

# Package ‘yuima’

April 29, 2017

**Type** Package

**Title** The YUIMA Project Package for SDEs

**Version** 1.6.4

**Depends** R(>= 2.10.0), methods, zoo, stats4, utils, expm, cubature,  
mvtnorm

**Imports** Rcpp (>= 0.12.1)

**Author** YUIMA Project Team

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

**Description** Simulation and Inference for SDEs and Other Stochastic Processes.

**License** GPL-2

**URL** <http://www.yuima-project.com>

**LinkingTo** Rcpp, RcppArmadillo

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2017-04-29 17:22:22 UTC

## R topics documented:

adaBayes . . . . .	3
asymptotic_term . . . . .	5
bns.test . . . . .	7
carma.info-class . . . . .	9
CarmaNoise . . . . .	10
cce . . . . .	13
cogarch.est.-class . . . . .	21
cogarch.est.incr-class . . . . .	22
cogarch.info-class . . . . .	23
cogarchNoise . . . . .	24
CPoint . . . . .	25
Diagnostic.Carma . . . . .	28
Diagnostic.Cogarch . . . . .	29

<code>gmm</code>	31
<code>hyavar</code>	33
<code>IC</code>	36
<code>info.Map-class</code>	40
<code>info.Ppr</code>	40
<code>Integral.sde</code>	40
<code>Integrand</code>	40
<code>Intensity.PPR</code>	41
<code>lambdaFromData</code>	41
<code>lasso</code>	42
<code>LawMethods</code>	43
<code>limiting.gamma</code>	44
<code>llag</code>	46
<code>LogSPX</code>	50
<code>lseBayes</code>	51
<code>mllag</code>	53
<code>mmfrac</code>	56
<code>model.parameter-class</code>	57
<code>mpv</code>	58
<code>MWK151</code>	60
<code>noisy.sampling</code>	61
<code>param.Integral</code>	63
<code>param.Map-class</code>	63
<code>phi.test</code>	64
<code>poisson.random.sampling</code>	65
<code>qgv</code>	66
<code>qmlle</code>	67
<code>qmlleLevy</code>	75
<code>rconst</code>	78
<code>rng</code>	79
<code>setCarma</code>	84
<code>setCharacteristic</code>	87
<code>setCogarch</code>	88
<code>setData</code>	90
<code>setFunctional</code>	91
<code>setHawkes</code>	93
<code>setIntegral</code>	94
<code>setLaw</code>	95
<code>setMap</code>	97
<code>setModel</code>	98
<code>setPoisson</code>	101
<code>setPpr</code>	103
<code>setSampling</code>	104
<code>setYuima</code>	106
<code>simFunctional</code>	107
<code>simulate</code>	108
<code>spectralcov</code>	115
<code>subsampling</code>	119

toLatex . . . . .	121
variable.Integral . . . . .	123
ybook . . . . .	123
yuima-class . . . . .	124
yuima.carma-class . . . . .	125
yuima.carma.qmle-class . . . . .	126
yuima.characteristic-class . . . . .	127
yuima.cogarch-class . . . . .	127
yuima.CP.qmle-class . . . . .	128
yuima.data-class . . . . .	129
yuima.functional-class . . . . .	130
yuima.Hawkes . . . . .	130
yuima.Integral-class . . . . .	131
yuima.law-class . . . . .	132
yuima.Map-class . . . . .	133
yuima.model-class . . . . .	134
yuima.multimodel-class . . . . .	135
yuima.poisson-class . . . . .	138
yuima.Ppr . . . . .	139
yuima.sampling-class . . . . .	139

**Index****141**

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 = "mcmc", mcmc = 1000,
rate = 1, rcpp = TRUE, algorithm = "randomwalk")
```

**Arguments**

yuima	a 'yuima' object.
start	initial suggestion for parameter values
prior	a list of prior distributions for the parameters specified by 'code'. Currently, dunif(z, min, max), dnorm(z, mean, sd), dbeta(z, shape1, shape2), dgamma(z, shape, rate) are available.
lower	a named list for specifying lower bounds of parameters
upper	a named list for specifying upper bounds of parameters
method	nomcmc requires package cubature
mcmc	number of iteration of Markov chain Monte Carlo method

rate	a thinning parameter. Only the first $n^{\wedge}rate$ observation will be used for inference.
rcpp	Logical value. If <code>rcpp = TRUE</code> (default), Rcpp code will be performed. Otherwise, usual R code will be performed.
algorithm	Logical value when <code>method = mcmc</code> . If <code>algorithm = randomwalk</code> (default), the random-walk Metropolis algorithm will be performed. If <code>algorithm = MpCN</code> , the Mixed preconditioned Crank-Nicolson algorithm will be performed.

### Details

Calculate the Bayes estimator for stochastic processes by using the quasi-likelihood function. The calculation is performed by the Markov chain Monte Carlo method. Currently, the Random-walk Metropolis algorithm and the Mixed preconditioned Crank-Nicolson algorithm is implemented.

### Value

vector            a vector of the paramter estimate

### Note

`algorithm = nomcmc` is unstable.

### Author(s)

Kengo Kamatani with YUIMA project Team

### References

- Yoshida, N. (2011). Polynomial type large deviation inequalities and quasi-likelihood analysis for stochastic differential equations. *Annals of the Institute of Statistical Mathematics*, 63(3), 431-479.
- Uchida, M., & Yoshida, N. (2014). Adaptive Bayes type estimators of ergodic diffusion processes from discrete observations. *Statistical Inference for Stochastic Processes*, 17(2), 181-219.
- Kamatani, K. (2017). Ergodicity of Markov chain Monte Carlo with reversible proposal. *Journal of Applied Probability*, 54(2).

### Examples

