilon_i \sim N(0, \gamma^2 \tilde{s}(t)) \).

**Slots**

- `theta` parameter \( \theta \)
- `gamma2` parameter \( \gamma^2 \)
- `xi` parameter \( \xi \)
- `fun` function \( f(t, N, \theta) \)
- `stNfun` function \( \tilde{s}(t) \)
- `Lambda` function \( \Lambda(t, \xi) \)
- `prior` list of prior parameters
- `start` list of starting values for the Metropolis within Gibbs sampler

**Examples**

```r
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 $\tilde{\theta} = \log(1 + \theta)$
phi parameter $\phi$
gamma2 parameter $\gamma^2$
ox parameter $\xi$

Lambda function $\Lambda(t, \xi)$
prior list of prior parameters for $\phi, \tilde{\theta}, \gamma^2$
priorDensity list of prior density function for $\xi$
start list of starting values for the Metropolis within Gibbs sampler

Examples

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

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 $\tilde{s}(t, y)$
prior list of prior parameters
start list of starting values for the Metropolis within Gibbs sampler
mixedRegression-class

S4 class of model informations for the hierarchical (mixed) regression model

Description

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

Slots

- phi parameter $\phi$
- mu parameter $\mu$
- Omega parameter $\Omega$
- gamma2 parameter $\gamma^2$
- fun function $f(\phi, t)$
- st.fun function $\tilde{s}(t)$
- 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)
b.fun <- function(phi, t) 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("mixedRegression", parameter, prior, start,
b.fun = b.fun, sT.fun = sT.fun, y0.fun = y0.fun)

mixedRegression-class

27
NHPP-class

S4 class of model informations for non-homogeneous Poisson process

Description

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

Slots

- $x_i$ 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

```r
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

```r
## 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

```r
model <- set.to.class("Diffusion", b.fun = function(phi, t, y) phi[1]-phi[2]*y, parameter = list(phi = c(1, 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, mrow = 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))
```

Description

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

Usage

```r
# 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

```r
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))
```

Description

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

Usage

```r
## 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

```r
## 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),
y = 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(par = c(5, 4.5, 4, 2) + 0.1, mrow = c(2,1)), xlab = "iteration")
plot(est, style = "acf", main = "", par2plot = c(TRUE, 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)
```

Description

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

```r
## 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

```r
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), y = 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(par = 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

```r
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

```r
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 = "")
```

---

### Description

Plot method for the Bayesian estimation results of the Merton model.
Usage

```r
## 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: (\(\phi, \tilde{\theta}, \gamma^2, \xi, N\))
- `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

```r
model <- set.to.class("Merton", Lambda = function(t, xi) (t/xi[2])^xi[1],
parameter = list(thetaF = 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 = "")
```

Description

Plot method for the Bayesian estimation results of the hierarchical (mixed) diffusion model.
plot.est.mixedDiffusion-method

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: (µ, Ω, γ²)
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, mtext = 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 method for the Bayesian estimation results

**Description**

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

**Usage**

```r
## 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**

```r
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.NHPP-method

Plot method for the Bayesian estimation results

Description

Plot method for the estimation results of the NHPP.

Usage

```r
## 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: (φ, θ, γ^2, ξ, 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

```r
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 = "")
```
description

Plot method for the estimation results of the regression model.

usage

```r
## 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

```r
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(mat = 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))
```
Bayesian prediction function

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

usage
\[
pred.base(\text{samples}, \text{Fun}, \text{dens}, \text{len} = 100, x0, \text{method} = c(\"vector\", \"free\"),
\text{pred.alg} = c(\"Distribution\", \"Trajectory\"), \text{sampling.alg} = c(\"RejSamp\", \"InvMethod\"), \text{candArea}, \text{grid} = 0.001)
\]

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

value
vector of samples from prediction

references
Prediction for a diffusion process

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

Usage
```r
## 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
Predict, est.hiddenDiffusion-method

References


Examples

```r
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, 10000) # better: 10000
plot(est_diff)
## Not run:
pred_diff <- predict(est_diff, t = seq(0, 1, by = 0.1))
pred_diff2 <- predict(est_diff, b.fun.mat = function(phi, t, y) phi[,1]) # much faster
pred_diff3 <- predict(est_diff, which.series = "current", b.fun.mat = function(phi, t, y) phi[,1],
                      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, \( Z_i = Y_i + \epsilon_i, dY_i = b(\phi, t, Y_i) dt + \gamma \tilde{s}(t, Y_i) dW_i \).

**Usage**

