Package 'yuima'

September 29, 2014

Type Package

Title The YUIMA Project package for SDEs

Version 1.0.36

Date 2014-09-29

Depends methods, zoo, stats4, utils, expm, cubature, mvtnorm

Author YUIMA Project Team

Maintainer Stefano M. Iacus <stefano.iacus@unimi.it>

Description Simulation and Inference for Stochastic Differential Equations

License GPL-2

URL http://R-Forge.R-project.org/projects/yuima/

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-09-29 15:19:08

R topics documented:

`adaBayes` *Adaptive Bayes estimator for the parameters in sde model*

Description

Adaptive Bayes estimator for the parameters in a specific type of sde.

Usage

```
adaBayes(yuima, start, prior,
    lower, upper, method = "nomcmc")
```

Arguments

yuima	a 'yuima' object.
start	initial suggestion for parameter values
prior	a list of prior distributions for the parameters [function].
lower	...
upper	...
method	nomcmc requires package cubature

Details

Calculate the values of the parameters theta1 and theta2. When the quasi-likelihood is too large for the given data, the integral in the Bayes estimator may diverges. For such a case, offset values for the parameters can be specified not to diverge.

Value

vector	a vector of the paramter estimate

Note

This routine is not stable and accurate. Dr.Kamatani is now working for improvements.

Author(s)

The YUIMA Project Team

Examples

```
set.seed(123)

ymodel <- setModel(drift="(-1)*theta2*x", diffusion="sqrt(theta1^2+1)",
                   time.variable="t", state.variable="x", solve.variable="x")
n <- 500
h <- 1/((n)^(2/3))
ysamp <- setSampling(Terminal=(n)^(1/3), n=n)
yuima <- setYuima(model=ymodel, sampling=ysamp)
param.true <- list(theta2=0.3, theta1=0.5)
yuima <- simulate(yuima, xinit=1, true.parameter=param.true)

prior.theta1 <- function(theta2)
  1*(theta2 > 0 & theta2 < 1)

prior.theta2 <- function(theta1)
  1*(theta1 > 0 & theta1 < 1)

prior <- list(theta2=list(measure.type="code",df="dunif(z,0,1)"),
theta1=list(measure.type="code",df="dunif(z,0,1)"))

param.init <- list(theta2=0.35,theta1=0.52)
```

```
lower = c(0,0)
upper=c(1,1)

bayes1 <- adaBayes(yuima, start=param.init, prior=prior, method="nomcmc")
bayes1@coef

mle1 <- qmle(yuima, start=param.init, lower=list(theta1=0,theta2=0),
      upper=list(theta1=1,theta2=1), method="L-BFGS-B")
mle1@coef
```

`asymptotic_term` *asymptotic expansion of the expected value of the functional*

Description

calculate the fisrt and second term of asymptotic expansion of the functional mean.

Usage

```
asymptotic_term(yuima, block=100, rho, g, expand.var="e")
```

Arguments

`yuima`	an yuima object containing model and functional.
`block`	the number of trapezoids for integrals.
`rho`	specify discounting factor in mean integral.
`g`	arbitrary measurable function for mean integral.
`expand.var`	default expand.var="e".

Details

Calculate the first and second term of asymptotic expansion of the expected value of the functional associated with a sde. The returned value d0 + epsilon * d1 is approximation of the expected value.

Value

`terms`	list of 1st and 2nd asymptotic terms, terms$d0 and terms$d1.

Note

we need to fix this routine.

Author(s)

YUIMA Project Team

Examples

```
# to the Black-Scholes economy:
# dXt^e = Xt^e * dt + e * Xt^e * dWt
diff.matrix <- "x*e"
model <- setModel(drift = "x", diffusion = diff.matrix)
# call option is evaluated by averating
# max{ (1/T)*int_0^T Xt^e dt, 0}, the first argument is the functional of interest:
Terminal <- 1
xinit <- c(1)
f <- list( c(expression(x/Terminal)), c(expression(0)))
F <- 0
division <- 1000
e <- .3
yuima <- setYuima(model = model, sampling = setSampling(Terminal=Terminal, n=division))
yuima <- setFunctional( yuima, f=f,F=F, xinit=xinit,e=e)

# asymptotic expansion
rho <- expression(0)
F0 <- F0(yuima)
get_ge <- function(x,epsilon,K,F0){
  tmp <- (F0 - K) + (epsilon * x)
  tmp[(epsilon * x) < (K-F0)] <- 0
  return( tmp )
}
g <- function(x) get_ge(x,epsilon=e,K=1,F0=F0)
set.seed(123)
asymp <- asymptotic_term(yuima, block=10, rho,g)
asymp
sum(asymp$d0 + e * asymp$d1)


### An example of multivariate case: Heston model
## a <- 1;C <- 1;d <- 10;R<-.1
## diff.matrix <- matrix( c("x1*sqrt(x2)*e", "e*R*sqrt(x2)",0,"sqrt(x2*(1-R^2))*e"), 2,2)
## model <- setModel(drift = c("a*x1","C*(10-x2)"),
## diffusion = diff.matrix,solve.variable=c("x1","x2"),state.variable=c("x1","x2"))
## call option is evaluated by averating
## max{ (1/T)*int_0^T Xt^e dt, 0}, the first argument is the functional of interest:
##
## Terminal <- 1
## xinit <- c(1,1)
##
## f <- list( c(expression(0), expression(0)),
## c(expression(0), expression(0)) , c(expression(0), expression(0))  )
## F <- expression(x1,x2)
##
## division <- 1000
## e <- .3
##
## yuima <- setYuima(model = model, sampling = setSampling(Terminal=Terminal, n=division))
## yuima <- setFunctional( yuima, f=f,F=F, xinit=xinit,e=e)
##
```

```
## rho <- expression(x1)
## F0 <- F0(yuima)
## get_ge <- function(x){
##  return( max(x[1],0))
## }
## g <- function(x) get_ge(x)
## set.seed(123)
## asymp <- asymptotic_term(yuima, block=10, rho,g)
## sum(asymp$d0 + e * asymp$d1)
```

`bns.test` *Barndorff-Nielsen and Shephard's Test for the Presence of Jumps Using Bipower Variation*

Description

Tests the presence of jumps using the statistic proposed in Barndorff-Nielsen and Shephard (2004,2006) for each components.

Usage

```
bns.test(yuima, r = rep(1, 4), type = "standard", adj = TRUE)
```

Arguments

`yuima`	an object of `yuima-class` or `yuima.data-class`.
`r`	a vector of non-negative numbers or a list of vectors of non-negative numbers. Theoretically, it is necessary that `sum(r)=4` and `max(r)<2`.
`type`	type of the test statistic to use. `standard` is default.
`adj`	logical; if `TRUE`, the maximum adjustment suggested in Barndorff-Nielsen and Shephard (2004) is applied to the test statistic when `type` is equal to either "`log`" or "`ratio`".

Details

For the `i`-th component, the test statistic is equal to the `i`-th component of `sqrt(n)*(mpv(yuima,2)-mpv(yuima,c(1,1)))/`
when `type="standard"`, `sqrt(n)*log(mpv(yuima,2)/mpv(yuima,c(1,1)))/sqrt(vartheta*mpv(yuima,r)/mpv(yuim`
when `type="log"` and `sqrt(n)*(1-mpv(yuima,c(1,1))/mpv(yuima,2))/sqrt(vartheta*mpv(yuima,r)/mpv(yuima,c`
when `type="ratio"`. Here, `n` is equal to the length of the `i`-th component of the `zoo.data` of `yuima` minus 1 and `vartheta` is `pi^2/4+pi-5`. When `adj=TRUE`, `mpv(yuima,r)[i]/mpv(yuima,c(1,1))^2)[i]` is replaced with 1 if it is less than 1.

Value

A list with the same length as the `zoo.data` of `yuima`. Each components of the list has class "`htest`" and contains the following components:

`statistic`	the value of the test statistic of the corresponding component of the `zoo.data` of `yuima`.
`p.value`	an approximate p-value for the test of the corresponding component.
`method`	the character string "`Barndorff-Nielsen and Shephard jump test`".
`data.name`	the character string "`xi`", where `i` is the number of the component.

Note

Theoretically, this test may be invalid if sampling is irregular.

Author(s)

The YUIMA Project Team

References

Barndorff-Nielsen, O. E. and Shephard, N. (2004) Power and bipower variation with stochastic volatility and jumps, *Journal of Financial Econometrics*, **2**, no. 1, 1–37.

Barndorff-Nielsen, O. E. and Shephard, N. (2006) Econometrics of testing for jumps in financial economics using bipower variation, *Journal of Financial Econometrics*, **4**, no. 1, 1–30.

Huang, X. and Tauchen, G. (2005) The relative contribution of jumps to total price variance, *Journal of Financial Econometrics*, **3**, no. 4, 456–499.

See Also

`mpv`

Examples

```
set.seed(123)

# One-dimensional case
## Model: dXt=t*dWt+t*dzt,
## where zt is a compound Poisson process with intensity 5 and jump sizes distribution N(0,0.1).

model <- setModel(drift=0,diffusion="t",jump.coeff="t",measure.type="CP",
                  measure=list(intensity=5,df=list("dnorm(z,0,sqrt(0.1))")),
                  time.variable="t")

yuima.samp <- setSampling(Terminal = 1, n = 390)
yuima <- setYuima(model = model, sampling = yuima.samp)
yuima <- simulate(yuima)
plot(yuima) # The path seems to involve some jumps
```

```
bns.test(yuima) # standard type

bns.test(yuima,type="log") # log type

bns.test(yuima,type="ratio") # ratio type

# Multi-dimensional case
## Model: dXkt=t*dWk_t (k=1,2,3) (no jump case).

diff.matrix <- diag(3)
diag(diff.matrix) <- c("t","t","t")
model <- setModel(drift=c(0,0,0),diffusion=diff.matrix,time.variable="t",
                  solve.variable=c("x1","x2","x3"))

yuima.samp <- setSampling(Terminal = 1, n = 390)
yuima <- setYuima(model = model, sampling = yuima.samp)
yuima <- simulate(yuima)
plot(yuima)

bns.test(yuima)
```

`carma.info-class` *Class for information about CARMA(p,q) model*

Description

The `carma.info-class` is a class of the **yuima** package.

Details

The `carma.info-class` object cannot be directly specified by the user but it is constructed when the `yuima.carma-class` object is constructed via `setCarma`.

Slots

`p`: Number of autoregressive coefficients.

`q`: Number of moving average coefficients.

`loc.par`: Label of location coefficient.

`scale.par`: Label of scale coefficient.

`ar.par`: Label of autoregressive coefficients.

`ma.par`: Label of moving average coefficients.

`lin.par`: Label of linear coefficients.

`Carma.var`: Label of the observed process.

`Latent.var`: Label of the unobserved process.

`XinExpr`: Logical variable. If `XinExpr=FALSE`, the starting condition of `Latent.var` is zero otherwise each component of Latent.var has a parameter as a starting point.

Author(s)

The YUIMA Project Team

`CarmaNoise` *Estimation for the underlying Levy in a carma model*

Description

Retrieve the increment of the underlying Levy for the carma(p,q) process using the approach developed in Brockwell et al.(2011)

Usage

```
CarmaNoise(yuima, param, data=NULL, NoNeg.Noise=FALSE)
```

Arguments

`yuima`	a yuima object or an object of `yuima.carma-class`.
`param`	`list` of parameters for the carma.
`data`	an object of class `yuima.data-class` contains the observations available at uniformly spaced time. If `data=NULL`, the default, the 'CarmaNoise' uses the data in an object of `yuima.data-class`.
`NoNeg.Noise`	Estimate a non-negative Levy-Driven Carma process. By default `NoNeg.Noise=FALSE`.

Value

`incr.Levy`	a numeric object contains the estimated increments.

Note

The function `qmle` uses the function `CarmaNoise` for estimation of underlying Levy in the carma model.

Author(s)

The YUIMA Project Team

References

Brockwell, P., Davis, A. R. and Yang. Y. (2011) Estimation for Non-Negative Levy-Driven CARMA Process, *Journal of Business And Economic Statistics*, **29** - 2, 250-259.

Examples

```
## Not run:
#Ex.1: Carma(p=3, q=0) process driven by a brownian motion.

mod0<-setCarma(p=3,q=0)

# We fix the autoregressive and moving average parameters
# to ensure the existence of a second order stationary solution for the process.

true.parm0 <-list(a1=4,a2=4.75,a3=1.5,b0=1)

# We simulate a trajectory of the Carma model.

numb.sim<-1000
samp0<-setSampling(Terminal=100,n=numb.sim)
set.seed(100)
incr.W<-matrix(rnorm(n=numb.sim,mean=0,sd=sqrt(100/numb.sim)),1,numb.sim)

sim0<-simulate(mod0,
               true.parameter=true.parm0,
               sampling=samp0, increment.W=incr.W)

#Applying the CarmaNoise

system.time(
  inc.Levy0<-CarmaNoise(sim0,true.parm0)
)

# We compare the orginal with the estimated noise increments

par(mfrow=c(1,2))
plot(t(incr.W)[1:998],type="l", ylab="",xlab="time")
title(main="True Brownian Motion",font.main="1")
plot(inc.Levy0,type="l", main="Filtered Brownian Motion",font.main="1",ylab="",xlab="time")

# Ex.2: carma(2,1) driven  by a compound poisson
# where jump size is normally distributed and
# the lambda is equal to 1.

mod1<-setCarma(p=2,
               q=1,
               measure=list(intensity="Lamb",df=list("dnorm(z, 0, 1)")),
               measure.type="CP")

true.parm1 <-list(a1=1.39631, a2=0.05029,
                  b0=1,b1=2,
                  Lamb=1)

# We generate a sample path.

samp1<-setSampling(Terminal=100,n=200)
set.seed(123)
```

```
sim1<-simulate(mod1,
               true.parameter=true.parm1,
               sampling=samp1)

# We estimate the parameter using qmle.
carmaopt1 <- qmle(sim1, start=true.parm1)
summary(carmaopt1)
# Internally qmle uses CarmaNoise. The result is in
plot(carmaopt1)

# Ex.3: Carma(p=2,q=1) with scale and location parameters
# driven by a Compound Poisson
# with jump size normally distributed.
mod2<-setCarma(p=2,
               q=1,
               loc.par="mu",
               scale.par="sig",
               measure=list(intensity="Lamb",df=list("dnorm(z, 0, 1)")),
               measure.type="CP")

true.parm2 <-list(a1=1.39631,
                  a2=0.05029,
                  b0=1,
                  b1=2,
                  Lamb=1,
                  mu=0.5,
                  sig=0.23)
# We simulate the sample path
set.seed(123)
sim2<-simulate(mod2,
               true.parameter=true.parm2,
               sampling=samp1)

# We estimate the Carma and we plot the underlying noise.

carmaopt2 <- qmle(sim2, start=true.parm2)
summary(carmaopt2)

# Increments estimated by CarmaNoise
plot(carmaopt2)

## End(Not run)
```

---

cce *Nonsynchronous Cumulative Covariance Estimator*

Description

This function estimates the covariance between two Ito processes when they are observed at discrete times possibly nonsynchronously. It can apply to irregularly sampled one-dimensional data as a special case.