```
## Not run:
set.seed(123)

b <- c("-theta1*x1+theta2*sin(x2)+50", "-theta3*x2+theta4*cos(x1)+25")
a <- matrix(c("4+theta5", "1", "1", "2+theta6"), 2, 2)

true = list(theta1 = 0.5, theta2 = 5, theta3 = 0.3,
            theta4 = 5, theta5 = 1, theta6 = 1)
lower = list(theta1=0.1, theta2=0.1, theta3=0,
            theta4=0.1, theta5=0.1, theta6=0.1)
upper = list(theta1=1, theta2=10, theta3=0.9,
            theta4=10, theta5=10, theta6=10)
start = list(theta1=runif(1),
            theta2=rnorm(1),
```

```

theta3=rbeta(1,1,1),
theta4=rnorm(1),
theta5=rgamma(1,1,1),
theta6=rexp(1))

yuimamodel <- setModel(drift=b,diffusion=a,state.variable=c("x1", "x2"),solve.variable=c("x1","x2"))
yuimasamp <- setSampling(Terminal=50,n=50*10)
yuima <- setYuima(model = yuimamodel, sampling = yuimasamp)
yuima <- simulate(yuima, xinit = c(100,80),
                 true.parameter = true,sampling = yuimasamp)

prior <-
  list(
    theta1=list(measure.type="code",df="dunif(z,0,1)"),
    theta2=list(measure.type="code",df="dnorm(z,0,1)"),
    theta3=list(measure.type="code",df="dbeta(z,1,1)"),
    theta4=list(measure.type="code",df="dgamma(z,1,1)"),
    theta5=list(measure.type="code",df="dnorm(z,0,1)"),
    theta6=list(measure.type="code",df="dnorm(z,0,1)")
  )

set.seed(123)
mle <- qmle(yuima, start = start, lower = lower, upper = upper, method = "L-BFGS-B",rcpp=TRUE)
print(mle@coef)
bayes <- adaBayes(yuima, start=start, prior=prior,
                 method="mcmc",
                 mcmc=1000,rcpp=TRUE, lower = lower, upper = upper)

print(bayes@coef)