```r
## 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**


---

**Examples**

```r
## 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)
```

---

**Description**

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

**predict.est.hiddenmixedDiffusion-method**

_Prediction for a hierarchical (mixed) hidden diffusion process model_
Usage

```r
## 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


Examples

```r
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))
```
predict.est.jumpDiffusion-method

Example:

```r
# 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")
```

## 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

```r
# 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-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
- **cand.length**: length of candidate samples (if method = "vector"), for jump diffusion
predict.est.jumpRegression-method

**Description**

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

**Examples**

```r
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)
```

**References**


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 "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


Examples

t <- seq(0,1, by = 0.01)
c1 <- 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])
predict.est.Merton-method

```
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)
```

### 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


Examples

```r
c1 <- 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(c1, t = t, y0 = 0.5)
est <- estimate(c1, 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 \( dY_t = b(\phi, t, Y_t)dt + \gamma^{\phi}(t, Y_t)dW_t, \phi \sim N(\mu, \Omega). \)

Usage

```r
## 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
predict, est.mixedDiffusion-method

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


Examples

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

**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)
predict.est.mixedRegression-method

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


Examples

```r
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)
```
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
```r
## 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
predict.est.NHPP-method

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


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")

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))

# 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 $y_i = f(\phi, t_i) + \epsilon_i, \epsilon_i \sim N(0, \gamma^2 s(t_i))$.

**Usage**

```r
## 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**


**prediction.intervals**

Bayesian prediction interval function

**Description**

Calculation of quantiles $\frac{1}{2}$ and $1 - \frac{1}{2}$ from predictive distribution based on distribution function $F(x, x_0, \text{samples})$. 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

**Examples**

```r
  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)
```
**proposal**

*Sampling from lognormal proposal density*

**Description**

Drawing one sample from the lognormal distribution with mean \( \text{parold} \) and standard deviation \( \text{propsd} \). Used in Metropolis Hastings algorithms.

**Usage**

```r
proposal(parold, propsd)
```

**Arguments**

- `parold`: the parameter from the last iteration step
- `propsd`: proposal standard deviation

**Examples**

```r
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**

```r
proposalRatio(parOld, parNew, propsd)
```

**Arguments**

- `parOld`: the parameter from the last iteration step
- `parNew`: drawn candidate
- `propsd`: proposal standard deviation

**Examples**

```r
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 \( y_i = f(\phi, t_i) + \epsilon_i, \epsilon_i \sim N(0, \gamma^2 s(t_i)) \).

Slots
- phi parameter \( \phi \)
- gamma2 parameter \( \gamma^2 \)
- fun function \( f(\phi, t) \)
- st.fun function \( s(t) \)
- prior list of prior parameters
- start list of starting values for the Metropolis within Gibbs sampler

Examples
```r
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
```r
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


Examples

```r
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**

```r
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 \( s \)
- `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

`setNtoNclass` is the central function to define a S4 model class, where the `simulate` and the `estimate` methods build up. Main input parameter is `classNname`, 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 `setNtoNclass(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 \( s(\gamma^2, t, y) = \gamma \tilde{s}(t, y) \). \( \text{sT.fun} \) stands for \( \tilde{s}(t, y) \). In the case of a regression model, "Regression" or "mixedRegression", \( \text{sT.fun} \) means the variance function dependent on \( t \) of the regression error \( \epsilon_i \sim N(0, \sigma^2 \tilde{s}(t)) \). In both cases, default value is \( \text{sT.fun} = \text{function}(t, y) 1 \). \( y0.fun \) 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. \( \text{fun} \) 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

```r
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))
```

Simulation of diffusion process

Description

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

Usage

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

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 graphically

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)

Description

Simulation of a hidden stochastic process model
\[ Z_t = Y_t + \epsilon_t, \quad dY_t = b(\phi, t, Y_t)dt + \gamma \tilde{s}(t, Y_t)dW_t, \quad \epsilon_t \sim N(0, \sigma^2), \quad Y_{t_0} = y_0(\phi, t_0). \]