Usage

```
cce(x, method="HY", theta, kn, g=function(x)min(x,1-x), refreshing = TRUE,
    cwise = TRUE, delta = 0, adj = TRUE, K, c.two, J = 1, c.multi, kernel, H,
    c.RK, eta = 3/5, m = 2, ftregion = 0, opt.method = "BFGS", vol.init = NA,
    covol.init = NA, nvar.init = NA, ncov.init = NA, ..., mn, alpha = 0.4,
    frequency = 300, avg = TRUE, threshold, utime, psd = FALSE)
```

Arguments

x	an object of yuima-class or yuima.data-class.
method	the method to be used. See 'Details'.
theta	a numeric vector or matrix. If it is a matrix, each of its components indicates the tuning parameter which determines the pre-averaging window lengths kn to be used for estimating the corresponding component. If it is a numeric vector, it is converted to a matrix as (C+t(C))/2, where C=matrix(theta,d,d) and d=dim(x). The default value is 0.15 for the method "PHY" or "PTHY" following Christensen et al. (2011), while it is 1 for the method "MRC" following Christensen et al. (2010).
kn	an integer-valued vector or matrix indicating the pre-averaging window length(s). For the methods "PHY" or "PTHY", see 'Details' for the default value. For the method "MRC", the default value is ceiling(theta*n^(1+delta)), where n is the number of the refresh times associated with the data minus 1.
g	a function indicating the weight function to be used. The default value is the Bartlett window: function(x)min(x,1-x).
refreshing	logical. If TRUE, the data is pre-synchronized by the next-tick interpolation in the refresh times.
cwise	logical. If TRUE, the estimator is calculated componentwise.
delta	a non-negative number indicating the order of the pre-averaging window length(s) kn.
adj	logical. If TRUE, a finite-sample adjustment is performed. For the method "MRC", see Christensen et al. (2010) for details. For the method "TSCV", see Zhang (2011) and Zhang et al. (2005) for details.
K	a positive integer indicating the large time-scale parameter. The default value is ceiling(c.two*n^(2/3)), where n is the number of the refresh times associated with the data minus 1.
c.two	a positive number indicating the tuning parameter which determines the scale of the large time-scale parameter K. The default value is the average of the numeric vector each of whose components is the roughly estimated optimal value in the sense of the minimizer of the theoretical asymptotic variance of the estimator of the corresponding diagonal component. The theoretical asymptotic variance is considered in the standard case and given in Zhang et al. (2005).
J	a positive integer indicating the small time-scale parameter.
c.multi	a numeric vector or matrix. If it is a matrix, each of its components indicates the tuning parameter which determines (the scale of) the number of the time scales to be used for estimating the corresponding component. If it is a numeric vector,

it is converted to a matrix as `(C+t(C))/2`, where `C=matrix(c.multi,d,d)` and `d=dim(x)`. The default value is the numeric vector each of whose components is the roughly estimated optimal value in the sense of minimizing the theoretical asymptotic variance of the estimator of the corresponding diagonal component. The theoretical asymptotic variance is considered in the standard case and given in Bibinger et al. (2012).

`kernel` a function indicating the kernel function to be used. The default value is the Parzan kernel, which is recommended in Barndorff-Nielsen et al. (2009, 2011).

`H` a positive number indicating the bandwidth parameter. The default value is `c.RK*n^eta`, where `n` is the number of the refresh times associated with the data minus 1.

`c.RK` a positive number indicating the tuning parameter which determines the scale of the bandwidth parameter `H`. The default value is the average of the numeric vector each of whose components is the roughly estimated optimal value in the sense of minimizing the theoretical asymptotic variance of the estimator of the corresponding diagonal component. The theoretical asymptotic variance is considered in the standard case and given in Barndorff-Nielsen et al. (2009, 2011).

`eta` a positive number indicating the tuning parameter which determines the order of the bandwidth parameter `H`.

`m` a positive integer indicating the number of the end points to be jittered.

`ftregion` a non-negative number indicating the length of the flat-top region. `ftregion=0` (the default) means that a non-flat-top realized kernel studied in Barndorff-Nielsen et al. (2011) is used. `ftregion=1/H` means that a flat-top realized kernel studied in Barndorff-Nielsen et al. (2008) is used. See Varneskov (2011) for other values.

`opt.method` passed to the argument `method` of constrOptim.

`vol.init` a numeric vector each of whose components indicates the initial value to be used to estimate the integrated volatility of the corresponding component, which is passed to the optimizer. Components involving `NA` are filled with the rough estimated true values.

`covol.init` a numeric matrix each of whose columns indicates the initial value to be used to estimate the integrated covariance of the corresponding component, which is passed to the optimizer. Components involving `NA` are filled with the rough estimated true values.

`nvar.init` a numeric vector each of whose components indicates the initial value to be used to estimate the variance of noise of the corresponding component, which is passed to the optimizer. Components involving `NA` are filled with the rough estimated true values.

`ncov.init` a numeric matrix each of whose columns indicates the initial value to be used to estimate the covariance of noise of the corresponding component, which is passed to the optimizer. Components involving `NA` are filled with the rough estimated true values.

`...` other named arguments to be passed to constrOptim.

`mn` a positive integer indicating the number of terms to be used for calculating the SIML estimator. The default value is `ceiling(n^alpha)`, where `n` is the number of the refresh times associated with the data minus 1.

alpha	a postive number indicating the order of mn.
frequency	a positive integer indicating the frequency (seconds) of the calendar time sampling to be used.
avg	logical. If TRUE, the averaged subsampling estimator is calculated. Otherwise the simple sparsely subsampled estimator is calculated.
threshold	a numeric vector or list indicating the threshold parameter(s). Each of its components indicates the threshold parameter or process to be used for estimating the corresponding component. If it is a numeric vector, the elements in threshold are recycled if there are two few elements in threshold. The default value is determined following Koike (2013a) (for the method "THY") and Koike (2013b) (for the method "PTHY").
utime	a positive number indicating what seconds the unit time means. The default value is 23400 (this means 6.5 hours) if the time indices of the (first component) of x is numeric, otherwise 1.
psd	logical. If TRUE, the estimated covariance matrix C is converted to (C%*%C)^(1/2) for ensuring the positive semi-definiteness.

Details

This function is a method for objects of yuima.data-class and yuima-class. It extracts the data slot when applied to a an object of yuima-class.

Typical usages are

```
cce(x,psd=FALSE)
cce(x,method="PHY",theta,kn,g,refreshing=TRUE,cwise=TRUE,psd=FALSE)
cce(x,method="MRC",theta,kn,g,delta=0,avg=TRUE,psd=FALSE)
cce(x,method="TSCV",K,c.two,J=1,adj=TRUE,utime,psd=FALSE)
cce(x,method="GME",c.multi,utime,psd=FALSE)
cce(x,method="RK",kernel,H,c.RK,eta=3/5,m=2,ftregion=0,utime,psd=FALSE)
cce(x,method="QMLE",opt.method="BFGS",vol.init=NULL,covol.init=NULL,
 nvar.init=NULL,ncov.init=NULL,...,utime,psd=FALSE)
cce(x,method="SIML",mn,alpha=0.4,psd=FALSE)
cce(x,method="THY",threshold,psd=FALSE)
cce(x,method="PTHY",theta,kn,g,threshold,refreshing=TRUE,cwise=TRUE,psd=FALSE)
cce(x,method="SRC",frequency=300,avg=TRUE,utime,psd=FALSE)
cce(x,method="SBPC",frequency=300,avg=TRUE,utime,psd=FALSE)
```

The default method is method "HY", which is an implementation of the Hayashi-Yoshida estimator proposed in Hayashi and Yoshida (2005).

Method "PHY" is an implementation of the Pre-averaged Hayashi-Yoshida estimator proposed in Christensen et al. (2010).

Method "MRC" is an implementation of the Modulated Realized Covariance based on refresh time sampling proposed in Christensen et al. (2010).

Method "TSCV" is an implementation of the previous tick Two Scales realized CoVariance based on

refresh time sampling proposed in Zhang (2011).

Method "GME" is an implementation of the Generalized Multiscale Estimator proposed in Bibinger (2011).

Method "RK" is an implementation of the multivariate Realized Kernel based on refresh time sampling proposed in Barndorff-Nielsen et al. (2011).

Method "QMLE" is an implementation of the nonparametric Quasi Maximum Likelihood Estimator proposed in Ait-Sahalia et al. (2010).

Method "SIML" is an implementation of the Separating Information Maximum Likelihood estimator proposed in Kunitomo and Sato (2008) with the basis of refresh time sampling.

Method "THY" is an implementation of the Truncated Hayashi-Yoshida estimator proposed in Mancini and Gobbi (2012).

Method "PTHY" is an implementation of the Pre-averaged Truncated Hayashi-Yoshida estimator, which is a thresholding version of the pre-averaged Hayashi-Yoshida estimator.

Method "SRC" is an implementation of the calendar time Subsampled Realized Covariance.

Method "SBPC" is an implementation of the calendar time Subsampled realized BiPower Covariation.

The rough estimation procedures for selecting the default values of the tuning parameters are based on those in Barndorff-Nielsen et al. (2009).

For the methods "PHY" or "PTHY", the default value of kn changes depending on the values of refreshing and cwise. If both refreshing and cwise are TRUE (the default), the default value of kn is given by the matrix ceiling(theta*N), where N is a matrix whose diagonal components are identical with the vector length(x)-1 and whose (i, j)-th component is identical with the number of the refresh times associated with i-th and j-th components of x minus 1. If refreshing is TRUE while cwise is FALSE, the default value of kn is given by ceiling(mean(theta)*sqrt(n)), where n is the number of the refresh times associated with the data minus 1. If refreshing is FALSE while cwise is TRUE, the default value of kn is given by the matrix ceiling(theta*N0), where N0 is a matrix whose diagonal components are identical with the vector length(x)-1 and whose (i, j)-th component is identical with (length(x)[i]-1)+(length(x)[j]-1). If both refreshing and cwise are FALSE, the default value of kn is given by ceiling(mean(theta)*sqrt(sum(length(x)-1))) (following Christensen et al. (2011)).

Value

A list with components:

covmat	the estimated covariance matrix
cormat	the estimated correlation matrix

Note

The example shows the central limit theorem for the nonsynchronous covariance estimator. Estimation of the asymptotic variance with p-values and the second-order correction will be provided in a future version of the package.

Author(s)

The YUIMA Project Team

References

Ait-Sahalia, Y., Fan, J. and Xiu, D. (2010) High-frequency covariance estimates with noisy and asynchronous financial data, *Journal of the American Statistical Association*, **105**, no. 492, 1504–1517.

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A. and Shephard, N. (2008) Designing realised kernels to measure the ex-post variation of equity prices in the presence of noise, *Econometrica*, **76**, no. 6, 1481–1536.

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A. and Shephard, N. (2009) Realized kernels in practice: trades and quotes, *Econometrics Journal*, **12**, C1–C32.

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A. and Shephard, N. (2011) Multivariate realised kernels: Consistent positive semi-definite estimators of the covariation of equity prices with noise and non-synchronous trading, *Journal of Econometrics*, **162**, 149–169.

Bibinger, M. (2011) Efficient covariance estimation for asynchronous noisy high-frequency data, *Scandinavian Journal of Statistics*, **38**, 23–45.

Bibinger, M. (2012) An estimator for the quadratic covariation of asynchronously observed Ito processes with noise: asymptotic distribution theory, *Stochastic processes and their applications*, **122**, 2411–2453.

Christensen, K., Kinnebrock, S. and Podolskij, M. (2010) Pre-averaging estimators of the ex-post covariance matrix in noisy diffusion models with non-synchronous data, *Journal of Econometrics*, **159**, 116–133.

Christensen, K., Podolskij, M. and Vetter, M. (2011) On covariation estimation for multivariate continuous Ito semimartingales with noise in non-synchronous observation schemes, CREATES Research Paper 2011-53, Aarhus University.

Hayashi, T. and Yoshida, N. (2005) On covariance estimation of non-synchronously observed diffusion processes, *Bernoulli*, **11**, no. 2, 359–379.

Hayashi, T. and Yoshida, N. (2008) Asymptotic normality of a covariance estimator for nonsynchronously observed diffusion processes, *Annals of the Institute of Statistical Mathematics*, **60**, no. 2, 367–406.

Koike, Y. (2013a) An estimator for the cumulative co-volatility of asynchronously observed semimartingales with jumps, To apper in *Scandinavian Journal of Statistics*.

Koike, Y. (2013b) Estimation of integrated covariances in the simultaneous presence of nonsynchronicity, microstructure noise and jumps, arXiv:1302.5202.

Kunitomo, N. and Sato, S. (2008) Separating information maximum likelihood estimation of realized volatility and covariance with micro-market noise, CIRJE Discussion papers CIRJE-F-581, University of Tokyo.

Mancini, C. and Gobbi, F. (2012) Identifying the Brownian covariation from the co-jumps given discrete observations, *Econometric Theory*, **28**, 249–273.

Varneskov, R. T. (2011) Flat-top realized kernel estimation of quadratic covariation with non-synchronous and noisy asset prices, CREATES Research Paper 2011-35, Aarhus University.

Zhang, L. (2011) Estimating covariation: Epps effect, microstructure noise, *Journal of Econometrics*, **160**, 33–47.

Zhang, L., Mykland, P. A. and Ait-Sahalia, Y. (2005) A tale of two time scales: Determining integrated volatility with noisy high-frequency data, *Journal of the American Statistical Association*, **100**, no. 472, 1394–1411.

See Also

setModel

Examples

```
## Set a model
diff.coef.1 <- function(t, x1 = 0, x2 = 0) sqrt(1+t)
diff.coef.2 <- function(t, x1 = 0, x2 = 0) sqrt(1+t^2)
cor.rho <- function(t, x1 = 0, x2 = 0) sqrt(1/2)
diff.coef.matrix <- matrix(c("diff.coef.1(t,x1,x2)",
"diff.coef.2(t,x1,x2) * cor.rho(t,x1,x2)",
"", "diff.coef.2(t,x1,x2) * sqrt(1-cor.rho(t,x1,x2)^2)"), 2, 2)
cor.mod <- setModel(drift = c("", ""),
diffusion = diff.coef.matrix,solve.variable = c("x1", "x2"))

set.seed(111)

## We use a function poisson.random.sampling to get observation by Poisson sampling.
yuima.samp <- setSampling(Terminal = 1, n = 1200)
yuima <- setYuima(model = cor.mod, sampling = yuima.samp)
yuima <- simulate(yuima)
psample<- poisson.random.sampling(yuima, rate = c(0.2,0.3), n = 1000)

## cce takes the psample and returns an estimate of the quadratic covariation.
cce(psample)$covmat[1, 2]
##cce(psample)[1, 2]

## True value of the quadratic covariation.
cc.theta <- function(T, sigma1, sigma2, rho) {
  tmp <- function(t) return(sigma1(t) * sigma2(t) * rho(t))
integrate(tmp, 0, T)
}

theta <- cc.theta(T = 1, diff.coef.1, diff.coef.2, cor.rho)$value
cat(sprintf("theta =%.5f\n", theta))

names(psample@zoo.data)
```

```
# Example. A stochastic differential equation with nonlinear feedback.

## Set a model
drift.coef.1 <- function(x1,x2) x2
drift.coef.2 <- function(x1,x2) -x1
drift.coef.vector <- c("drift.coef.1","drift.coef.2")
diff.coef.1 <- function(t,x1,x2) sqrt(abs(x1))*sqrt(1+t)
diff.coef.2 <- function(t,x1,x2) sqrt(abs(x2))
cor.rho <- function(t,x1,x2) 1/(1+x1^2)
diff.coef.matrix <- matrix(c("diff.coef.1(t,x1,x2)",
"diff.coef.2(t,x1,x2) * cor.rho(t,x1,x2)","",
"diff.coef.2(t,x1,x2) * sqrt(1-cor.rho(t,x1,x2)^2)"), 2, 2)
cor.mod <- setModel(drift = drift.coef.vector,
 diffusion = diff.coef.matrix,solve.variable = c("x1", "x2"))

## Generate a path of the process
set.seed(111)
yuima.samp <- setSampling(Terminal = 1, n = 10000)
yuima <- setYuima(model = cor.mod, sampling = yuima.samp)
yuima <- simulate(yuima, xinit=c(2,3))
plot(yuima)


## The "true" value of the quadratic covariation.
cce(yuima)

## We use the function poisson.random.sampling to generate nonsynchronous
## observations by Poisson sampling.
psample<- poisson.random.sampling(yuima, rate = c(0.2,0.3), n = 3000)

## cce takes the psample to return an estimated value  of the quadratic covariation.
## The off-diagonal elements are the value of the Hayashi-Yoshida estimator.
cce(psample)
```

CPoint *Volatility structural change point estimator*

Description

Volatility structural change point estimator

Usage

```
CPoint(yuima, param1, param2, print=FALSE, symmetrized=FALSE, plot=FALSE)
qmleL(yuima, t,  ...)
qmleR(yuima, t,  ...)
```

Arguments

yuima	a yuima object.
param1	parameter values before the change point t
param2	parameter values after the change point t
plot	plot test statistics? Default is FALSE.
print	print some debug output. Default is FALSE.
t	time value. See Details.
symmetrized	if TRUE uses the symmetrized version of the quasi maximum-likelihood approximation.
...	passed to qmle method. See Examples.

Details

CPoint estimates the change point using quasi-maximum likelihood approach.

Function qmleL estimates the parameters in the diffusion matrix using observations up to time t.

Function qmleR estimates the parameters in the diffusion matrix using observations from time t to the end.

Arguments in both qmleL and qmleR follow the same rules as in qmle.