## End(Not run)

```

---

asymptotic\_term

*asymptotic expansion of the expected value of the functional*


---

## Description

calculate the first 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  $d0 + \text{epsilon} * d1$  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 averaging
# 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)
asyp <- asymptotic_term(yuima, block=10, rho,g)
asyp
sum(asyp$d0 + e * asyp$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 <a href="#">yuima-class</a> or <a href="#">yuima.data-class</a> .
r	a vector of non-negative numbers or a list of vectors of non-negative numbers. Theoretically, it is necessary that $\text{sum}(r)=4$ and $\text{max}(r)<2$ .
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  $\sqrt{n} * (\text{mpv}(\text{yuima}, 2) - \text{mpv}(\text{yuima}, c(1, 1))) /$  when type=“standard”,  $\sqrt{n} * \log(\text{mpv}(\text{yuima}, 2) / \text{mpv}(\text{yuima}, c(1, 1))) / \sqrt{\text{vartheta} * \text{mpv}(\text{yuima}, r) / \text{mpv}(\text{yuima}, c(1, 1))}$  when type=“log” and  $\sqrt{n} * (1 - \text{mpv}(\text{yuima}, c(1, 1)) / \text{mpv}(\text{yuima}, 2)) / \sqrt{\text{vartheta} * \text{mpv}(\text{yuima}, r) / \text{mpv}(\text{yuima}, c(1, 1))}$  when type=“ratio”. Here,  $n$  is equal to the length of the  $i$ -th component of the zoo.data of yuima minus 1 and  $\text{vartheta}$  is  $\pi^2/4 + \pi - 5$ . When adj=TRUE,  $\text{mpv}(\text{yuima}, r)[i] / \text{mpv}(\text{yuima}, c(1, 1))^2[i]$  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 “hctest” 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:  $dX_t = t \cdot dW_t + t \cdot dz_t$ ,
## where  $z_t$  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:  $dX_k t = t \cdot dW_{k,t}$  ( $k=1,2,3$ ) (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, 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 <a href="#">yuima-class</a> or <a href="#">yuima.data-class</a> .
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. (2013), 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 $\text{ceiling}(c.\text{two} \cdot n^{(2/3)})$ , 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 by Eq.(63) of 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 $C=\text{matrix}(c.\text{multi}, d, d)$ and $d=\text{dim}(x)$ . 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 by Eq.(37) of Zhang (2006).
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 $c.\text{RK} \cdot n^{\text{eta}}$ , 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. $\text{ftregion}=0$ (the default) means that a non-flat-top realized kernel studied in Barndorff-Nielsen et al. (2011) is used. $\text{ftregion}=1/H$ means that a flat-top realized kernel studied in Barndorff-Nielsen et al. (2008) is used. See Varneskov (2015) for other values.
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.
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.

<code>nvar.init</code>	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.
<code>ncov.init</code>	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.
<code>mn</code>	a positive integer indicating the number of terms to be used for calculating the SIML estimator. The default value is $\text{ceiling}(n^{\alpha})$ , where $n$ is the number of the refresh times associated with the data minus 1.
<code>alpha</code>	a positive number indicating the order of <code>mn</code> .
<code>frequency</code>	a positive integer indicating the frequency (seconds) of the calendar time sampling to be used.
<code>avg</code>	logical. If TRUE, the averaged subsampling estimator is calculated. Otherwise the simple sparsely subsampled estimator is calculated.
<code>threshold</code>	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 <code>threshold</code> are recycled if there are too few elements in <code>threshold</code> . The default value is determined following Koike (2014) (for the method "THY") and Koike (2015) (for the method "PTHY").
<code>utime</code>	a positive number indicating what seconds the number "1" corresponds to. The default value is 23400 (this means 6.5 hours) if the time indices of the (first component) of <code>x</code> is numeric, otherwise 1.
<code>psd</code>	logical. If TRUE, the estimated covariance matrix <code>C</code> is converted to $(C\%*\%C)^{1/2}$ for ensuring the positive semi-definiteness. In this case the absolute values of the estimated correlations are always ensured to be less than or equal to 1.

## Details

This function is a method for objects of `yuima.data-class` and `yuima-class`. It extracts the data slot when applied to 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",vol.init=NULL,covol.init=NULL,
    nvar.init=NULL,ncov.init=NULL,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 (2013) 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 Covariance.

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  $\text{ceiling}(\text{theta} * N)$ , where  $N$  is a matrix whose diagonal components are identical with the vector  $\text{length}(x) - 1$  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  $\text{ceiling}(\text{mean}(\text{theta}) * \text{sqrt}(n))$ , 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  $\text{ceiling}(\text{theta} * N_0)$ , where  $N_0$  is a matrix whose diagonal components are identical with the vector  $\text{length}(x) - 1$  and whose  $(i, j)$ -th component is identical with  $(\text{length}(x)[i] - 1) + (\text{length}(x)[j] - 1)$ . 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. (2013)).

For the method "QMLE", the optimization of the quasi-likelihood function is implemented via `arima0` using the fact that it can be seen as an MA(1) model's one: See Hansen et al. (2008) for details.

### Value

A list with components:

<code>covmat</code>	the estimated covariance matrix
<code>cormat</code>	the estimated correlation matrix

### Note

The example shows the central limit theorem for the nonsynchronous covariance estimator.

Estimation of the asymptotic variance can be implemented by `hyavar`. 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. (2013) On covariation estimation for multivariate continuous Ito semimartingales with noise in non-synchronous observation schemes, *Journal of Multivariate Analysis* **120** 59–84.

- Hansen, P. R., Large, J. and Lunde, A. (2008) Moving average-based estimators of integrated variance, *Econometric Reviews*, **27**, 79–111.
- 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. (2015) Estimation of integrated covariances in the simultaneous presence of nonsynchronicity, microstructure noise and jumps, *Econometric Theory*, doi:10.1017/S0266466614000954.
- Koike, Y. (2014) An estimator for the cumulative co-volatility of asynchronously observed semimartingales with jumps, *Scandinavian Journal of Statistics*, **41**, 460–481.
- Kunitomo, N. and Sato, S. (2013) Separating information maximum likelihood estimation of realized volatility and covariance with micro-market noise, *North American Journal of Economics and Finance*, **26**, 282–309.
- 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. (2015) Flat-top realized kernel estimation of quadratic covariation with non-synchronous and noisy asset prices, *Journal of Business & Economic Statistics*, doi:10.1080/07350015.2015.1005622.
- Zhang, L. (2006) Efficient estimation of stochastic volatility using noisy observations: a multi-scale approach, *Bernoulli*, **12**, no.6, 1019–1043.
- 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](#), [setData](#), [hyavar](#), [lmm](#)

### 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)

# Example. Epps effect for the realized covariance estimator

## Set a model
drift <- c(0,0)

sigma1 <- 1
sigma2 <- 1
rho <- 0.5

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"))

## Generate a path of the latent process
set.seed(116)

## We regard the unit interval as 6.5 hours and generate the path on it
## with the step size equal to 2 seconds

yuima.samp <- setSampling(Terminal = 1, n = 11700)
yuima <- setYuima(model = model, sampling = yuima.samp)
yuima <- simulate(yuima)

## We extract nonsynchronous observations from the path generated above
## by Poisson random sampling with the average duration equal to 10 seconds

psample <- poisson.random.sampling(yuima, rate = c(1/5,1/5), n = 11700)

## Hayashi-Yoshida estimator consistently estimates the true correlation
cce(psample)$cormat[1,2]

## If we synchronize the observation data on some regular grid
## by previous-tick interpolations and compute the correlation
## by therealized covariance based on such synchronized observations,
## we underestimate the true correlation (known as the Epps effect).
## This is illustrated by the following examples.

## Synchronization on the grid with 5 seconds steps
suppressWarnings(s1 <- cce(subsampling(psample, sampling = setSampling(n = 4680)))$cormat[1,2])
s1

## Synchronization on the grid with 10 seconds steps
suppressWarnings(s2 <- cce(subsampling(psample, sampling = setSampling(n = 2340)))$cormat[1,2])
s2