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 graphically

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)
Simulation of hierarchical (mixed) hidden diffusion model

Description

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

Usage

```r
## 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 graphically

Examples

```r
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)
```

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

```r
## 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
- `mw`: mesh width for finer Euler approximation to simulate time-continuity
- `plot.series`: logical(1), if TRUE, simulated series are depicted graphically

Examples

```r
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 the regression model $y_i = f(t_i, N_{t_i}, \theta) + \epsilon_i$ with $N_i \sim \text{Pois}(\Lambda(t, \xi)), \epsilon_i \sim N(0, \gamma^2 s(t))$.

Usage

```r
## 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 graphically
Examples

```r
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**

```r
## 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
- `plot.series`: logical(1), if TRUE, simulated series are depicted grafically

**Examples**

```r
model <- set.to.class("Merton", parameter = list(theta1 = 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 $dY_t = b(\phi_j, t, Y_t)dt + \gamma \tilde{s}(t, Y_t)dW_t, \phi_j \sim N(\mu, \Omega)$.

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 graphically

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

Usage
## S4 method for signature 'mixedRegression'
simulate(object, nsim = 1, seed = NULL, t,
          plot.series = TRUE)
Simulation of Poisson process

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

### Usage

```r
## 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 graphically

### Examples

```r
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 \( y_i = f(\phi, t_i) + \epsilon_i, \epsilon_i \sim N(0, \gamma^2 \hat{s}(t_i)) \).

**Usage**

```r
### 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 graphically

**Examples**

```r
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**

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

**Arguments**

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


Examples

```r
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

+Topic datasets
  Virkl, 67
+Topic package, (jump) diffusion, mixed (hidden) diffusion, non-homogeneous Poisson process
  BaPreStoPro-package, 3
ad.propSD, 6
BaPreStoPro (BaPreStoPro-package), 3
BaPreStoPro-package, 3
class.to.list, 7
diagnostic, 8
Diffusion-class, 8
dNtToTimes, 9
estimate, 9
estimate,Diffusion-method, 10
estimate,hiddenDiffusion-method, 11
estimate,hiddenmixedDiffusion-method, 12
estimate,jumpDiffusion-method, 13
estimate,jumpRegression-method, 15
estimate,Merton-method, 16
estimate,mixedDiffusion-method, 17
estimate,mixedRegression-method, 18
estimate,NHPP-method, 19
estimate,Regression-method, 20
hiddenDiffusion-class, 21
hiddenmixedDiffusion-class, 22
InvMethod, 23
jumpDiffusion-class, 24
jumpRegression-class, 25
Merton-class, 25
mixedDiffusion-class, 26
mixedRegression-class, 27
NHPP-class, 28
plot,est.Diffusion-method, 28
plot,est.hiddenDiffusion-method, 29
plot,est.hiddenmixedDiffusion-method, 30
plot,est.jumpDiffusion-method, 31
plot,est.jumpRegression-method, 32
plot,est.Merton-method, 33
plot,est.mixedDiffusion-method, 34
plot,est.mixedRegression-method, 36
plot,est.NHPP-method, 37
plot,est.Regression-method, 38
pred.base, 39
predict,est.Diffusion-method, 40
predict,est.hiddenDiffusion-method, 41
predict,est.hiddenmixedDiffusion-method, 42
predict,est.jumpDiffusion-method, 44
predict,est.jumpRegression-method, 45
predict,est.Merton-method, 47
predict,est.mixedDiffusion-method, 48
predict,est.mixedRegression-method, 50
predict,est.NHPP-method, 52
predict,est.Regression-method, 54
prediction.intervals, 55
proposal, 56, 56
proposalRatio, 56
Regression-class, 57
RejSampling, 57
set.to.class, 3, 10, 58
simulate,Diffusion-method, 59
simulate,hiddenDiffusion-method, 60
simulate,hiddenmixedDiffusion-method, 61
simulate, jumpDiffusion-method, 61
simulate, jumpRegression-method, 62
simulate, Merton-method, 63
simulate, mixedDiffusion-method, 64
simulate, mixedRegression-method, 64
simulate, NHPP-method, 65
simulate, Regression-method, 66

TimestoN, 66

Virkler, 67