Value

ans	a list with change point instant, and paramters before and after the change point.

Author(s)

The YUIMA Project Team

Examples

```
## Not run:
diff.matrix <- matrix(c("theta1.1*x1","0*x2","0*x1","theta1.2*x2"), 2, 2)

drift.c <- c("1-x1", "3-x2")
drift.matrix <- matrix(drift.c, 2, 1)

ymodel <- setModel(drift=drift.matrix, diffusion=diff.matrix, time.variable="t",
state.variable=c("x1", "x2"), solve.variable=c("x1", "x2"))
n <- 1000

set.seed(123)
```

```
t1 <- list(theta1.1=.1, theta1.2=0.2)
t2 <- list(theta1.1=.6, theta1.2=.6)

tau <- 0.4
ysamp1 <- setSampling(n=tau*n, Initial=0, delta=0.01)
yuima1 <- setYuima(model=ymodel, sampling=ysamp1)
yuima1 <- simulate(yuima1, xinit=c(1, 1), true.parameter=t1)

x1 <- yuima1@data@zoo.data[[1]]
x1 <- as.numeric(x1[length(x1)])
x2 <- yuima1@data@zoo.data[[2]]
x2 <- as.numeric(x2[length(x2)])

ysamp2 <- setSampling(Initial=n*tau*0.01, n=n*(1-tau), delta=0.01)
yuima2 <- setYuima(model=ymodel, sampling=ysamp2)

yuima2 <- simulate(yuima2, xinit=c(x1, x2), true.parameter=t2)


yuima <- yuima1
yuima@data@zoo.data[[1]] <- c(yuima1@data@zoo.data[[1]], yuima2@data@zoo.data[[1]][-1])
yuima@data@zoo.data[[2]] <- c(yuima1@data@zoo.data[[2]], yuima2@data@zoo.data[[2]][-1])

plot(yuima)

# estimation of change point for given parameter values
t.est <- CPoint(yuima,param1=t1,param2=t2, plot=TRUE)


low <- list(theta1.1=0, theta1.2=0)

# first state estimate of parameters using small
# portion of data in the tails
tmp1 <- qmleL(yuima,start=list(theta1.1=0.3,theta1.2=0.5),t=1.5,
        lower=low, method="L-BFGS-B")
tmp1
tmp2 <- qmleR(yuima,start=list(theta1.1=0.3,theta1.2=0.5), t=8.5,
        lower=low, method="L-BFGS-B")
tmp2


# first stage changepoint estimator
t.est2 <- CPoint(yuima,param1=coef(tmp1),param2=coef(tmp2))
t.est2$tau


# second stage estimation of parameters given first stage
# change point estimator
tmp11 <- qmleL(yuima,start=as.list(coef(tmp1)), t=t.est2$tau-0.1,
 lower=low, method="L-BFGS-B")
tmp11

tmp21 <- qmleR(yuima,start=as.list(coef(tmp2)), t=t.est2$tau+0.1,
```

```
 lower=low, method="L-BFGS-B")
tmp21

# second stage estimator of the change point
CPoint(yuima,param1=coef(tmp11),param2=coef(tmp21))


## One dimensional example: non linear case
diff.matrix <- matrix("(1+x1^2)^theta1", 1, 1)
drift.c <- c("x1")

ymodel <- setModel(drift=drift.c, diffusion=diff.matrix, time.variable="t",
state.variable=c("x1"), solve.variable=c("x1"))
n <- 500

set.seed(123)

y0 <- 5 # initial value
theta00 <- 1/5
gamma <- 1/4


theta01 <- theta00+n^(-gamma)


t1 <- list(theta1= theta00)
t2 <- list(theta1= theta01)

tau <- 0.4
ysamp1 <- setSampling(n=tau*n, Initial=0, delta=1/n)
yuima1 <- setYuima(model=ymodel, sampling=ysamp1)
yuima1 <- simulate(yuima1, xinit=c(5), true.parameter=t1)
x1 <- yuima1@data@zoo.data[[1]]
x1 <- as.numeric(x1[length(x1)])

ysamp2 <- setSampling(Initial=tau, n=n*(1-tau), delta=1/n)
yuima2 <- setYuima(model=ymodel, sampling=ysamp2)

yuima2 <- simulate(yuima2, xinit=c(x1), true.parameter=t2)


yuima <- yuima1
yuima@data@zoo.data[[1]] <- c(yuima1@data@zoo.data[[1]], yuima2@data@zoo.data[[1]][-1])


plot(yuima)


t.est <- CPoint(yuima,param1=t1,param2=t2)
t.est$tau

low <- list(theta1=0)
upp <- list(theta1=1)
```

```
# first state estimate of parameters using small
# portion of data in the tails
tmp1 <- qmleL(yuima,start=list(theta1=0.5),t=.15,lower=low, upper=upp,method="L-BFGS-B")
tmp1
tmp2 <- qmleR(yuima,start=list(theta1=0.5), t=.85,lower=low, upper=upp,method="L-BFGS-B")
tmp2



# first stage changepoint estimator
t.est2 <- CPoint(yuima,param1=coef(tmp1),param2=coef(tmp2))
t.est2$tau


# second stage estimation of parameters given first stage
# change point estimator
tmp11 <- qmleL(yuima,start=as.list(coef(tmp1)), t=t.est2$tau-0.1,
   lower=low, upper=upp,method="L-BFGS-B")
tmp11

tmp21 <- qmleR(yuima,start=as.list(coef(tmp2)), t=t.est2$tau+0.1,
  lower=low, upper=upp,method="L-BFGS-B")
tmp21


# second stage estimator of the change point
CPoint(yuima,param1=coef(tmp11),param2=coef(tmp21),plot=TRUE)

## End(Not run)
```

`lasso` *Adaptive LASSO estimation for stochastic differential equations*

Description

Adaptive LASSO estimation for stochastic differential equations.

Usage

```
lasso(yuima, lambda0, start, delta=1, ...)
```

Arguments

`yuima`	a yuima object.
`lambda0`	a named list with penalty for each parameter.
`start`	initial values to be passed to the optimizer.
`delta`	controls the amount of shrinking in the adaptive sequences.
`...`	passed to `optim` method. See Examples.

Details

lasso behaves more likely the standard qmle function in and argument method is one of the methods available in optim.

From initial guess of QML estimates, performs adaptive LASSO estimation using the Least Squares Approximation (LSA) as in Wang and Leng (2007, JASA).

Value

ans	a list with both QMLE and LASSO estimates.

Author(s)

The YUIMA Project Team

Examples

```
##multidimension case
diff.matrix <- matrix(c("theta1.1","theta1.2", "1", "1"), 2, 2)

drift.c <- c("-theta2.1*x1", "-theta2.2*x2", "-theta2.2", "-theta2.1")
drift.matrix <- matrix(drift.c, 2, 2)

ymodel <- setModel(drift=drift.matrix, diffusion=diff.matrix, time.variable="t",
                   state.variable=c("x1", "x2"), solve.variable=c("x1", "x2"))
n <- 100
ysamp <- setSampling(Terminal=(n)^(1/3), n=n)
yuima <- setYuima(model=ymodel, sampling=ysamp)
set.seed(123)

truep <- list(theta1.1=0.6, theta1.2=0,theta2.1=0.5, theta2.2=0)
yuima <- simulate(yuima, xinit=c(1, 1),
 true.parameter=truep)


est <- lasso(yuima, start=list(theta2.1=0.8, theta2.2=0.2, theta1.1=0.7, theta1.2=0.1),
 lower=list(theta1.1=1e-10,theta1.2=1e-10,theta2.1=.1,theta2.2=1e-10),
 upper=list(theta1.1=4,theta1.2=4,theta2.1=4,theta2.2=4), method="L-BFGS-B")

# TRUE
unlist(truep)

# QMLE
round(est$mle,3)

# LASSO
round(est$lasso,3)
```

limiting.gamma *calculate the value of limiting covariance matrices : Gamma*

Description

To confirm assysmptotic normality of theta estimators.

Usage

```
limiting.gamma(obj,theta,verbose=FALSE)
```

Arguments

obj	an yuima or yuima.model object.
theta	true theta
verbose	an option for display a verbose process.

Details

Calculate the value of limiting covariance matrices Gamma. The returned values gamma1 and gamma2 are used to confirm assysmptotic normality of theta estimators. this program is limitted to 1-dimention-sde model for now.

Value

gamma1	a theoretical figure for variance of theta1 estimator
gamma2	a theoretical figure for variance of theta2 estimator

Note

we need to fix this routine.

Author(s)

The YUIMA Project Team

Examples

```
set.seed(123)

## Yuima
diff.matrix <- matrix(c("theta1"), 1, 1)
myModel <- setModel(drift=c("(-1)*theta2*x"), diffusion=diff.matrix,
time.variable="t", state.variable="x")
n <- 100
mySampling <- setSampling(Terminal=(n)^(1/3), n=n)
myYuima <- setYuima(model=myModel, sampling=mySampling)
```

```
myYuima <- simulate(myYuima, xinit=1, true.parameter=list(theta1=0.6, theta2=0.3))

## theorical figure of theta
theta1 <- 3.5
theta2 <- 1.3

theta <- list(theta1, theta2)
lim.gamma <- limiting.gamma(obj=myYuima, theta=theta, verbose=TRUE)

## return theta1 and theta2 with list
lim.gamma$list

## return theta1 and theta2 with vector
lim.gamma$vec
```

`llag` *Lead Lag Estimator*

Description

Estimate the lead-lag parameters of discretely observed processes by maximizing the shifted Hayashi-Yoshida covariation contrast functions, following Hoffmann et al. (2013).

Usage

```
llag(x,from=-Inf,to=Inf,division=FALSE,verbose=FALSE,grid,psd=TRUE,
     plot=FALSE,ccor=FALSE)
```

Arguments

`x`	an object of `yuima-class` or `yuima.data-class`.
`verbose`	whether `llag` returns matrices or not.
`from`	a numeric vector each of whose component(s) indicates the lower end of a finite grid on which the contrast function is evaluated, if `grid` is missing.
`to`	a numeric vector each of whose component(s) indicates the upper end of a finite grid on which the contrast function is evaluated, if `grid` is missing.
`division`	a numeric vector each of whose component(s) indicates the number of the points of a finite grid on which the contrast function is evaluated, if `grid` is missing.
`grid`	a numeric vector or a list of numeric vectors. See 'Details'.
`psd`	logical. If `TRUE`, the estimated cross-correlation functions are converted to the interval [-1,1]. See 'Details'.
`plot`	logical. If `TRUE`, the estimated cross-correlation functions are plotted.
`ccor`	logical. If `TRUE`, the estimated cross-correlation functions are returned. This argument is ignored if `verbose` is `FALSE`.

Details

Let d be the number of the components of the `zoo.data` of the object `x`.

Let $Xi_{ti_0}, Xi_{ti_1}, \ldots, Xi_{ti_{n(i)}}$ be the observation data of the i-th component (i.e. the i-th component of the `zoo.data` of the object `x`).

The shifted Hayashi-Yoshida covariation contrast function $Uij(\theta)$ of the observations Xi and Xj $(i < j)$ is defined by the same way as in Hoffmann et al. (2013), which corresponds to their cross-covariance function. The lead-lag parameter $\theta_i j$ is defined as a maximizer of $|Uij(\theta)|$. $Uij(\theta)$ is evaluated on a finite grid Gij defined below. Thus $\theta_i j$ belongs to this grid. If there exist more than two maximizers, the lowest one is selected.

If `psd` is `TRUE`, For any i, j the matrix $C := (Ukl(\theta))_{k,l \in i,j}$ is converted to `(C%*%C)^(1/2)` for ensuring the positive semi-definiteness, and $Uij(\theta)$ is redefined as the $(1,2)$-component of the converted C. Here, $Ukk(\theta)$ is set to the realized volatility of Xk. In this case $\theta_i j$ is given as a maximizer of the cross-correlation functions.

The grid Gij is defined as follows. First, if `grid` is missing, Gij is given by

$$a, a + (b-a)/(N-1), \ldots, a + (N-2)(b-a)/(N-1), b,$$

where a, b and N are the $(d(i-1) - (i-1)i/2 + (j-i))$-th components of `from`, `to` and `division` respectively. If the corresponding component of `from` (resp. `to`) is `-Inf` (resp. `Inf`), $a = -(tj_n(j) - ti_0)$ (resp. $b = ti_n(i) - tj_0$) is used, while if the corresponding component of `division` is `FALSE`, $N = round(2max(n(i), n(j))) + 1$ is used. Missing components are filled with `-Inf` (resp. `Inf`, `FALSE`). The default value `-Inf` (resp. `Inf`, `FALSE`) means that all components are `-Inf` (resp. `Inf`, `FALSE`). Next, if `grid` is a numeric vector, Gij is given by `grid`. If `grid` is a list of numeric vectors, Gij is given by the $(d(i-1) - (i-1)i/2 + (j-i))$-th component of `grid`.

The estimated lead-lag parameters are returned as the skew-symmetric matrix $(\theta_i j)_{i,j=1,\ldots,d}$. If `verbose` is `TRUE`, the covariance matrix $(Uij(\theta_i j))_{i,j=1,\ldots,d}$ corresponding to the estimated lead-lag parameters, the corresponding correlation matrix and the computed contrast functions are also returned. If further `ccor` is `TRUE`,the computed cross-correlation functions are returned as a list with the length $d(d-1)/2$. For $i < j$, the $(d(i-1) - (i-1)i/2 + (j-i))$-th component of the list consists of an object $Uij(\theta)/sqrt(Uii(\theta) * Ujj(\theta))$ of class zoo indexed by Gij.

If `plot` is `TRUE`, the computed cross-correlation functions are plotted sequentially.

Value

If `verbose` is `FALSE`, a skew-symmetric matrix corresponding to the estimated lead-lag parameters is returned. If `verbose` is `TRUE`, a list with the following components is returned:

`lagcce`	a skew-symmetric matrix corresponding to the estimated lead-lag parameters.
`covmat`	a covariance matrix corresponding to the estimated lead-lag parameters.
`cormat`	a correlation matrix corresponding to the estimated lead-lag parameters.

If further `ccor` is `TRUE`, the following component is added to the returned list:

`ccor`	a list of computed cross-correlation functions.

Author(s)

The YUIMA Project Team

References

Hoffmann, M., Rosenbaum, M. and Yoshida, N. (2013) Estimation of the lead-lag parameter from non-synchronous data, *Bernoulli*, **19**, no. 2, 426–461.

See Also

cce

Examples

```
## Set a model
diff.coef.matrix <- matrix(c("sqrt(x1)", "3/5*sqrt(x2)",
 "1/3*sqrt(x3)", "", "4/5*sqrt(x2)","2/3*sqrt(x3)",
 "","","2/3*sqrt(x3)"), 3, 3)
drift <- c("1-x1","2*(10-x2)","3*(4-x3)")
cor.mod <- setModel(drift = drift,
 diffusion = diff.coef.matrix,
 solve.variable = c("x1", "x2","x3"))

set.seed(111)

## We use a function poisson.random.sampling
# to get observation by Poisson sampling.
yuima.samp <- setSampling(Terminal = 1, n = 1200)
yuima <- setYuima(model = cor.mod, sampling = yuima.samp)
yuima <- simulate(yuima,xinit=c(1,7,5))

# intentionally displace the second time series

  data2 <- yuima@data@zoo.data[[2]]
  time2 <- time(data2)
  theta2 <- 0.05   # the lag of x2 behind x1
  stime2 <- time2 + theta2
  time(yuima@data@zoo.data[[2]]) <- stime2

  data3 <- yuima@data@zoo.data[[3]]
  time3 <- time(data3)
  theta3 <- 0.12   # the lag of x3 behind x1
  stime3 <- time3 + theta3
  time(yuima@data@zoo.data[[3]]) <- stime3



#sampled data by Poisson rules
psample<- poisson.random.sampling(yuima,
 rate = c(0.2,0.3,0.4), n = 1000)


# plot
plot(psample)
```

```
#cce
cce(psample)

#lead-lag estimation (with cross-correlation plots)
par(mfcol=c(3,1))
result <- llag(psample,verbose=TRUE,plot=TRUE)

#estimated lead-lag parameter
result$lagcce
```

`mmfrac` *mmfrac*

Description

Estimates the drift of a fractional Ornstein-Uhlenbeck and, if necessary, also the Hurst and diffusion parameters.