```

```

## Synchronization on the grid with 20 seconds steps
suppressWarnings(s3 <- cce(subsampling(psample, sampling = setSampling(n = 1170)))$cormat[1,2])
s3

## Synchronization on the grid with 30 seconds steps
suppressWarnings(s4 <- cce(subsampling(psample, sampling = setSampling(n = 780)))$cormat[1,2])
s4

## Synchronization on the grid with 1 minute steps
suppressWarnings(s5 <- cce(subsampling(psample, sampling = setSampling(n = 390)))$cormat[1,2])
s5

plot(zoo(c(s1,s2,s3,s4,s5),c(5,10,20,30,60)),type="b",xlab="seconds",ylab="correlation",
main = "Epps effect for the realized covariance")

# Example. Non-synchronous and noisy observations of a correlated bivariate Brownian motion

## Generate noisy observations from the model used in the previous example
Omega <- 0.005*matrix(c(1,rho,rho,1),2,2) # covariance matrix of noise
noisy.psample <- noisy.sampling(psample,var.adj=Omega)
plot(noisy.psample)

## Hayashi-Yoshida estimator: inconsistent
cce(noisy.psample)$covmat

## Pre-averaged Hayashi-Yoshida estimator: consistent
cce(noisy.psample,method="PHY")$covmat

## Generalized multiscale estimator: consistent
cce(noisy.psample,method="GME")$covmat

## Multivariate realized kernel: consistent
cce(noisy.psample,method="RK")$covmat

## Nonparametric QMLE: consistent
cce(noisy.psample,method="QMLE")$covmat

```

---

cogarch.est.-class	<i>Class for Generalized Method of Moments Estimation for COGA-RCH(p,q) model</i>
--------------------	---

---

## Description

The `cogarch.est` class is a class of the **yuima** package that contains estimated parameters obtained by the function `gmm` or `qmle`.

**Slots**

`yuima`: is an object of of [yuima-class](#).

`objFun`: is an object of class character that indicates the objective function used in the minimization problem. See the documentation of the function [gmm](#) or [qml](#) for more details.

`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**

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

**Author(s)**

The YUIMA Project Team

---

`cogarch.est.incr-class`

*Class for Estimation of COGARCH( $p,q$ ) model with underlying increments*

---

**Description**

The `cogarch.est.incr` class is a class of the **yuima** package that extends the [cogarch.est-class](#) and is filled by the function [gmm](#) or [qml](#).

**Slots**

`Incr.Lev`: is an object of class [zoo](#) that contains the estimated increments of the noise obtained using [cogarchNoise](#).

`yuima`: is an object of of [yuima-class](#).

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

`objFun`: is an object of class character that indicates the objective function used in the minimization problem. See the documentation of the function [gmm](#) or [qml](#) for more details.

`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

**simulate** simulation method. For more information see [simulate](#).  
**plot** Plot method for estimated increment of the noise.  
**Methods mle** All methods for mle-class are available.

### Author(s)

The YUIMA Project Team

---

cogarch.info-class      *Class for information about CoGarch(p,q)*

---

### Description

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

### Slots

**p**: Number of autoregressive coefficients in the variance process.  
**q**: Number of moving average coefficients in the variance process.  
**ar.par**: Label of autoregressive coefficients.  
**ma.par**: Label of moving average coefficients.  
**loc.par**: Label of location coefficient in the variance process.  
**Cogarch.var**: Label of the observed process.  
**V.var**: Label of the variance process.  
**Latent.var**: Label of the latent process in the state representation of the variance.  
**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.  
**measure**: Levy measure for jump and quadratic part.  
**measure.type**: Type specification for Levy measure.

### Note

The cogarch.info-class object cannot be directly specified by the user but it is built when the [yuima.cogarch-class](#) object is constructed via [setCogarch](#).

### Author(s)

The YUIMA Project Team

---

 cogarchNoise
 

---

*Estimation for the underlying Levy in a COGARCH(p,q) model*


---

**Description**

Retrieve the increment of the underlying Levy for the COGARCH(p,q) process

**Usage**

```
cogarchNoise(yuima, data=NULL, param, mu=1)
```

**Arguments**

yuima	a yuima object or an object of <code>yuima.cogarch-class</code> .
data	an object of class <code>yuima.data-class</code> contains the observations available at uniformly spaced time. If data=NULL, the default, the cogarchNoise uses the data in an object of <code>yuima.data-class</code> .
param	list of parameters for the COGARCH(p,q).
mu	a numeric object that contains the value of the second moments of the levy measure.