Usage

```
mmfrac(yuima, ...)
```

Arguments

`yuima`	a `yuima` object.
`...`	arguments passed to `qgv`.

Details

Estimates the drift of s fractional Ornstein-Uhlenbeck and, if necessary, also the Hurst and diffusion parameters.

Value

an object of class mmfrac

Author(s)

The YUIMA Project Team

References

Brouste, A., Iacus, S.M. (2013) Parameter estimation for the discretely observed fractional Ornstein-Uhlenbeck process and the Yuima R package, Computational Statistics, pp. 1129–1147.

See Also

See also qgv.

Examples

```
# Estimating all Hurst parameter, diffusion coefficient  and drift coefficient
# in fractional Ornstein-Uhlenbeck

model<-setModel(drift="-x*lambda",hurst=NA,diffusion="theta")
sampling<-setSampling(T=100,n=10000)
yui1<-simulate(model,true.param=list(theta=1,lambda=4),hurst=0.7,sampling=sampling)
mmfrac(yui1)
```

---

model.parameter-class *Class for the parameter description of stochastic differential equations*

Description

The model.parameter-class is a class of the **yuima** package.

Details

The model.parameter-class object cannot be directly specified by the user but it is constructed when the yuima.model-class object is constructed via setModel. All the terms which are not in the list of *solution, state, time, jump* variables are considered as parameters. These parameters are identified in the different components of the model (drift, diffusion and jump part). This information is later used to draw inference jointly or separately for the different parameters depending on the model in hands.

Slots

drift: A vector of names belonging to the drift coefficient.

diffusion: A vector of names of parameters belonging to the diffusion coefficient.

jump: A vector of names of parameters belonging to the jump coefficient.

measure: A vector of names of parameters belonging to the Levy measure.

xinit: A vector of names of parameters belonging to the initial condition.

all: A vector of names of all the parameters found in the components of the model.

common: A vector of names of the parameters in common among drift, diffusion, jump and measure term.

Author(s)

The YUIMA Project Team

mpv *Realized Multipower Variation*

Description

The function returns the realized MultiPower Variation (mpv), defined in Barndorff-Nielsen and Shephard (2004), for each components.

Usage

```
mpv(yuima, r = 2, normalize = TRUE)
```

Arguments

yuima	an object of yuima-class or yuima.data-class.
r	a vector of non-negative numbers or a list of vectors of non-negative numbers.
normalize	logical. See 'Details'.

Details

Let d be the number of the components of the zoo.data of yuima.

Let $X^i_{t_0}, X^i_{t_1}, \ldots, X^i_{t_n}$ be the observation data of the i-th component (i.e. the i-th component of the zoo.data of yuima).

When r is a k-dimensional vector of non-negative numbers, mpv(yuima,r,normalize=TRUE) is defined as the d-dimensional vector with i-th element equal to

$$\mu_{r[1]}^{-1} \cdots \mu_{r[k]}^{-1} n^{\frac{r[1]+\cdots+r[k]}{2}-1} \sum_{j=1}^{n-k+1} |\Delta X^i_{t_j}|^{r[1]} |\Delta X^i_{t_{j+1}}|^{r[2]} \cdots |\Delta X^i_{t_{j+k-1}}|^{r[k]},$$

where μ_p is the p-th absolute moment of the standard normal distribution and $\Delta X^i_{t_j} = X^i_{t_j} - X^i_{t_{j-1}}$. If normalize is FALSE the result is not multiplied by $\mu_{r[1]}^{-1} \cdots \mu_{r[k]}^{-1}$.

When r is a list of vectors of non-negative numbers, mpv(yuima,r,normalize=FALSE) is defined as the d-dimensional vector with i-th element equal to

$$\mu_{r_1^i}^{-1} \cdots \mu_{r_{k_i}^i}^{-1} n^{\frac{r_1^i+\cdots+r_{k_i}^i}{2}-1} \sum_{j=1}^{n-k_i+1} |\Delta X^i_{t_j}|^{r_1^i} |\Delta X^i_{t_{j+1}}|^{r_2^i} \cdots |\Delta X^i_{t_{j+k_i-1}}|^{r_{k_i}^i},$$

where $r_1^i, \ldots, r_{k_i}^i$ is the i-th component of r. If normalize is FALSE the result is not multiplied by $\mu_{r_1^i}^{-1} \cdots \mu_{r_{k_i}^i}^{-1}$.

Value

A numeric vector with the same length as the zoo.data of yuima

Author(s)

The YUIMA Project Team

References

Barndorff-Nielsen, O. E. and Shephard, N. (2004) Power and bipower variation with stochastic volatility and jumps, *Journal of Financial Econometrics*, **2**, no. 1, 1–37.

Barndorff-Nielsen, O. E. , Graversen, S. E. , Jacod, J. , Podolskij M. and Shephard, N. (2006) A central limit theorem for realised power and bipower variations of continuous semimartingales, in: Kabanov, Y. , Lipster, R. , Stoyanov J. (Eds.), From Stochastic Calculus to Mathematical Finance: The Shiryaev Festschrift, Springer-Verlag, Berlin, pp. 33–68.

See Also

setModel,cce

Examples

```
set.seed(123)

# One-dimensional case
## Model: dXt=t*dWt+t*dzt,
## where zt is a compound Poisson process with intensity 5 and jump sizes distribution N(0,0.1).

model <- setModel(drift=0,diffusion="t",jump.coeff="t",measure.type="CP",
                  measure=list(intensity=5,df=list("dnorm(z,0,sqrt(0.1))")),
                  time.variable="t")

yuima.samp <- setSampling(Terminal = 1, n = 390)
yuima <- setYuima(model = model, sampling = yuima.samp)
yuima <- simulate(yuima)
plot(yuima)

mpv(yuima) # true value is 1/3
mpv(yuima,1) # true value is 1/2
mpv(yuima,rep(2/3,3)) # true value is 1/3

# Multi-dimensional case
## Model: dXkt=t*dWk_t (k=1,2,3).

diff.matrix <- diag(3)
diag(diff.matrix) <- c("t","t","t")
model <- setModel(drift=c(0,0,0),diffusion=diff.matrix,time.variable="t",
                  solve.variable=c("x1","x2","x3"))

yuima.samp <- setSampling(Terminal = 1, n = 390)
yuima <- setYuima(model = model, sampling = yuima.samp)
yuima <- simulate(yuima)
plot(yuima)
```

```
mpv(yuima,list(c(1,1),1,rep(2/3,3))) # true varue is c(1/3,1/2,1/3)
```

MWK151 *Graybill - Methuselah Walk - PILO - ITRDB CA535*

Description

Graybill - Methuselah Walk - PILO - ITRDB CA535, pine tree width in mm from -608 to 1957.

Usage

```
data(MWK151)
```

Details

The full data records of past temperature, precipitation, and climate and environmental change derived from tree ring measurements. Parameter keywords describe what was measured in this data set. Additional summary information can be found in the abstracts of papers listed in the data set citations, however many of the data sets arise from unpublished research contributed to the International Tree Ring Data Bank. Additional information on data processing and analysis for International Tree Ring Data Bank (ITRDB) data sets can be found on the Tree Ring Page http://www.ncdc.noaa.gov/paleo/treering.html.

The MWK151 is only a small part of the data relative to one tree and contains measurement of a tree's ring width in mm, from -608 to 1957.

Source

http://hurricane.ncdc.noaa.gov/pls/paleox/f?p=519:1:::::P1_STUDY_ID:3376

References

Graybill, D.A., and Shiyatov, S.G., Dendroclimatic evidence from the northern Soviet Union, in Climate since A.D. 1500, edited by R.S. Bradley and P.D. Jones, Routledge, London, 393-414, 1992.

Examples

```
data(MWK151)
```

noisy.sampling *Noisy Observation Generator*

Description

Generates a new observation data contaminated by noise.

Usage

```
noisy.sampling(x, var.adj = 0, rng = "rnorm", mean.adj = 0, ...,
               end.coef = 0, n, order.adj = 0, znoise)
```

Arguments

x	an object of yuima-class or yuima.data-class.
var.adj	a matrix or list to be used for adjusting the variance matrix of the exogenous noise.
rng	a function to be used for generating the random numbers for the exogenous noise.
mean.adj	a numeric vector to be used for adjusting the mean vector of the exogenous noise.
...	passed to rng.
end.coef	a numeric vector or list to be used for adjusting the variance of the endogenous noise.
n	a numeric vector to be used for adjusting the scale of the endogenous noise.
order.adj	a positive number to be used for adjusting the order of the noise.
znoise	a list indicating other sources of noise processes. The default value is as.list(double(dim(x))).

Details

TBA

Value

an object of yuima.data-class.

Author(s)

The YUIMA Project Team

See Also

cce

Examples

```
## Set a model (a two-dimensional normal model sampled by a Poisson random sampling)
set.seed(123)

drift <- c(0,0)

sigma1 <- 1
sigma2 <- 1
rho <- 0.7

diffusion <- matrix(c(sigma1,sigma2*rho,0,sigma2*sqrt(1-rho^2)),2,2)

model <- setModel(drift=drift,diffusion=diffusion,
                  state.variable=c("x1","x2"),solve.variable=c("x1","x2"))

yuima.samp <- setSampling(Terminal = 1, n = 2340)
yuima <- setYuima(model = model, sampling = yuima.samp)
yuima <- simulate(yuima)

## Poisson random sampling
psample<- poisson.random.sampling(yuima, rate = c(1/3,1/6), n = 2340)

## Plot the path without noise
plot(psample)

# Set a matrix as the variance of noise
Omega <- 0.01*diffusion

## Contaminate the observation data by centered normal distributed noise
## with the variance matrix equal to 1% of the diffusion
noisy.psample1 <- noisy.sampling(psample,var.adj=Omega)
plot(noisy.psample1)

## Contaminate the observation data by centered uniformly distributed noise
## with the variance matrix equal to 1% of the diffusion
noisy.psample2 <- noisy.sampling(psample,var.adj=Omega,rng="runif",min=-sqrt(3),max=sqrt(3))
plot(noisy.psample2)

## Contaminate the observation data by centered exponentially distributed noise
## with the variance matrix equal to 1% of the diffusion
noisy.psample3 <- noisy.sampling(psample,var.adj=Omega,rng="rexp",rate=1,mean.adj=1)
plot(noisy.psample3)

## Contaminate the observation data by its return series
## multiplied by -0.1 times the square root of the intensity vector
## of the Poisson random sampling
noisy.psample4 <- noisy.sampling(psample,end.coef=-0.1,n=2340*c(1/3,1/6))
plot(noisy.psample4)

## An application:
## Adding a compound Poisson jumps to the observation data
```

```
## Set a compound Poisson process
intensity <- 5
j.num <- rpois(1,intensity) # Set a number of jumps
j.idx <- unique(ceiling(2340*runif(j.num))) # Set time indices of jumps
jump <- matrix(0,2,2341)
jump[,j.idx+1] <- sqrt(0.25/intensity)*diffusion
grid <- seq(0,1,by=1/2340)
CPprocess <- list(zoo(cumsum(jump[1,]),grid),zoo(cumsum(jump[2,]),grid))

## Adding the jumps
yuima.jump <- noisy.sampling(yuima,znoise=CPprocess)
plot(yuima.jump)

## Poisson random sampling
psample.jump <- poisson.random.sampling(yuima.jump, rate = c(1/3,1/6), n = 2340)
plot(psample.jump)
```

`phi.test` *Phi-divergence test statistic for stochastic differential equations*

Description

Phi-divergence test statistic for stochastic differential equations.

Usage

```
phi.test(yuima, H0, H1, phi, print=FALSE,...)
```

Arguments

`yuima`	a yuima object.
`H0`	a named list of parameter under H0.
`H1`	a named list of parameter under H1.
`phi`	the phi function to be used in the test. See Details.
`print`	you can see a progress of the estimation when print is `TRUE`.
`...`	passed to `qmle` function.

Details

`phi.test` executes a Phi-divergence test. If `H1` is not specified this hypothesis is filled with the QMLE estimates.

If `phi` is missing, then `phi(x)=1-x+x*log(x)` and the Phi-divergence statistic corresponds to the likelihood ratio test statistic.

Value

`ans`	an obkect of class `phitest`.

Author(s)

The YUIMA Project Team

Examples

```
model<- setModel(drift="t1*(t2-x)",diffusion="t3")
T<-10
n<-1000
sampling <- setSampling(Terminal=T,n=n)
yuima<-setYuima(model=model, sampling=sampling)

h0 <- list(t1=0.3, t2=1, t3=0.25)
X <- simulate(yuima, xinit=1, true=h0)
h1 <- list(t1=0.3, t2=0.2, t3=0.1)

phi1 <- function(x) 1-x+x*log(x)

phi.test(X, H0=h0, H1=h1,phi=phi1)
phi.test(X, H0=h0, phi=phi1, start=h0, lower=list(t1=0.1, t2=0.1, t3=0.1),
   upper=list(t1=2,t2=2,t3=2),method="L-BFGS-B")
phi.test(X, H0=h1, phi=phi1, start=h0, lower=list(t1=0.1, t2=0.1, t3=0.1),
  upper=list(t1=2,t2=2,t3=2),method="L-BFGS-B")
```

---

poisson.random.sampling

Poisson random sampling method

Description

Poisson random sampling method.

Usage

```
poisson.random.sampling(x, rate, n)
```

Arguments

x	an object of yuima.data-class or yuima-class.
rate	a Poisson intensity or a vector of Poisson intensities.
n	a common multiplier to the Poisson intensities. The default value is 1.

Details

It returns an object of type yuima.data-class which is a copy of the original input data where observations are sampled according to the Poisson process. The unsampled data are set to NA.

Value

an object of yuima.data-class.

Author(s)

The YUIMA Project Team

See Also

cce

Examples

```
## Set a model
diff.coef.1 <- function(t, x1=0, x2) x2*(1+t)
diff.coef.2 <- function(t, x1, x2=0) x1*sqrt(1+t^2)
cor.rho <- function(t, x1=0, x2=0) sqrt((1+cos(x1*x2))/2)
diff.coef.matrix <- matrix(c("diff.coef.1(t,x1,x2)",
"diff.coef.2(t,x1,x2)*cor.rho(t,x1,x2)", "",
"diff.coef.2(t,x1,x2)*sqrt(1-cor.rho(t,x1,x2)^2)"),2,2)
cor.mod <- setModel(drift=c("","",""), diffusion=diff.coef.matrix,
solve.variable=c("x1", "x2"), xinit=c(3,2))
set.seed(111)

## We first simulate the two dimensional diffusion model
yuima.samp <- setSampling(Terminal=1, n=1200)
yuima <- setYuima(model=cor.mod, sampling=yuima.samp)
yuima.sim <- simulate(yuima)

## Then we use function poisson.random.sampling to get observations
## by Poisson sampling.
psample <- poisson.random.sampling(yuima.sim,  rate = c(0.2, 0.3), n=1000)
str(psample)
```

---

qgv *qgv*

Description

Estimate the local Holder exponent with quadratic generalized variations method

Usage

```
qgv(yuima, filter.type = "Daubechies", order = 2, a = NULL)
```

Arguments

yuima	A yuima object.
filter.type	The filter.type can be set to "Daubechies" or "Classical".
order	The order of the filter a to be chosen
a	Any other filter

Details

Estimation of the Hurst index and the constant of the fractional Ornstein-Uhlenbeck process.

Value

an object of class qgv

Author(s)

The YUIMA Project Team

References

Brouste, A., Iacus, S.M. (2013) Parameter estimation for the discretely observed fractional Ornstein-Uhlenbeck process and the Yuima R package, Computational Statistics, pp. 1129–1147.

See Also

See also mmfrac.

Examples

```
# Estimating both Hurst parameter and diffusion coefficient in fractional Ornstein-Uhlenbeck

model<-setModel(drift="-x*lambda",hurst=NA,diffusion="theta")
sampling<-setSampling(T=100,n=10000)
yui1<-simulate(model,true.param=list(theta=1,lambda=4),hurst=0.7,sampling=sampling)
qgv(yui1)


# Estimating Hurst parameter only in diffusion processes

model2<-setModel(drift="-x*lambda",hurst=NA,diffusion="theta*sqrt(x)")
sampling<-setSampling(T=1,n=10000)
yui2<-simulate(model2,true.param=list(theta=1,lambda=4),hurst=0.7,sampling=sampling,xinit=10)
qgv(yui2)
```

---

qmle *Calculate quasi-likelihood and ML estimator of least squares estimator*

Description

Calculate the quasi-likelihood and estimate of the parameters of the stochastic differential equation by the maximum likelihood method or least squares estimator of the drift parameter.

Usage

```
qmle(yuima, start, method="BFGS", fixed = list(), print=FALSE, lower, upper,
 joint=FALSE, Est.Incr="Carma.IncPar",aggregation=TRUE, threshold=NULL, ...)
quasilogl(yuima, param, print=FALSE)
lse(yuima, start, lower, upper, method = "BFGS", ...)
```

Arguments

yuima	a yuima object.
print	you can see a progress of the estimation when print is TRUE.
method	see Details.
param	list of parameters for the quasi loglikelihood.
lower	a named list for specifying lower bounds of parameters
upper	a named list for specifying upper bounds of parameters
start	initial values to be passed to the optimizer.
fixed	for conditional (quasi)maximum likelihood estimation.
joint	perform joint estimation or two stage estimation? by default joint=FALSE.
Est.Incr	If the yuima model is an object of yuima.carma-class the qmle returns an object of yuima.carma.qmle-class or object of class mle-class. By default Est.Incr="Carma.IncPar".
aggregation	If aggregation=TRUE, before the estimation of the levy parameters we aggregate the increments.
threshold	If the model has Compund Poisson type jumps, the threshold is used to perform thresholding of the increments.
...	passed to optim method. See Examples.

Details

qmle behaves more likely the standard mle function in **stats4** and argument method is one of the methods available in optim.

lse calculates least squares estimators of the drift parameters. This is useful for initial guess of qmle estimation.

quasilogl returns the valueof the quasi loglikelihood for a given yuima object and list of parameters coef.

Value

QL	a real value.
opt	a list with components the same as 'optim' function.
carmaopt	if the model is an object of yuima.carma-class, qmle returns an object yuima.carma.qmle-class

Note

The function qmle uses the function optim internally.

The function qmle uses the function CarmaNoise internally for estimation of underlying Levy if the model is an object of yuima.carma-class.

Author(s)

The YUIMA Project Team

Examples

```
#dXt^e = -theta2 * Xt^e * dt + theta1 * dWt
diff.matrix <- matrix(c("theta1"), 1, 1)
ymodel <- setModel(drift=c("(-1)*theta2*x"), diffusion=diff.matrix,
  time.variable="t", state.variable="x", solve.variable="x")
n <- 100

ysamp <- setSampling(Terminal=(n)^(1/3), n=n)
yuima <- setYuima(model=ymodel, sampling=ysamp)
set.seed(123)
yuima <- simulate(yuima, xinit=1, true.parameter=list(theta1=0.3,
theta2=0.1))
QL <- quasilogl(yuima, param=list(theta2=0.8, theta1=0.7))
##QL <- ql(yuima, 0.8, 0.7, h=1/((n)^(2/3)))
QL

## another way of parameter specification
##param <- list(theta2=0.8, theta1=0.7)
##QL <- ql(yuima, h=1/((n)^(2/3)), param=param)
##QL


## old code
##system.time(
##opt <- ml.ql(yuima, 0.8, 0.7, h=1/((n)^(2/3)), c(0, 1), c(0, 1))
##)
##cat(sprintf("\nTrue param. theta2 = .3, theta1 = .1\n"))
##print(coef(opt))


system.time(
opt2 <- qmle(yuima, start=list(theta1=0.8, theta2=0.7), lower=list(theta1=0,theta2=0),
 upper=list(theta1=1,theta2=1), method="L-BFGS-B")
)
cat(sprintf("\nTrue param. theta2 = .3, theta1 = .1\n"))
print(coef(opt2))

## initial guess for theta2 by least squares estimator
tmp <- lse(yuima, start=list(theta2=0.7), lower=list(theta2=0), upper=list(theta2=1))
tmp
```

```
system.time(
opt3 <- qmle(yuima, start=list(theta1=0.8, theta2=tmp), lower=list(theta1=0,theta2=0),
 upper=list(theta1=1,theta2=1), method="L-BFGS-B")
)
cat(sprintf("\nTrue param. theta2 = .3, theta1 = .1\n"))
print(coef(opt3))


## perform joint estimation? Non-optimal, just for didactic purposes
system.time(
opt4 <- qmle(yuima, start=list(theta1=0.8, theta2=0.7), lower=list(theta1=0,theta2=0),
 upper=list(theta1=1,theta2=1), method="L-BFGS-B", joint=TRUE)
)
cat(sprintf("\nTrue param. theta2 = .3, theta1 = .1\n"))
print(coef(opt4))

## old code
##system.time(
##opt <- ml.ql(yuima, 0.8, 0.7, h=1/((n)^(2/3)), c(0, 1), c(0, 1), method="Newton")
##)
##cat(sprintf("\nTrue param. theta2 = .3, theta1 = .1\n"))
##print(coef(opt))


## Not run:

###multidimension case
##dXt^e = - drift.matrix * Xt^e * dt + diff.matrix * dWt
diff.matrix <- matrix(c("theta1.1","theta1.2", "1", "1"), 2, 2)

drift.c <- c("-theta2.1*x1", "-theta2.2*x2", "-theta2.2", "-theta2.1")
drift.matrix <- matrix(drift.c, 2, 2)

ymodel <- setModel(drift=drift.matrix, diffusion=diff.matrix, time.variable="t",
                   state.variable=c("x1", "x2"), solve.variable=c("x1", "x2"))
n <- 100
ysamp <- setSampling(Terminal=(n)^(1/3), n=n)
yuima <- setYuima(model=ymodel, sampling=ysamp)
set.seed(123)

##xinit=c(x1, x2) #true.parameter=c(theta2.1, theta2.2, theta1.1, theta1.2)
yuima <- simulate(yuima, xinit=c(1, 1),
true.parameter=list(theta2.1=0.5, theta2.2=0.3, theta1.1=0.6, theta1.2=0.2))

## theta2 <- c(0.8, 0.2) #c(theta2.1, theta2.2)
##theta1 <- c(0.7, 0.1) #c(theta1.1, theta1.2)
## QL <- ql(yuima, theta2, theta1, h=1/((n)^(2/3)))
## QL

## ## another way of parameter specification
## #param <- list(theta2=theta2, theta1=theta1)
## #QL <- ql(yuima, h=1/((n)^(2/3)), param=param)
## #QL
```

```
## theta2.1.lim <- c(0, 1)
## theta2.2.lim <- c(0, 1)
## theta1.1.lim <- c(0, 1)
## theta1.2.lim <- c(0, 1)
## theta2.lim <- t( matrix( c(theta2.1.lim, theta2.2.lim), 2, 2) )
## theta1.lim <- t( matrix( c(theta1.1.lim, theta1.2.lim), 2, 2) )

## system.time(
## opt <- ml.ql(yuima, theta2, theta1, h=1/((n)^(2/3)), theta2.lim, theta1.lim)
## )
## opt@coef

system.time(
opt2 <- qmle(yuima, start=list(theta2.1=0.8, theta2.2=0.2, theta1.1=0.7, theta1.2=0.1),
 lower=list(theta1.1=.1,theta1.2=.1,theta2.1=.1,theta2.2=.1),
 upper=list(theta1.1=4,theta1.2=4,theta2.1=4,theta2.2=4), method="L-BFGS-B")
)
opt2@coef
summary(opt2)

## unconstrained optimization
system.time(
opt3 <- qmle(yuima, start=list(theta2.1=0.8, theta2.2=0.2, theta1.1=0.7, theta1.2=0.1))
)
opt3@coef
summary(opt3)


quasilogl(yuima, param=list(theta2.1=0.8, theta2.2=0.2, theta1.1=0.7, theta1.2=0.1))

##system.time(
##opt <- ml.ql(yuima, theta2, theta1, h=1/((n)^(2/3)), theta2.lim, theta1.lim, method="Newton")
##)
##opt@coef
##

# carma(p=2,q=0) driven by a brownian motion without location parameter


mod0<-setCarma(p=2,
               q=0,
               scale.par="sigma")

true.parm0 <-list(a1=1.39631,
                  a2=0.05029,
                  b0=1,
                  sigma=0.23)

samp0<-setSampling(Terminal=100,n=250)
set.seed(123)
sim0<-simulate(mod0,
               true.parameter=true.parm0,
```

```
                sampling=samp0)

system.time(
carmaopt0 <- qmle(sim0, start=list(a1=1.39631,a2=0.05029,
                                b0=1,
                                 sigma=0.23))
)

summary(carmaopt0)




# carma(p=2,q=1) driven by a brownian motion without location parameter

mod1<-setCarma(p=2,
                q=1)

true.parm1 <-list(a1=1.39631,
                  a2=0.05029,
                  b0=1,
                  b1=2)

samp1<-setSampling(Terminal=100,n=250)
set.seed(123)
sim1<-simulate(mod1,
                true.parameter=true.parm1,
                sampling=samp1)

system.time(
  carmaopt1 <- qmle(sim1, start=list(a1=1.39631,a2=0.05029,
                                     b0=1,b1=2),joint=TRUE)
)

summary(carmaopt1)

plot(carmaopt1)

# carma(p=2,q=1) driven by a compound poisson process where the jump size is normally distributed.

mod2<-setCarma(p=2,
                q=1,
                measure=list(intensity="1",df=list("dnorm(z, 0, 1)")),
                measure.type="CP")

true.parm2 <-list(a1=1.39631,
                  a2=0.05029,
                  b0=1,
                  b1=2)

samp2<-setSampling(Terminal=100,n=250)
set.seed(123)
```

```
sim2<-simulate(mod2,
               true.parameter=true.parm2,
               sampling=samp2)

system.time(
  carmaopt2 <- qmle(sim2, start=list(a1=1.39631,a2=0.05029,
                                     b0=1,b1=2),joint=TRUE)
)

summary(carmaopt2)

plot(carmaopt2)

# carma(p=2,q=1) driven by a normal inverse gaussian process
mod3<-setCarma(p=2,q=1,
               measure=list(df=list("rNIG(z, alpha, beta, delta1, mu)")),
               measure.type="code")
#

# True param
true.param3<-list(a1=1.39631,
                  a2=0.05029,
                  b0=1,
                  b1=2,
                  alpha=1,
                  beta=0,
                  delta1=1,
                  mu=0)

samp3<-setSampling(Terminal=100,n=200)
set.seed(123)

sim3<-simulate(mod3,
               true.parameter=true.param3,
               sampling=samp3)


carmaopt3<-qmle(sim3,start=true.param3)

summary(carmaopt3)

plot(carmaopt3)


## End(Not run)
```

`rconst` *Fictitious rng for the constant random variable used to generate and describe Poisson jumps.*

Description

Fictitious rng for the constant random variable used to generate and describe Poisson jumps.

Usage

```
rconst(n, k = 1)
dconst(x, k = 1)
```

Arguments

n	number of replications
k	the size of the jump
x	the fictitious argument

Value

returns a numeric vector

Author(s)

The YUIMA Project Team

Examples

```
dconst(1,1)
dconst(2,1)
dconst(2,2)

rconst(10,3)
```

---

rng *Random Number Generators for yuima package*

Description

`simulate` function uses specific random number generators to generate Levy paths.

Usage

```
rIG(x,delta,gamma)
rNIG(x,alpha=1,beta=0,delta=1,mu=0,Lambda)
rbgamma(x,delta.plus,gamma.plus,delta.minus,gamma.minus)
rngamma(x,lambda,alpha,beta,mu,Lambda)
rstable(x,alpha,beta,sigma,gamma)
```

Arguments

x	Number of R.Ns to be geneated.
delta	param
delta.plus	param
delta.minus	param
gamma	param
gamma.plus	param
gamma.minus	param
alpha	param
beta	parame
mu	param
lambda	param
Lambda	param
sigma	param

Details

to be written

Value

rIG	A vector of random numbers.
rNIG	A vector of random numbers.
rbgamma	A vector of random numbers.

Note

Random number generators may be rewritten with C language to speed up.

Author(s)

The YUIMA Project Team

setCarma *Continuous Autoregressive Moving Average (p, q) model*

Description

'setCarma' describes the following model:

```
Vt = c0 + sigma (b0 Xt(0) + ... + b(q) Xt(q))

dXt(0) = Xt(1) dt

...

dXt(p-2) = Xt(p-1) dt

dXt(p-1) = (-a(p) Xt(0) - ... - a(1) Xt(p-1))dt + (gamma(0) + gamma(1) Xt(0) + ... + gamma(p) Xt(p-1))c
```

The continuous ARMA process using the state-space representation as in Brockwell (2000) is obtained by choosing:

```
gamma(0) = 1, gamma(1) = gamma(2) = ... = gamma(p) = 0.
```

Please refer to the vignettes and the examples or the **yuima** documentation for details.

Usage

```
setCarma(p,q,loc.par=NULL,scale.par=NULL,ar.par="a",ma.par="b",
lin.par=NULL,Carma.var="v",Latent.var="x",XinExpr=FALSE, ...)
```

Arguments

p	a non-negative integer that indicates the number of the autoregressive coefficients.
q	a non-negative integer that indicates the number of the moving average coefficients.
loc.par	location coefficient. The default value loc.par=NULL implies that c0=0.
scale.par	scale coefficient. The default value scale.par=NULL implies that sigma=1.
ar.par	a character-string that is the label of the autoregressive coefficients. The default Value is ar.par="a".
ma.par	a character-string that is the label of the moving average coefficients. The default Value is ma.par="b".
Carma.var	a character-string that is the label of the observed process. Defaults to "v".
Latent.var	a character-string that is the label of the unobserved process. Defaults to "x".
lin.par	a character-string that is the label of the linear coefficients. If lin.par=NULL, the default, the 'setCarma' builds the CARMA(p, q) model defined as in Brockwell (2000).
XinExpr	a logical variable. The default value XinExpr=FALSE implies that the starting condition for Latent.var is zero. If XinExpr=TRUE, each component of Latent.var has a parameter as a initial value.

`...` Arguments to be passed to 'setCarma', such as the slots of `yuima.model-class`

`measure` Levy measure of jump variables.

`measure.type` type specification for Levy measure.

`xinit` a vector of `expressions` identyfying the starting conditions for CARMA model.

Details

Please refer to the vignettes and the examples or to the **yuimadocs** package.

An object of `yuima.carma-class` contains:

`info`: It is an object of `carma.info-class` which is a list of arguments that identifies the carma(p,q) model

and the same slots in an object of `yuima.model-class` .

Value

`model` an object of `yuima.carma-class`.

Note

There may be missing information in the model description. Please contribute with suggestions and fixings.

Author(s)

The YUIMA Project Team

References

Brockwell, P. (2000) Continuous-time ARMA processes, *Stochastic Processes: Theory and Methods. Handbook of Statistics*, **19**, (C. R. Rao and D. N. Shandhag, eds.) 249-276. North-Holland, Amsterdam.

Examples

```
# Ex 1. (Continuous ARMA process driven by a Brownian Motion)
# To describe the state-space representation of a CARMA(p=3,q=1) model:
# Vt=c0+alpha0*X0t+alpha1*X1t
# dX0t = X1t*dt
# dX1t = X2t*dt
# dX2t = (-beta3*X0t-beta2*X1t-beta1*X2t)dt+dWt
# we set
mod1<-setCarma(p=3,
               q=1,
               loc.par="c0")
# Look at the model structure by
str(mod1)
```

```
# Ex 2. (General setCarma model driven by a Brownian Motion)
# To describe the model defined as:
# Vt=c0+alpha0*X0t+alpha1*X1t
# dX0t = X1t*dt
# dX1t = X2t*dt
# dX2t = (-beta3*X0t-beta2*X1t-beta1*X2t)dt+(c0+alpha0*X0t)dWt
# we set
mod2 <- setCarma(p=3,
                 q=1,
                 loc.par="c0",
                 ma.par="alpha",
                 ar.par="beta",
                 lin.par="alpha")
# Look at the model structure by
str(mod2)

# Ex 3. (Continuous Arma model driven by a Levy process)
# To specify the CARMA(p=3,q=1) model driven by a Compound Poisson process defined as:
# Vt=c0+alpha0*X0t+alpha1*X1t
# dX0t = X1t*dt
# dX1t = X2t*dt
# dX2t = (-beta3*X0t-beta2*X1t-beta1*X2t)dt+dzt
# we set the Levy measure as in setModel
mod3 <- setCarma(p=3,
                 q=1,
                 loc.par="c0",
                 measure=list(intensity="1",df=list("dnorm(z, 0, 1)")),
                 measure.type="CP")
# Look at the model structure by
str(mod3)

# Ex 4. (General setCarma  model driven by a Levy process)
# Vt=c0+alpha0*X0t+alpha1*X1t
# dX0t = X1t*dt
# dX1t = X2t*dt
# dX2t = (-beta3*X1t-beta2*X2t-beta1*X3t)dt+(c0+alpha0*X0t)dzt
mod4 <- setCarma(p=3,
                 q=1,
                 loc.par="c0",
                 ma.par="alpha",
                 ar.par="beta",
                 lin.par="alpha",
                 measure=list(intensity="1",df=list("dnorm(z, 0, 1)")),
                 measure.type="CP")
# Look at the model structure by
str(mod4)
```

---

setCharacteristic *Set characteristic information and create a 'characteristic' object.*

Description

setCharacteristic is a constructor for characteristic class.