**Value**

incr.Levy	a numeric object contains the estimated increments.
model	an object of class yuima containing the state, the variance and the cogarch process.

**Note**

The function cogarchNoise assumes the underlying Levy process is centered in zero.

The function gmm uses the function cogarchNoise for estimation of underlying Levy in the COGARCH(p,q) model.

**Author(s)**

The YUIMA Project Team

**References**

Chadraa. (2009) Statistical Modelling with COGARCH(P,Q) Processes, *PhD Thesis*.

**Examples**

```
# Insert here some examples
```



---

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 <a href="#">qmle</a> 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 <- qmlL(yuima, start=list(theta1.1=0.3, theta1.2=0.5), t=1.5,
lower=low, method="L-BFGS-B")
tmp1
tmp2 <- qmlR(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)

```

---

Diagnostic.Carma

*Diagnostic Carma model*


---

### Description

This function verifies if the condition of stationarity is satisfied.

**Usage**

```
Diagnostic.Carma(carma)
```

**Arguments**

carma            An object of class `yuima.qmle-class` where the slot `model` is a carma process.

**Value**

Logical variable. If TRUE, Carma is stationary.

**Author(s)**

YUIMA TEAM

**Examples**

```
mod1 <- setCarma(p = 2, q = 1, scale.par = "sig",
  Carma.var = "y")

param1 <- list(a1 = 1.39631, a2 = 0.05029, b0 = 1,
  b1 = 1, sig = 1)
samp1 <- setSampling(Terminal = 100, n = 200)

set.seed(123)

sim1 <- simulate(mod1, true.parameter = param1,
  sampling = samp1)

est1 <- qmle(sim1, start = param1)

Diagnostic.Carma(est1)
```

---

Diagnostic.Cogarch     *Function for checking the statistical properties of the COGARCH(p,q) model*

---

**Description**

The function check the statistical properties of the COGARCH(p,q) model. We verify if the process has a strict positive stationary variance model.

**Usage**

```
Diagnostic.Cogarch(yuima.cogarch, param = list(), matrixS = NULL, mu = 1, display = TRUE)
```

**Arguments**

yuima.cogarch	an object of class yuima.cogarch, yuima or a class cogarch.gmm-class
param	a list containing the values of the parameters
matrixS	a Square matrix.
mu	first moment of the Levy measure.
display	a logical variable, if TRUE the function displays the result in the console.

**Value**

The function returns a List with entries:

meanVarianceProc	Unconditional Stationary mean of the variance process.
meanStateVariable	Unconditional Stationary mean of the state process.
stationary	If TRUE, the COGARCH(p,q) has stationary variance.
positivity	If TRUE, the variance process is strictly positive.

**Author(s)**

YUIMA Project Team

**Examples**

```
## Not run:
# Definition of the COGARCH(1,1) process driven by a Variance Gamma noise:
param.VG <- list(a1 = 0.038, b1 = 0.053,
                a0 = 0.04/0.053, lambda = 1, alpha = sqrt(2), beta = 0, mu = 0,
                x01 = 50.33)

cog.VG <- setCogarch(p = 1, q = 1, work = FALSE,
                    measure=list(df="rvgamma(z, lambda, alpha, beta, mu)"),
                    measure.type = "code",
                    Cogarch.var = "y",
                    V.var = "v", Latent.var="x",
                    XinExpr=TRUE)

# Verify the stationarity and the positivity of the variance process

test <- Diagnostic.Cogarch(cog.VG,param=param.VG)
show(test)

# Simulate a sample path

set.seed(210)

Term=800
num=24000
```

```

samp.VG <- setSampling(Terminal=Term, n=num)

sim.VG <- simulate(cog.VG,
                  true.parameter=param.VG,
                  sampling=samp.VG,
                  method="euler")

plot(sim.VG)

# Estimate the model

res.VG <- gmm(sim.VG, start = param.VG, Est.Incr = "IncrPar")

summary(res.VG)

# Check if the estimated COGARCH(1,1) has a positive and stationary variance

test1<-Diagnostic.Cogarch(res.VG)
show(test1)

# Simulate a COGARCH sample path using the estimated COGARCH(1,1)
# and the recovered increments of underlying Variance Gamma Noise

esttraj<-simulate(res.VG)
plot(esttraj)

## End(Not run)

```

---

gmm

---

*Method of Moments for COGARCH(P,Q).*


---

## Description

The function returns the estimated parameters of a COGARCH(P,Q) model. The parameters are obtained by matching theoretical vs empirical autocorrelation function. The theoretical autocorrelation function is computed according the methodology developed in Chadraa (2009).

## Usage