Usage

```
setCharacteristic(equation.number,time.scale)
```

Arguments

equation.number	The number of equations modeled in yuima object.
time.scale	time.scale assumed in the model.

Details

class characteristic has two slots, equation.number is the number of equations handled in the yuima object, and time.scale is a hoge of characteristic.

Value

An object of class characteristic.

Author(s)

The YUIMA Project Team

---

setData *Set and access data of an object of type "yuima.data" or "yuima".*

Description

setData constructs an object of yuima.data-class.

get.zoo.data returns the content of the zoo.data slot of a yuima.data-class object. (Note: value is a list of zoo objects).

plot plot method for object of yuima.data-class or yuima-class.

dim returns the dim of the zoo.data slot of a yuima.data-class object.

length returns the length of the time series in zoo.data slot of a yuima.data-class object. cbind.yuima bind yuima.data object.

Usage

```
setData(original.data, delta=NULL)
get.zoo.data(x)
```

Arguments

original.data	some type of data, usually some sort of time series. The function always tries to convert to the input data into an object of zoo-type. See Details.
x	an object of type yuima.data-class or yuima-class.
delta	If there is the need to redefine on the fly the delta increment of the data to make it consistent to statistical theory. See Details.

Details

Objects in the yuima.data-class contain two slots:

original.data**:** The slot original.data contains, as the name suggests, a copy of the original data passed to the function setData. It is intended for backup purposes.

zoo.data**:** the function setData tries to convert original.data into an object of class zoo. The coerced zoo data are stored in the slot zoo.data. If the conversion fails the function exits with an error. Internally, the **yuima** package stores and operates on zoo-type objects.

The function get.zoo.data returns the content of the slot zoo.data of x if x is of yuima.data-class or the content of x@data@zoo.data if x is of yuima-class.

Value

value	a list of object(s) of yuima.data-class for setData. The content of the zoo.data slot for get.zoo.data

Author(s)

The YUIMA Project Team

Examples

```
X <- ts(matrix(rnorm(200),100,2))
mydata <- setData(X)
str(get.zoo.data(mydata))
dim(mydata)
length(mydata)
plot(mydata)

# exactly the same output
mysde <- setYuima(data=setData(X))
str(get.zoo.data(mysde))
plot(mysde)
dim(mysde)
length(mysde)

# changing delta on the fly to 1/252
mysde2 <- setYuima(data=setData(X, delta=1/252))
str(get.zoo.data(mysde2))
plot(mysde2)
dim(mysde2)
```

```
length(mysde2)
```

`setFunctional` *Description of a functional associated with a perturbed stochastic differential equation*

Description

This function is used to give a description of the stochastic differential equation. The functional represent the price of the option in financial economics, for example.

Usage

```
setFunctional(model, F, f, xinit,e)
```

Arguments

`model`	yuima or yuima.model object.
`F`	function of X_t and $epsilon$
`f`	list of functions of X_t and $epsilon$
`xinit`	initial values of state variable.
`e`	epsilon parameter

Details

You should look at the vignette and examples.

The object foi contains several "slots". To see inside its structure we use the R command str. f and Fare R (list of) expressions which contains the functional of interest specification. e is a small parameter on which we conduct asymptotic expansion of the functional.

Value

`yuima`	an object of class 'yuima' containing object of class 'functional'. If yuima object was given as 'model' argument, the result is just added and the other slots of the object are maintained.

Note

There may be missing information in the model description. Please contribute with suggestions and fixings.

Author(s)

The YUIMA Project Team

Examples

```
set.seed(123)
# to the Black-Scholes economy:
# dXt^e = Xt^e * dt + e * Xt^e * dWt
diff.matrix <- matrix( c("x*e"), 1,1)
model <- setModel(drift = c("x"), diffusion = diff.matrix)
# call option is evaluated by averating
# max{ (1/T)*int_0^T Xt^e dt, 0}, the first argument is the functional of interest:
Terminal <- 1
xinit <- c(1)
f <- list( c(expression(x/Terminal)), c(expression(0)))
F <- 0
division <- 1000
e <- .3
yuima <- setYuima(model = model,sampling = setSampling(Terminal = Terminal, n = division))
yuima <- setFunctional( model = yuima, xinit=xinit, f=f,F=F,e=e)
# look at the model structure
str(yuima@functional)
```

---

setModel *Basic description of stochastic differential equations (SDE)*

Description

'setModel' gives a description of stochastic differential equation with or without jumps of the following form:

`dXt = a(t,Xt, alpha)dt + b(t,Xt,beta)dWt + c(t,Xt,gamma)dZt,  X0=x0`

All functions relying on the **yuima** package will get as much information as possible from the different slots of the `yuima-class` structure without replicating the same code twice. If there are missing pieces of information, some default values can be assumed.

Usage

```
setModel(drift = NULL, diffusion = NULL, hurst = 0.5, jump.coeff = NULL,
measure = list(), measure.type = character(), state.variable = "x",
jump.variable = "z", time.variable = "t", solve.variable, xinit)
```

Arguments

`drift`	a vector of `expressions` (the default value is 0 when `drift=NULL`).
`diffusion`	a matrix of `expressions` (the default value is 0 when `diffusion=NULL`).
`hurst`	the Hurst parameter of the gaussian noise. If `h=0.5`, the default, the process is Wiener otherwise it is fractional Brownian motion with that precise value of the Hurst index. Can be set to `NA` for further specification.
`jump.coeff`	a matrix of `expressions` for the jump component.
`measure`	Levy measure for jump variables.

measure.type	type specification for Levy measures.
state.variable	a vector of names of the state variables in the drift and diffusion coefficients.
jump.variable	a vector of names of the jump variables in the jump coefficient.
time.variable	the name of the time variable.
solve.variable	a vector of names of the variables in the left-hand-side of the equations in the model; solve.variable equals state.variable as long as we have no exogenous variable other than statistical parameters in the coefficients (drift and diffusion).
xinit	a vector of numbers identifying the initial value of the solve.variable.

Details

Please refer to the vignettes and the examples or to the **yuimadocs** package.

An object of yuima.model-class contains several slots:

drift: an R expression which specifies the drift coefficient (a vector).

diffusion: an R expression which specifies the diffusion coefficient (a matrix).

jump.coeff: coefficient of the jump term.

measure: the Levy measure of the driving Levy process.

measure.type: specifies the type of the measure, such as CP, code or density. See below.

parameter: a short name for "parameters". It is an object of model.parameter-class which is a list of vectors of names of parameters belonging to the single components of the model (drift, diffusion, jump and measure), the names of common parameters and the names of all parameters. For more details see model.parameter-class documentation page.

solve.variable: a vector of variable names, each element corresponds to the name of the solution variable (left-hand-side) of each equation in the model, in the corresponding order.

state.variable: identifies the state variables in the R expression. By default, it is assumed to be x.

jump.variable: the variable for the jump coefficient. By default, it is assumed to be z.

time: the time variable. By default, it is assumed to be t.

solve.variable: used to identify the solution variables in the R expression, i.e. the variable with respect to which the stochastic differential equation has to be solved. By default, it is assumed to be x, otherwise the user can choose any other model specification.

noise.number: denotes the number of sources of noise. Currently only for the Gaussian part.

equation.number: denotes the dimension of the stochastic differential equation.

dimension: the dimensions of the parameters in the parameter slot.

xinit: denotes the initial value of the stochastic differential equation.

The yuima.model-class structure assumes that the user either uses the default names for state.variable, jump.variable, solution.variable and time.variable or specifies his/her own names. All the rest of the terms in the R expressions are considered as parameters and identified accordingly in the parameter slot.

Value

model — an object of yuima.model-class.

Note

There may be missing information in the model description. Please contribute with suggestions and fixings.

Author(s)

The YUIMA Project Team

Examples

```
# Ex 1. (One-dimensional diffusion process)
# To describe
# dXt = -3*Xt*dt + (1/(1+Xt^2+t))dWt,
# we set
mod1 <- setModel(drift = "-3*x", diffusion = "1/(1+x^2+t)", solve.variable = c("x"))
# We may omit the solve.variable; then the default variable x is used
mod1 <- setModel(drift = "-3*x", diffusion = "1/(1+x^2+t)")
# Look at the model structure by
str(mod1)

# Ex 2. (Two-dimensional diffusion process with three factors)
# To describe
# dX1t = -3*X1t*dt + dW1t +x2t*dW3t,
# dX2t = -(X1t + 2*X2t)*dt + x1t*dW1t + 3*dW2t,
# we set the drift coefficient
a <- c("-3*x1","-x1-2*x2")
# and also the diffusion coefficient
b <- matrix(c("1","x1","0","3","x2","0"),2,3)
# Then set
mod2 <- setModel(drift = a, diffusion = b, solve.variable = c("x1","x2"))
# Look at the model structure by
str(mod2)
# The noise.number is automatically determined by inputting the diffusion matrix expression.
# If the dimensions of the drift differs from the number of the rows of the diffusion,
# the error message is returned.

# Ex 3. (Process with jumps)
## specify the jump term as c(x,t)dz
mod3 <- setModel(drift=c("-theta*x"), diffusion="sigma",
jump.coeff="1", measure=list(intensity="1", df=list("dnorm(z, 0, 1)")),
measure.type="CP", solve.variable="x")
# Look at the model structure by
str(mod3)

# Ex 4. (Process with fractional Gaussian noise)
# dYt = 3*Yt*dt + dWt^h
mod4 <- setModel(drift="3*y", diffusion=1, hurst=0.3, solve.variable=c("y"))
# Look at the model structure by
```

```
str(mod4)
```

---

setPoisson *Basic constructor for Compound Poisson processes*

Description

'setPoisson' construct a Compound Poisson model specification for a process of the form:

`Mt = m0+sum_{i=0}^Nt c*Y_{tau_i},  M0=m0`

where Nt is a homogeneous or time-inhomogeneous Poisson process, tau_i is the sequence of random times of Nt and Y is a sequence of i.i.d. random jumps.

Usage

```
setPoisson(intensity = 1, df = NULL, scale = 1, dimension=1, ...)
```

Arguments

`intensity`	either and expression or a numerical value representing the intensity function of the Poisson process Nt.
`df`	is the density of jump random variables Y.
`scale`	this is the scaling factor c.
`dimension`	this is the dimension of the jump component.
`...`	passed to `setModel`

Details

An object of `yuima.model-class` where the `model` slot is of class `yuima.poisson-class`.

Value

`model`	an object of `yuima.model-class`.

Author(s)

The YUIMA Project Team

Examples

```
Terminal <- 10
samp <- setSampling(T=Terminal,n=1000)

# Ex 1. (Simple homogeneous Poisson process)
mod1 <- setPoisson(intensity="lambda", df=list("dconst(z,1)"))
set.seed(123)
y1 <- simulate(mod1, true.par=list(lambda=1),sampling=samp)
plot(y1)
```

```
# scaling the jumps
mod2 <- setPoisson(intensity="lambda", df=list("dconst(z,1)"),scale=5)
set.seed(123)
y2 <- simulate(mod2, true.par=list(lambda=1),sampling=samp)
plot(y2)

# scaling the jumps through the constant distribution
mod3 <- setPoisson(intensity="lambda", df=list("dconst(z,5)"))
set.seed(123)
y3 <- simulate(mod3, true.par=list(lambda=1),sampling=samp)
plot(y3)

# Ex 2. (Time inhomogeneous Poisson process)
mod4 <- setPoisson(intensity="beta*(1+sin(lambda*t))", df=list("dconst(z,1)"))
set.seed(123)
lambda <- 3
beta <- 5
y4 <- simulate(mod4, true.par=list(lambda=lambda,beta=beta),sampling=samp)
par(mfrow=c(2,1))
par(mar=c(3,3,1,1))
plot(y4)
f <- function(t) beta*(1+sin(lambda*t))
curve(f, 0, Terminal, col="red")

# Ex 2. (Time inhomogeneous Compound Poisson process with Gaussian Jumps)
mod5 <- setPoisson(intensity="beta*(1+sin(lambda*t))", df=list("dnorm(z,mu,sigma)"))
set.seed(123)
y5 <- simulate(mod5, true.par=list(lambda=lambda,beta=beta,mu=0, sigma=2),sampling=samp)
plot(y5)
f <- function(t)  beta*(1+sin(lambda*t))
curve(f, 0, Terminal, col="red")
```

`setSampling` *Set sampling information and create a 'sampling' object.*

Description

`setSampling` is a constructor for `yuima.sampling-class`.

Usage

```
setSampling(Initial = 0, Terminal = 1, n = 100, delta,
 grid, random = FALSE, sdelta=as.numeric(NULL),
 sgrid=as.numeric(NULL), interpolation="pt" )
```

Arguments

Initial	Initial time of the grid.
Terminal	Terminal time of the grid.
n	number of time intervals.
delta	mesh size in case of regular time grid.
grid	a grid of times for the simulation, possibly empty.
random	specify if it is random sampling. See Details.
sdelta	mesh size in case of regular space grid.
sgrid	a grid in space for the simulation, possibly empty.
interpolation	a rule of interpolation in case of subsampling. By default, the previous tick interpolation. See Details.

Details

The function creates an object of type yuima.sampling-class with several slots.

Initial: initial time of the grid.

Terminal: terminal time fo the grid.

n: the number of observations - 1.

delta: in case of a regular time grid it is the mesh.

grid: the grid of times.

random: either FALSE or the distribution of the random times.

regular: indicator of whether the grid is regular or not. For internal use only.

sdelta: in case of a regular space grid it is the mesh.

sgrid: the grid in space.

oindex: in case of interpolation, a vector of indexes corresponding to the original observations used for the approximation.

interpolation: the name of the interpolation method used.

In case of subsampling, the observations are subsampled on some given grid/sgrid or according to some random times. When the original observations do not exist at a give point of the grid they are obtained by some approximation method. Available methods are "pt" or "previous tick" observation method, "nt" or "next tick" observation method, or by l"linear" interpolation. In case of interpolation, the slot oindex contains the vector of indexes corresponding to the original observations used for the approximation. For the linear method the index corresponds to the left-most observation.

The slot random is used as information in case a grid is already determined (e.g. n or delta, etc. ot the grid itself are given) or if some subsampling has occurred or if some particular method which causes a random grid is used in simulation (for example the space discretized Euler scheme). The slot random contains a list of two elements distr and scale, where distr is a the distribution of independent random times and scale is either a scaling constant or a scaling function. If the grid of times is deterministic, then random is FALSE.

If not specified and random=FALSE, the slot grid is filled automatically by the function. It is eventually modified or created after the call to the function simulate.

If delta is not specified, it is calculated as (Terminal-Initial)/n). If delta is specified, the Terminal is adjusted to be equal to Initial+n*delta.

The vectors delta, n, Initial and Terminal may have different lengths, but then they are extended to the maximal length to keep consistency. See examples.

If grid is specified, it takes precedence over all other arguments.

Value

An object of type yuima.sampling-class.

Author(s)

The YUIMA Project Team

Examples

```
samp <- setSampling(Terminal=1, n=1000)
str(samp)

samp <- setSampling(Terminal=1, n=1000, delta=0.3)
str(samp)


samp <- setSampling(Terminal=1, n=1000, delta=c(0.1,0.3))
str(samp)

samp <- setSampling(Terminal=1:3, n=1000)
str(samp)
```

---

setYuima — *Creates a "yuima" object by combining "model", "data", "sampling", "characteristic" and "functional"slots.*

Description

setYuima constructs an object of yuima-class.

Usage

```
setYuima(data, model, sampling, characteristic, functional)
```

Arguments

data	an object of yuima.data-class.
model	an object of yuima.model-class.
sampling	an object of yuima.sampling-class.
characteristic	an object of yuima.characteristic-class.
functional	an object of class yuima.functional-class.

Details

The yuima-class object is the main object of the **yuima** package. Some of the slots can be missing.

The slot data contains the data, either empirical or simulated.

The slot model contains the description of the (statistical) model which is used to generate the data via different simulation schemes, to draw inference from the data or both.

The sampling slot contains information on how the data have been collected or how they should be simulated.

The slot characteristic contains information on PLEASE FINISH THIS. The slot functional contains information on PLEASE FINISH THIS.

Please refer to the vignettes and the examples in the **yuimadocs** package for more informations.

Value

an object of yuima-class.

Author(s)

The YUIMA Project Team

Examples

```
# Creation of a yuima object with all slots for a
# stochastic differential equation
# dXt^e = -theta2 * Xt^e * dt + theta1 * dWt
diffusion <- matrix(c("theta1"), 1, 1)
drift <- c("-1*theta2*x")
ymodel <- setModel(drift=drift, diffusion=diffusion)
n <- 100
ysamp <- setSampling(Terminal=1, n=n)

yuima <- setYuima(model=ymodel, sampling=ysamp)

str(yuima)
```

simFunctional *Calculate the value of functional*

Description

Calculate the value of functional associated with sde by Euler scheme.

Usage

```
simFunctional(yuima, expand.var="e")
Fnorm(yuima, expand.var="e")
F0(yuima, expand.var="e")
```

Arguments

yuima	a yuima object containing model, functional and data.
expand.var	default expand.var="e".

Details

Calculate the value of functional of interest. Fnorm returns normalized one, and F0 returns the value for the case small parameter epsilon = 0. In simFunctional and Fnorm, yuima MUST contains the 'data' slot (X in legacy version)

Value

Fe	a real value

Note

we need to fix this routine.

Author(s)

YUIMA Project Team

Examples

```
set.seed(123)
# to the Black-Scholes economy:
# dXt^e = Xt^e * dt + e * Xt^e * dWt
diff.matrix <- matrix( c("x*e"), 1,1)
model <- setModel(drift = c("x"), diffusion = diff.matrix)
# call option is evaluated by averating
# max{ (1/T)*int_0^T Xt^e dt, 0}, the first argument is the functional of interest:
Terminal <- 1
xinit <- c(1)
f <- list( c(expression(x/Terminal)), c(expression(0)))
F <- 0
```

```
division <- 1000
e <- .3
samp <- setSampling(Terminal = Terminal, n = division)
yuima <- setYuima(model = model,sampling = samp)
yuima <- setFunctional( yuima, xinit=xinit, f=f,F=F,e=e)
# evaluate the functional value

yuima <- simulate(yuima,xinit=xinit,true.par=e)
Fe <- simFunctional(yuima)
Fe
Fenorm <- Fnorm(yuima)
Fenorm
```

`simulate` *Simulator function for multi-dimensional stochastic processes*

Description

Simulate multi-dimensional stochastic processes.

Usage

```
simulate(object, nsim=1, seed=NULL, xinit, true.parameter, space.discretized = FALSE,
 increment.W = NULL, increment.L = NULL, hurst, methodfGn = "WoodChan",
  sampling=sampling, subsampling=subsampling, ...)
```

Arguments

`object`	an `yuima-class`, `yuima.model-class` or `yuima.carma-class` object.
`xinit`	initial value vector of state variables.
`true.parameter`	named list of parameters.
`space.discretized`	flag to switch to space-discretized Euler Maruyama method.
`increment.W`	to specify Wiener increment for each time tics in advance.
`increment.L`	to specify Levy increment for each time tics in advance.
`nsim`	Not used yet. Included only to match the standard genenirc in package `stats`.
`seed`	Not used yet. Included only to match the standard genenirc in package `stats`.
`hurst`	value of Hurst parameter for simulation of the fGn. Overrides the specified hurst slot.
`methodfGn`	simulation methods for fractional Gaussian noise.
`...`	passed to `setSampling` to create a sampling
`sampling`	a `yuima.sampling-class` object.
`subsampling`	a `yuima.sampling-class` object.

Details

simulate is a function to solve SDE using the Euler-Maruyama method. This function supports usual Euler-Maruyama method for multidimensional SDE, and space discretized Euler-Maruyama method for one dimensional SDE.

It simulates solutions of stochastic differential equations with Gaussian noise, fractional Gaussian noise awith/without jumps.

If a yuima-class object is passed as input, then the sampling information is taken from the slot sampling of the object. If a yuima.carma-class object, a yuima.model-class object or a yuima-class object with missing sampling slot is passed as input the sampling argument is used. If this argument is missing then the sampling structure is constructed from Initial, Terminal, etc. arguments (see setSampling for details on how to use these arguments).

Value

yuima a yuima-class object.

Note

There may be missing information in the simulate description. Please contribute with suggestions and fixings.

Author(s)

The YUIMA Project Team

Examples

```
set.seed(123)

# Path-simulation for 1-dim diffusion process.
# dXt = -0.3*Xt*dt + dWt
mod <- setModel(drift="-0.3*y", diffusion=1, solve.variable=c("y"))
str(mod)

# Set the model in an `yuima' object with a sampling scheme.
T <- 1
n <- 1000
samp <- setSampling(Terminal=T, n=n)
ou <- setYuima(model=mod, sampling=samp)

# Solve SDEs using Euler-Maruyama method.
par(mfrow=c(3,1))
ou <- simulate(ou, xinit=1)
plot(ou)


set.seed(123)
ouB <- simulate(mod, xinit=1,sampling=samp)
plot(ouB)
```

```
set.seed(123)
ouC <- simulate(mod, xinit=1, Terminal=1, n=1000)
plot(ouC)

par(mfrow=c(1,1))


# Path-simulation for 1-dim diffusion process.
# dXt = theta*Xt*dt + dWt
mod1 <- setModel(drift="theta*y", diffusion=1, solve.variable=c("y"))
str(mod1)
ou1 <- setYuima(model=mod1, sampling=samp)

# Solve SDEs using Euler-Maruyama method.
ou1 <- simulate(ou1, xinit=1, true.p = list(theta=-0.3))
plot(ou1)

## Not run:

# A multi-dimensional (correlated) diffusion process.
# To describe the following model:
# X=(X1,X2,X3); dXt = U(t,Xt)dt + V(t)dWt
# For drift coeffcient
U <- c("-x1","-2*x2","-t*x3")
# For diffusion coefficient of X1
v1 <- function(t) 0.5*sqrt(t)
# For diffusion coefficient of X2
v2 <- function(t) sqrt(t)
# For diffusion coefficient of X3
v3 <- function(t) 2*sqrt(t)
# correlation
rho <- function(t) sqrt(1/2)
# coefficient matrix for diffusion term
V <- matrix( c( "v1(t)",
                "v2(t) * rho(t)",
                "v3(t) * rho(t)",
                "",
                "v2(t) * sqrt(1-rho(t)^2)",
                "",
                "",
                "",
                "v3(t) * sqrt(1-rho(t)^2)"
               ), 3, 3)
# Model sde using "setModel" function
cor.mod <- setModel(drift = U, diffusion = V,
state.variable=c("x1","x2","x3"),
solve.variable=c("x1","x2","x3") )
str(cor.mod)

# Set the `yuima' object.
cor.samp <- setSampling(Terminal=T, n=n)
cor <- setYuima(model=cor.mod, sampling=cor.samp)
```

```
# Solve SDEs using Euler-Maruyama method.
set.seed(123)
cor <- simulate(cor)
plot(cor)

# A non-negative process (CIR process)
# dXt= a*(c-y)*dt + b*sqrt(Xt)*dWt
 sq <- function(x){y = 0;if(x>0){y = sqrt(x);};return(y);}
 model<- setModel(drift="0.8*(0.2-x)",
  diffusion="0.5*sq(x)",solve.variable=c("x"))
 T<-10
 n<-1000
 sampling <- setSampling(Terminal=T,n=n)
 yuima<-setYuima(model=model, sampling=sampling)
 cir<-simulate(yuima,xinit=0.1)
 plot(cir)

# solve SDEs using Space-discretized Euler-Maruyama method
v4 <- function(t,x){
  return(0.5*(1-x)*sqrt(t))
}
mod_sd <- setModel(drift = c("0.1*x1", "0.2*x2"),
                    diffusion = c("v1(t)","v4(t,x2)"),
                    solve.var=c("x1","x2")
                    )
samp_sd <- setSampling(Terminal=T, n=n)
sd <- setYuima(model=mod_sd, sampling=samp_sd)
sd <- simulate(sd, xinit=c(1,1), space.discretized=TRUE)
plot(sd)


## example of simulation by specifying increments
## Path-simulation for 1-dim diffusion process
## dXt = -0.3*Xt*dt + dWt

mod <- setModel(drift="-0.3*y", diffusion=1,solve.variable=c("y"))
str(mod)

## Set the model in an `yuima' object with a sampling scheme.
Terminal <- 1
n <- 500
mod.sampling <- setSampling(Terminal=Terminal, n=n)
yuima.mod <- setYuima(model=mod, sampling=mod.sampling)

##use original increment
delta <- Terminal/n
my.dW <- rnorm(n * yuima.mod@model@noise.number, 0, sqrt(delta))
my.dW <- t(matrix(my.dW, nrow=n, ncol=yuima.mod@model@noise.number))

## Solve SDEs using Euler-Maruyama method.
yuima.mod <- simulate(yuima.mod,
                      xinit=1,
```

```
                    space.discretized=FALSE,
                    increment.W=my.dW)
if( !is.null(yuima.mod) ){
 dev.new()
 # x11()
  plot(yuima.mod)
}

## A multi-dimensional (correlated) diffusion process.
## To describe the following model:
## X=(X1,X2,X3); dXt = U(t,Xt)dt + V(t)dWt
## For drift coeffcient
U <- c("-x1","-2*x2","-t*x3")
## For process 1
diff.coef.1 <- function(t) 0.5*sqrt(t)
## For process 2
diff.coef.2 <- function(t) sqrt(t)
## For process 3
diff.coef.3 <- function(t) 2*sqrt(t)
## correlation
cor.rho <- function(t) sqrt(1/2)
## coefficient matrix for diffusion term
V <- matrix( c( "diff.coef.1(t)",
               "diff.coef.2(t) * cor.rho(t)",
               "diff.coef.3(t) * cor.rho(t)",
               "",
               "diff.coef.2(t)",
               "diff.coef.3(t) * sqrt(1-cor.rho(t)^2)",
               "diff.coef.1(t) * cor.rho(t)",
               "",
               "diff.coef.3(t)"
               ), 3, 3)
## Model sde using "setModel" function
cor.mod <- setModel(drift = U, diffusion = V,
                    solve.variable=c("x1","x2","x3") )
str(cor.mod)
## Set the `yuima' object.
set.seed(123)
obj.sampling <- setSampling(Terminal=Terminal, n=n)
yuima.obj <- setYuima(model=cor.mod, sampling=obj.sampling)

##use original dW
my.dW <- rnorm(n * yuima.obj@model@noise.number, 0, sqrt(delta))
my.dW <- t(matrix(my.dW, nrow=n, ncol=yuima.obj@model@noise.number))

## Solve SDEs using Euler-Maruyama method.
yuima.obj.path <- simulate(yuima.obj, space.discretized=FALSE,
 increment.W=my.dW)
if( !is.null(yuima.obj.path) ){
  dev.new()
#  x11()
  plot(yuima.obj.path)
}
```

```
##:: sample for Levy process ("CP" type)
## specify the jump term as c(x,t)dz
obj.model <- setModel(drift=c("-theta*x"), diffusion="sigma",
jump.coeff="1", measure=list(intensity="1", df=list("dnorm(z, 0, 1)")),
measure.type="CP", solve.variable="x")

##:: Parameters
lambda <- 3
theta <- 6
sigma <- 1
xinit <- runif(1)
N <- 500
h <- N^(-0.7)
eps <- h/50
n <- 50*N
T <- N*h

set.seed(123)
obj.sampling <- setSampling(Terminal=T, n=n)
obj.yuima <- setYuima(model=obj.model, sampling=obj.sampling)
X <- simulate(obj.yuima, xinit=xinit, true.parameter=list(theta=theta, sigma=sigma))
dev.new()
plot(X)


##:: sample for Levy process ("CP" type)
## specify the jump term as c(x,t,z)
## same plot as above example
obj.model <- setModel(drift=c("-theta*x"), diffusion="sigma",
jump.coeff="z", measure=list(intensity="1", df=list("dnorm(z, 0, 1)")),
measure.type="CP", solve.variable="x")

set.seed(123)
obj.sampling <- setSampling(Terminal=T, n=n)
obj.yuima <- setYuima(model=obj.model, sampling=obj.sampling)
X <- simulate(obj.yuima, xinit=xinit, true.parameter=list(theta=theta, sigma=sigma))
dev.new()
plot(X)




##:: sample for Levy process ("code" type)
## dX_{t} = -x dt + dZ_t
obj.model <- setModel(drift="-x", xinit=1, jump.coeff="1", measure.type="code",
measure=list(df="rIG(z, 1, 0.1)"))
obj.sampling <- setSampling(Terminal=10, n=10000)
obj.yuima <- setYuima(model=obj.model, sampling=obj.sampling)
result <- simulate(obj.yuima)
dev.new()
plot(result)
```

```
##:: sample for multidimensional Levy process ("code" type)
## dX = (theta - A X)dt + dZ,
##    theta=(theta_1, theta_2) = c(1,.5)
##    A=[a_ij], a_11 = 2, a_12 = 1, a_21 = 1, a_22=2
require(yuima)
x0 <- c(1,1)
beta <- c(.1,.1)
mu <- c(0,0)
delta0 <- 1
alpha <- 1
Lambda <- matrix(c(1,0,0,1),2,2)
cc <- matrix(c(1,0,0,1),2,2)
obj.model <- setModel(drift=c("1 - 2*x1-x2",".5-x1-2*x2"), xinit=x0,
solve.variable=c("x1","x2"), jump.coeff=cc, measure.type="code",
 measure=list(df="rNIG(z, alpha, beta, delta0, mu, Lambda)"))
obj.sampling <- setSampling(Terminal=10, n=10000)
obj.yuima <- setYuima(model=obj.model, sampling=obj.sampling)
result <- simulate(obj.yuima,true.par=list( alpha=alpha,
 beta=beta, delta0=delta0, mu=mu, Lambda=Lambda))
plot(result)


# Path-simulation for a Carma(p=2,q=1) model driven by a Brownian motion:
carma1<-setCarma(p=2,q=1)
str(carma1)

# Set the sampling scheme
samp<-setSampling(Terminal=100,n=10000)

# Set the values of the model parameters
par.carma1<-list(b0=1,b1=2.8,a1=2.66,a2=0.3)

set.seed(123)
sim.carma1<-simulate(carma1,
                     true.parameter=par.carma1,
                     sampling=samp)

plot(sim.carma1)



# Path-simulation for a Carma(p=2,q=1) model driven by a Compound Poisson process.
carma1<-setCarma(p=2,
                 q=1,
                 measure=list(intensity="1",df=list("dnorm(z, 0, 1)")),
                 measure.type="CP")

# Set Sampling scheme
samp<-setSampling(Terminal=100,n=10000)

# Fix carma parameters
par.carma1<-list(b0=1,
```

```
                  b1=2.8,
                  a1=2.66,
                  a2=0.3)

set.seed(123)
sim.carma1<-simulate(carma1,
                     true.parameter=par.carma1,
                     sampling=samp)

plot(sim.carma1)

## End(Not run)
```

---

subsampling *subsampling*

Description

subsampling

Usage

```
subsampling(x, sampling, ...)
```

Arguments

x	an yuima-class or yuima.model-class object.
sampling	a yuima.sampling-class object.
...	used to create a sampling structure

Details

When subsampling on some grid of times, it may happen that no data is available at the given grid point. In this case it is possible to use several techniques. Different options are avaiable specifying the argument, or the slot, interpolation:

"none" **or** "exact" no interpolation. If no data point exists at a given grid point, NA is returned in the subsampled data

"pt" **or** "previous" the first data on the left of the grid point instant is used.

"nt" **or** "next" the first data on the right of the grid point instant is used.

"lin" **or** "linear" the average of the values of the first data on the left and the first data to the right of the grid point instant is used.