```

gmm(yuima, data = NULL, start,
    method="BFGS", fixed = list(), lower, upper, lag.max = NULL,
    equally.spaced = FALSE, aggregation=TRUE, Est.Incr = "NoIncr", objFun = "L2")

```

## Arguments

yuima	a yuima object or an object of <a href="#">yuima.cogarch-class</a> .
data	an object of class <a href="#">yuima.data-class</a> contains the observations available at uniformly spaced time. If data=NULL, the default, the function uses the data in an object of <a href="#">yuima-class</a> .

start	a list containing the starting values for the optimization routine.
method	a string indicating one of the methods available in <code>optim</code> .
fixed	a list of fixed parameters in optimization routine.
lower	a named list for specifying lower bounds of parameters.
upper	a named list for specifying upper bounds of parameters.
lag.max	maximum lag at which to calculate the theoretical and empirical acf. Default is $\sqrt{N}$ where N is the number of observation.
equally.spaced	Logical variable. If <code>equally.spaced = TRUE.</code> , the function use the returns of COGARCH(P,Q) evaluated at unitary length for the computation of the empirical autocorrelations. If <code>equally.spaced = FALSE</code> , the increments are evaluated on the interval with frequency specified in an object of class <code>yuima.data-class</code> that contains the observed time series.
aggregation	If <code>aggregation=TRUE</code> , before the estimation of the levy parameters we aggregate the estimated increments
Est.Incr	a string variable, If <code>Est.Incr = "NoIncr"</code> , default value, gmm returns an object of class <code>cogarch.est-class</code> that contains the COGARCH parameters. If <code>Est.Incr = "Incr"</code> or <code>Est.Incr = "IncrPar"</code> the output is an object of class <code>cogarch.est.incr-class</code> . In the first case the object contains the increments of underlying noise while in the second case also the estimated parameter of levy measure.
objFun	a string variable that indentifies the objective function in the optimization step. <code>objFun = "L2"</code> , default value, the objective function is a quadratic form where the weighting Matrix is the identity one. <code>objFun = "L2CUE"</code> the weighting matrix is estimated using Continuously Updating GMM (L2CUE). <code>objFun = "L1"</code> , the objective function is the mean absolute error. In the last case the standard error for estimators are not available.

### Details

The routine is based on three steps: estimation of the COGARCH parameters, recovering the increments of the underlying Levy process and estimation of the levy measure parameters. The last two steps are available on request by the user.

### Value

The function returns a list with the same components of the object obtained when the function `optim` is used.

### Author(s)

The YUIMA Project Team.

### References

Chadraa, E. (2009) Statistical Modeling with COGARCH(P,Q) Processes. Phd Thesis



## Examples

```
## Not run:
# Example COGARCH(1,1): the parameters are the same used in Haugh et al. 2005. In this case
# we assume the underlying noise is a symmetric variance gamma.
# As first step we define the COGARCH(1,1) in yuima:

mod1 <- setCogarch(p = 1, q = 1, work = FALSE,
                  measure=list(df="rbgamma(z,1,sqrt(2)),1,sqrt(2)"),
                  measure.type = "code", Cogarch.var = "y",
                  V.var = "v", Latent.var="x",XinExpr=TRUE)

param <- list(a1 = 0.038, b1 = 0.053,
             a0 = 0.04/0.053, x01 = 20)

# We generate a trajectory
samp <- setSampling(Terminal=10000, n=100000)
set.seed(210)
sim1 <- simulate(mod1, sampling = samp, true.parameter = param)

# We estimate the model

res1 <- gmm(yuima = sim1, start = param)

summary(res1)

## End(Not run)
```

---

hyavar

*Asymptotic Variance Estimator for the Hayashi-Yoshida estimator*


---

## Description

This function estimates the asymptotic variances of covariance and correlation estimates by the Hayashi-Yoshida estimator.

## Usage

```
hyavar(yuima, bw, nonneg = TRUE, psd = TRUE)
```

## Arguments

yuima	an object of yuima-class or yuima.data-class.
bw	a positive number or a numeric matrix. If it is a matrix, each component indicate the bandwidth parameter for the kernel estimators used to estimate the asymptotic variance of the corresponding component (necessary only for off-diagonal components). If it is a number, it is converted to a matrix as <code>matrix(bw, d, d)</code> , where $d = \dim(x)$ . The default value is the matrix whose $(i, j)$ -th component is given by $\min(n_i, n_j)^{0.45}$ , where $n_i$ denotes the number of the observations for the $i$ -th component of the data.

nonneg	logical. If TRUE, the asymptotic variance estimates for correlations are always ensured to be non-negative. See ‘Details’.
psd	passed to <a href="#">cce</a> .

### Details

The precise description of the method used to estimate the asymptotic variances is as follows. For diagonal components, they are estimated by the realized quarticity multiplied by  $2/3$ . Its theoretical validity is ensured by Hayashi et al. (2011), for example. For off-diagonal components, they are estimated by the naive kernel approach described in Section 8.2 of Hayashi and Yoshida (2011). Note that the asymptotic covariance between a diagonal component and another component, which is necessary to evaluate the asymptotic variances of correlation estimates, is not provided in Hayashi and Yoshida (2011), but it can be derived in a similar manner to that paper.

If `nonneg` is TRUE, negative values of the asymptotic variances of correlations are avoided in the following way. The computed asymptotic variance-covariance matrix of the vector  $(HY_{ii}, HY_{ij}, HY_{jj})$  is converted to its spectral absolute value. Here,  $HY_{ij}$  denotes the Hayashi-Yoshida estimator for the  $(i, j)$ -th component.

The function also returns the covariance and correlation matrices calculated by the Hayashi-Yoshida estimator (using [cce](#)).

### Value

A list with components:

<code>covmat</code>	the estimated covariance matrix
<code>cormat</code>	the estimated correlation matrix
<code>avar.cov</code>	the estimated asymptotic variances for covariances
<code>avar.cor</code>	the estimated asymptotic variances for correlations

### Note

Construction of kernel-type estimators for off-diagonal components is implemented after pseudo-aggregation described in Bibinger (2011).

### Author(s)

The YUIMA Project Team

### References

- Barndorff-Nielsen, O. E. and Shephard, N. (2004) Econometric analysis of realized covariation: High frequency based covariance, regression, and correlation in financial economics, *Econometrica*, **72**, no. 3, 885–925.
- Bibinger, M. (2011) Asymptotics of Asynchronicity, technical report, Available at arXiv:1106.4222.
- Hayashi, T., Jacod, J. and Yoshida, N. (2011) Irregular sampling and central limit theorems for power variations: The continuous case, *Annales de l’Institut Henri Poincaré - Probabilités et Statistiques*, **47**, no. 4, 1197–1218.

Hayashi, T. and Yoshida, N. (2011) Nonsynchronous covariation process and limit theorems, *Stochastic processes and their applications*, **121**, 2416–2454.

### See Also

[setData](#), [cce](#)

### 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)

## Constructing a 95% confidence interval for the quadratic covariation from psample
result <- hyavar(psample)
thetahat <- result$covmat[1,2] # estimate of the quadratic covariation
se <- sqrt(result$avar.cov[1,2]) # estimated standard error
c(lower = thetahat + qnorm(0.025) * se, upper = thetahat + qnorm(0.975) * se)

## 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)
}

# contained in the constructed confidence interval
cc.theta(T = 1, diff.coef.1, diff.coef.2, cor.rho)$value

# 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" values of the covariance and correlation.
result.full <- cce(yuima)
(cov.true <- result.full$covmat[1,2]) # covariance
(cor.true <- result.full$cormat[1,2]) # correlation

## 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)

## Constructing 95% confidence intervals for the covariation from psample
result <- hyavar(psample)
cov.est <- result$covmat[1,2] # estimate of covariance
cor.est <- result$cormat[1,2] # estimate of correlation
se.cov <- sqrt(result$avar.cov[1,2]) # estimated standard error of covariance
se.cor <- sqrt(result$avar.cor[1,2]) # estimated standard error of correlation

## 95% confidence interval for covariance
c(lower = cov.est + qnorm(0.025) * se.cov,
  upper = cov.est + qnorm(0.975) * se.cov) # contains cov.true

## 95% confidence interval for correlation
c(lower = cor.est + qnorm(0.025) * se.cor,
  upper = cor.est + qnorm(0.975) * se.cor) # contains cor.true

## We can also use the Fisher z transformation to construct a
## 95% confidence interval for correlation
## It often improves the finite sample behavior of the asymptotic
## theory (cf. Section 4.2.3 of Barndorff-Nielsen and Shephard (2004))
z <- atanh(cor.est) # the Fisher z transformation of the estimated correlation
se.z <- se.cor/(1 - cor.est^2) # standard error for z (calculated by the delta method)
## 95% confidence interval for correlation via the Fisher z transformation
c(lower = tanh(z + qnorm(0.025) * se.z), upper = tanh(z + qnorm(0.975) * se.z))

```

**Description**

Calculate the information criteria BIC, Quasi-BIC (QBIC) and CIC for the stochastic differential equation.

**Usage**

```
IC(yuima, data, start, lower, upper, joint = FALSE, rcpp = FALSE, ...)
```

**Arguments**

yuima	a yuima object.
data	the data to be used.
start	a named list of the initial values of the parameters for optimization.
lower	a named list for specifying lower bounds of the parameters.
upper	a named list for specifying upper bounds of the parameters.
joint	perform joint parameter estimation or two stage parameter estimation? (default joint=FALSE) If two.step=TRUE, this argument is not used.
rcpp	use C++ code? (default rcpp=FALSE)
...	

**Details**

Please see specifications in <https://sites.google.com/site/shoichieguchi90en/specification>

**Value**

par	the estimators of the parameters.
BIC	a value of BIC.
QBIC	a value of QBIC.
CIC	a value of CIC.

**Note**

The function IC uses the function [qmle](#) with method="L-BFGS-B" internally.

**Author(s)**

The YUIMA Project Team

Contacts: Shoichi Eguchi <s-eguchi@math.kyushu-u.ac.jp>

## References

## AIC, BIC

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In Second International Symposium on Information Theory (Tsahkadsor, 1971), 267-281. [http://link.springer.com/chapter/10.1007/978-1-4612-1694-0\\_15](http://link.springer.com/chapter/10.1007/978-1-4612-1694-0_15)

Schwarz, G. (1978). Estimating the dimension of a model. The Annals of Statistics, 6(2), 461-464. <http://projecteuclid.org/euclid.aos/1176344136>

## BIC, Quasi-BIC

Eguchi, S. and Masuda, H. (2016). Schwarz type model comparison for LAQ models. [arXiv:1606.01627v2](https://arxiv.org/abs/1606.01627v2).