Value

yuima	a yuima.data-class object.

Author(s)

The YUIMA Project Team

Examples

```
## Set a model
diff.coef.1 <- function(t, x1=0, x2) x2*(1+t)
diff.coef.2 <- function(t, x1, x2=0) x1*sqrt(1+t^2)
cor.rho <- function(t, x1=0, x2=0) sqrt((1+cos(x1*x2))/2)
diff.coef.matrix <- matrix(c("diff.coef.1(t,x1,x2)",
"diff.coef.2(t,x1,x2)*cor.rho(t,x1,x2)", "",
"diff.coef.2(t,x1,x2)*sqrt(1-cor.rho(t,x1,x2)^2)"),2,2)
cor.mod <- setModel(drift=c("","",""), diffusion=diff.coef.matrix,
solve.variable=c("x1", "x2"), xinit=c(3,2))
set.seed(111)

## We first simulate the two dimensional diffusion model
yuima.samp <- setSampling(Terminal=1, n=1200)
yuima <- setYuima(model=cor.mod, sampling=yuima.samp)
yuima.sim <- simulate(yuima)

plot(yuima.sim, plot.type="single")

## random sampling with exponential times
## one random sequence per time series
newsamp <- setSampling(
 random=list(rdist=c( function(x) rexp(x, rate=10),
  function(x) rexp(x, rate=20))) )
newdata <- subsampling(yuima.sim, sampling=newsamp)
points(get.zoo.data(newdata)[[1]],col="red")
points(get.zoo.data(newdata)[[2]],col="green")


plot(yuima.sim, plot.type="single")

## deterministic subsampling with different
## frequence for each time series
newsamp <- setSampling(delta=c(0.1,0.2))
newdata <- subsampling(yuima.sim, sampling=newsamp)
points(get.zoo.data(newdata)[[1]],col="red")
points(get.zoo.data(newdata)[[2]],col="green")
```

`toLatex` *Additional Methods for LaTeX Representations for Yuima objects*

Description

These methods convert `yuima-class`, `yuima.model-class` or `yuima.carma-class` objects to character vectors with LaTeX markup.

Usage

```
## S3 method for class 'yuima'
toLatex(object,...)
## S3 method for class 'yuima.model'
toLatex(object,...)
## S3 method for class 'yuima.carma'
toLatex(object,...)
```

Arguments

object	object of a class yuima, yuima.model or yuima.carma.
...	currently not used.

Details

This methos tries to convert a formal description of the model slot of the yuima object into a LaTeX formula. This is just a simple proof of concept and probably further LaTex manipulations for use in papers. Copy and paste of the output of `toLatex` into a real LaTeX file should do the job.

Examples

```
# dXt = theta*Xt*dt + dWt
mod1 <- setModel(drift="theta*y", diffusion=1, solve.variable=c("y"))
str(mod1)
toLatex(mod1)

# A multi-dimensional (correlated) diffusion process.
# To describe the following model:
# X=(X1,X2,X3); dXt = U(t,Xt)dt + V(t)dWt
# For drift coeffcient
U <- c("-x1","-2*x2","-t*x3")
# For diffusion coefficient of X1
v1 <- function(t) 0.5*sqrt(t)
# For diffusion coefficient of X2
v2 <- function(t) sqrt(t)
# For diffusion coefficient of X3
v3 <- function(t) 2*sqrt(t)
# correlation
rho <- function(t) sqrt(1/2)
# coefficient matrix for diffusion term
V <- matrix( c( "v1(t)",
                "v2(t) * rho(t)",
                "v3(t) * rho(t)",
                "",
                "v2(t) * sqrt(1-rho(t)^2)",
                "",
                "",
                "",
                "v3(t) * sqrt(1-rho(t)^2)"
               ), 3, 3)
# Model sde using "setModel" function
```

```
cor.mod <- setModel(drift = U, diffusion = V,
state.variable=c("x1","x2","x3"),
solve.variable=c("x1","x2","x3") )
str(cor.mod)
toLatex(cor.mod)

# A CARMA(p=3,q=1) process.
carma1<-setCarma(p=3,q=1,loc.par="c",scale.par="s")
str(carma1)
toLatex(carma1)
```

---

yuima-class *Class for stochastic differential equations*

Description

The yuima S4 class is a class of the **yuima** package.

Details

The yuima-class object is the main object of the **yuima** package. Some of the slots may be missing.

The data slot contains the data, either empirical or simulated.

The model contains the description of the (statistical) model which is used to generate the data via different simulation schemes, to draw inference from the data or both.

The sampling slot contains information on how the data have been collected or how they should be generated.

The slot characteristic contains information on PLEASE FINISH THIS. The slot functional contains information on PLEASE FINISH THIS.

Slots

data: an object of class yuima.data-class

model: an object of class yuima.model-class

sampling: an object of class yuima.sampling-class

characteristic: an object of class yuima.characteristic-class

functional: an object of class yuima.functional-class

Methods

new signature(x = "yuima", data = "yuima.data", model = "yuima.model", sampling = "yuima.sampling", the function makes a copy of the prototype object from the class definition of yuima-class, then calls the initialize method passing as arguments the newly created object and the remaining arguments.

initialize signature(x = "yuima", data = "yuima.data", model = "yuima.model", sampling = "yuima.samplin makes a copy of each argument in the corresponding slots of the object x.

get.data `signature(x = "yuima")`: returns the content of the slot `data`.

plot `signature(x = "yuima", ...)`: calls `plot` from the `zoo` package with argument `x@data@zoo.data`. Additional arguments `...` are passed as is to the `plot` function.

dim `signature(x = "yuima")`: the number of SDEs in the `yuima` object.

length `signature(x = "yuima")`: a vector of length of each SDE described in the `yuima` object.

cce `signature(x = "yuima")`: calculates the asyncronous covariance estimator on the data contained in `x@data@zoo.data`. For more details see `cce`.

llag `signature(x = "yuima")`: calculates the lead lag estimate r on the data contained in `x@data@zoo.data`. For more details see `llag`.

simulate simulation method. For more information see `simulate`.

cbind.yuima `signature(x = "yuima")`: bind yuima.data object.

Author(s)

The YUIMA Project Team

`yuima.carma-class` *Class for the mathematical description of CARMA(p,q) model*

Description

The `yuima.carma` class is a class of the **yuima** package that extends the `yuima.model-class`.

Slots

`info`**:** is an `carma.info-class` object that describes the structure of the CARMA(p,q) model.

`drift`**:** is an R expression which specifies the drift coefficient (a vector).

`diffusion`**:** is an R expression which specifies the diffusion coefficient (a matrix).

`hurst`**:** the Hurst parameter of the gaussian noise. If `h=0.5`, the process is Wiener otherwise it is fractional Brownian motion with that precise value of the Hurst index. Can be set to `NA` for further specification.

`jump.coeff`**:** a vector of `expressions` for the jump component.

`measure`**:** Levy measure for jump variables.

`measure.type`**:** Type specification for Levy measures.

state.variable a vector of names identifying the names used to denote the state variable in the drift and diffusion specifications.

`parameter`**:** which is a short name for "parameters", is an object of class `model.parameter-class`. For more details see `model.parameter-class` documentation page.

`state.variable`**:** identifies the state variables in the R expression.

`jump.variable`**:** identifies the variable for the jump coefficient.

`time.variable`**:** the time variable.

`noise.number`: denotes the number of sources of noise. Currently only for the Gaussian part.

`equation.number`: denotes the dimension of the stochastic differential equation.

`dimension`: the dimensions of the parameter given in the `parameter` slot.

`solve.variable`: identifies the variable with respect to which the stochastic differential equation has to be solved.

`xinit`: contains the initial value of the stochastic differential equation.

`J.flag`: wheather jump.coeff include jump.variable.

Methods

simulate simulation method. For more information see `simulate`.

toLatex This method converts an object of `yuima.carma-class` to character vectors with LaTeX markup.

CarmaNoise Recovering underlying Levy. For more information see `CarmaNoise`.

qmle Quasi maximum likelihood estimation procedure. For more information see `qmle`.

Author(s)

The YUIMA Project Team

`yuima.carma.qmle-class` *Class for Quasi Maximum Likelihood Estimation of CARMA(p,q) model*

Description

The `yuima.carma.qmle` class is a class of the **yuima** package that extends the `mle-class` of the **stats4** package.

Slots

`Incr.Lev`: is an object of class `zoo` that contains the estimated increments of the noise obtained using `CarmaNoise`.

`model`: is an object of of `yuima.carma-class`.

`logL.Incr`: is an object of class `numeric` that contains the value of the log-likelihood for estimated Levy increments.

`call`: is an object of class `language`.

`coef`: is an object of class `numeric` that contains estimated parameters.

`fullcoef`: is an object of class `numeric` that contains estimated and fixed parameters.

`vcov`: is an object of class `matrix`.

`min`: is an object of class `numeric`.

`minuslogl`: is an object of class `function`.

`method`: is an object of class `character`.

Methods

plot Plot method for estimated increment of the noise.

Methods mle All methods for `mle-class` are available.

Author(s)

The YUIMA Project Team

---

yuima.characteristic-class *Classe for stochastic differential equations characteristic scheme*

Description

The `yuima.characteristic` class is a class of the **yuima** package.

Slots

`equation.number`**:** The number of equations modeled in the `yuima` object.

`time.scale`**:** The time scale assumed in the `yuima` object.

Author(s)

The YUIMA Project Team

---

yuima.CP.qmle-class *Class for Quasi Maximum Likelihood Estimation of Compound Poisson-based and SDE models*

Description

The `yuima.CP.qmle` class is a class of the **yuima** package that extends the `mle-class` of the **stats4** package.

Slots

`Jump.times`**:** a vector which contains the estimated time of jumps.

`Jump.values`**:** a vector which contains the jumps.

`X.values`**:** the value of the process at the jump times.

`model`**:** is an object of of `yuima.model-class`.

`call`**:** is an object of class `language`.

`coef`**:** is an object of class `numeric` that contains estimated parameters.

`fullcoef`**:** is an object of class `numeric` that contains estimated and fixed parameters.

`vcov`: is an object of class `matrix`.

`min`: is an object of class `numeric`.

`minuslogl`: is an object of class `function`.

`method`: is an object of class `character`.

`model`: is an object of class `yuima.model-class`.

Methods

plot Plot method for plotting the jump times.

Methods mle All methods for `mle-class` are available.

Author(s)

The YUIMA Project Team

`yuima.data-class` *Class "yuima.data" for the data slot of a "yuima" class object*

Description

The `yuima.data-class` is a class of the **yuima** package used to store the data which are hold in the slot `data` of an object of the `yuima-class`.

Objects from this class contain either true data or simulated data.

Details

Objects in this class are created or initialized using the methods `new` or `initialize` or via the function `setData`. The preferred way to construct an object in this class is to use the function `setData`.

Objects in this class are used to store the data which are hold in the slot `data` of an object of the `yuima-class`.

Objects in this class contain two slots described here.

`original.data`: The slot `original.data` contains, as the name suggests, a copy of the original data passed by the user to methods `new` or `initialize` or to the function `setData`. It is intended for backup purposes.

`zoo.data`: When a new object of this class is created or initialized using the `original.data`, the package tries to convert `original.data` into an object of class `zoo`. Once coerced to `zoo`, the data are stored in the slot `zoo.data`.

If the conversion fails, the initialization or creation of the object fails.

Internally, the **yuima** package stores and operates on `zoo`-type objects.

If data are obtained by simulation, the `original.data` slot is usually empty.

Slots

`original.data`**:** The original data.

`zoo.data`**:** A list of zoo format data.

Methods

new `signature(x = "yuima.data", original.data)`: the function makes a copy of the prototype object from the class definition of `yuima.data-class`, then calls the `initialize` method passing as arguments the newly created object and the `original.data`.

initialize `signature(x = "yuima.data", original.data)`: makes a copy of `original.data` into the slot `original.data` of `x` and tries to coerce `original.data` into an object of class `zoo`. The result is put in the slot `zoo.data` of `x`. If coercion fails, the `intialize` method fails as well.

get.zoo.data `signature(x = "yuima.data")`: returns the content of the slot `zoo.data` of `x`.

plot `signature(x = "yuima.data", ...)`: calls `plot` from the `zoo` package with argument `x@zoo.data`. Additional arguments `...` are passed as is to the `plot` function.

dim `signature(x = "yuima.data")`: calls `dim` from the `zoo` package with argument `x@zoo.data`.

length `signature(x = "yuima.data")`: calls `length` from the `zoo` package with argument `x@zoo.data`.

cce `signature(x = "yuima.data")`: calculates asyncronous covariance estimator on the data contained in `x@zoo.data`. For more details see `cce`.

llag `signature(x = "yuima.data")`: calculates lead lag estimate on the data contained in `x@zoo.data`. For more details see `llag`.

cbind.yuima `signature(x = "yuima.data")`: bind yuima.data object.

Author(s)

The YUIMA Project Team

---

yuima.functional-class

Classes for stochastic differential equations data object

Description

The `yuima.functional` class is a class of the **yuima** package.

Author(s)

YUIMA Project

yuima.model-class *Classes for the mathematical description of stochastic differential equations*

Description

The yuima.model class is a class of the **yuima** package.

Slots

drift: is an R expression which specifies the drift coefficient (a vector).

diffusion: is an R expression which specifies the diffusion coefficient (a matrix).

hurst: the Hurst parameter of the gaussian noise. If h=0.5, the process is Wiener otherwise it is fractional Brownian motion with that precise value of the Hurst index. Can be set to NA for further specification.

jump.coeff: a matrix of expressions for the jump component.

measure: Levy measure for jump variables.

measure.type: Type specification for Levy measures.

state.variable a vector of names identifying the names used to denote the state variable in the drift and diffusion specifications.

parameter: which is a short name for "parameters", is an object of class model.parameter-class. For more details see model.parameter-class documentation page.

state.variable: identifies the state variables in the R expression.

jump.variable: identifies the variable for the jump coefficient.

time.variable: the time variable.

noise.number: denotes the number of sources of noise. Currently only for the Gaussian part.

equation.number: denotes the dimension of the stochastic differential equation.

dimension: the dimensions of the parameter given in the parameter slot.

solve.variable: identifies the variable with respect to which the stochastic differential equation has to be solved.

xinit: contains the initial value of the stochastic differential equation.

J.flag: wheather jump.coeff include jump.variable.

Author(s)

The YUIMA Project Team

yuima.poisson-class *Class for the mathematical description of Compound Poisson processes*

Description

The yuima.poisson class is a class of the **yuima** package that extends the yuima.model-class.

Slots

drift: always expression((0)).

diffusion: a list of expression((0)).

hurst: always h=0.5, but ignored for this model.

jump.coeff: set according to scale in setPoisson.

measure: a list containting the intensity measure and the jump distribution.

measure.type: always "CP".

state.variable a vector of names identifying the names used to denote the state variable in the drift and diffusion specifications.

parameter: which is a short name for "parameters", is an object of class model.parameter-class. For more details see model.parameter-class documentation page.

state.variable: identifies the state variables in the R expression.

jump.variable: identifies the variable for the jump coefficient.

time.variable: the time variable.

noise.number: denotes the number of sources of noise.

equation.number: denotes the dimension of the stochastic differential equation.

dimension: the dimensions of the parameter given in the parameter slot.

solve.variable: identifies the variable with respect to which the stochastic differential equation has to be solved.

xinit: contains the initial value of the stochastic differential equation.

J.flag: wheather jump.coeff include jump.variable.

Methods

simulate simulation method. For more information see simulate.

qmle Quasi maximum likelihood estimation procedure. For more information see qmle.

Author(s)

The YUIMA Project Team

yuima.sampling-class *Classes for stochastic differential equations sampling scheme*

Description

The yuima.sampling class is a class of the **yuima** package.

Details

This object is created by setSampling or as a result of a simulation scheme by the simulate function or after subsampling via the function subsampling.

Slots

Initial: initial time of the grid.

Terminal: terminal time fo the grid.

n: the number of observations - 1.

delta: in case of a regular grid is the mesh.

grid: the grid of times.

random: either FALSE or the distribution of the random times.

regular: indicator of whether the grid is regular or not. For internal use only.

sdelta: in case of a regular space grid it is the mesh.

sgrid: the grid in space.

oindex: in case of interpolation, a vector of indexes corresponding to the original observations used for the approximation.

interpolation: the name of the interpolation method used.

Author(s)

The YUIMA Project Team

Index