## CIC

Uchida, M. (2010). Contrast-based information criterion for ergodic diffusion processes from discrete observations. Annals of the Institute of Statistical Mathematics, 62(1), 161-187. <http://link.springer.com/article/10.1007/s009-0245-1>

## Examples

```
### Ex.1
set.seed(123)

N <- 1000 # number of data
h <- N^(-2/3) # sampling stepsize
Ter <- N*h # terminal sampling time

## Data generate (dXt = -Xt*dt + exp((-2*cos(Xt) + 1)/2)*dWt)
mod <- setModel(drift="theta21*x", diffusion="exp((theta11*cos(x)+theta12)/2)")
samp <- setSampling(Terminal=Ter, n = N)
yuima <- setYuima(model=mod, sampling=setSampling(Terminal=Ter, n=50*N))
simu.yuima <- simulate(yuima, xinit=1, true.parameter=list(theta11=-2, theta12=1,
  theta21=-1), subsampling=samp)

Xt <- NULL
for(i in 1:(N+1)){
  Xt <- c(Xt, simu.yuima@data@original.data[50*(i-1)+1])
}

## Parameter settings
para.init <- list(theta11=runif(1,max=-1.5,min=-2.5), theta12=runif(1,max=1.5,min=0.5),
  theta21=runif(1,max=-0.5,min=-1.5))
para.low <- list(theta11=-7, theta12=-4, theta21=-6)
para.upp <- list(theta11=3, theta12=6, theta21=4)

## Ex.1.1 (dXt = (theta21*x)*dt + exp((theta11*cos(x)+theta12)/2)*dWt)
mod1 <- setModel(drift="theta21*x", diffusion="exp((theta11*cos(x)+theta12)/2)")
samp1 <- setSampling(Terminal=Ter, n = N)
yuima1 <- setYuima(model=mod1, sampling=samp1)
ic1 <- IC(yuima1, data=Xt, start=para.init, upper=para.upp, lower=para.low, rcpp=TRUE)
ic1

## Ex.1.2 (dXt = (theta21*x)*dt + exp(theta11*cos(x)/2)*dWt)
```

```

mod2 <- setModel(drift="theta21*x", diffusion="exp(theta11*cos(x)/2)")
samp2 <- setSampling(Terminal=Ter, n = N)
yuima2 <- setYuima(model=mod2, sampling=samp2)
ic2 <- IC(yuima2, data=Xt, start=para.init, upper=para.upp, lower=para.low, rcpp=TRUE)
ic2

## Not run:
### Ex.2 (multidimansion case)
set.seed(123)

N <- 3000 # number of data
h <- N^(-2/3) # sampling stepsize
Ter <- N*h # terminal sampling time

## Data generate
diff.coef.matrix <- matrix(c("beta1+1", "beta3*x1", "-beta2*x1", "beta4+1"), 2, 2)
drif.coef.vec <- c("alpha1*x1", "alpha2*x2")
mod <- setModel(drift = drif.coef.vec, diffusion = diff.coef.matrix,
               state.variable = c("x1", "x2"), solve.variable = c("x1", "x2"))
samp <- setSampling(Terminal = Ter, n = N)
yuima <- setYuima(model = mod, sampling = setSampling(Terminal = Ter, n = 50*N))
simu.yuima <- simulate(yuima, xinit = c(1,1), true.parameter = list(alpha1=-2, alpha2=-1,
                          beta1=1, beta2=1, beta3=1, beta4=2), subsampling = samp)
Xt <- matrix(0,(N+1),2)
for(i in 1:(N+1)){
  Xt[i,] <- simu.yuima@data@original.data[50*(i-1)+1,]
}

## Parameter settings
para.init <- list(alpha1 = runif(1,min=-3,max=-1), alpha2 = runif(1,min=-2,max=0),
                 alpha3 = runif(1,min=-1,max=1), beta1 = runif(1,min=0,max=2),
                 beta2 = runif(1,min=0,max=2), beta3 = runif(1,min=0,max=2),
                 beta4 = runif(1,min=1,max=3))
para.low <- list(alpha1 = -5, alpha2 = -5, alpha3 = -5, beta1 = 0, beta2 = 0, beta3 = 0, beta4 = 0)
para.upp <- list(alpha1 = 5, alpha2 = 5, alpha3 = 5, beta1 = 5, beta2 = 5, beta3 = 5, beta4 = 5)

## Ex.2.1
diff.coef.matrix1 <- matrix(c("beta1+1", "beta3*x1", "-beta2*x1", "beta4+1"), 2, 2)
drif.coef.vec1 <- c("alpha1*x1", "alpha2*x2+alpha3")
mod1 <- setModel(drift = drif.coef.vec1, diffusion = diff.coef.matrix1,
               state.variable = c("x1", "x2"), solve.variable = c("x1", "x2"))
samp1 <- setSampling(Terminal=Ter, n = N)
yuima1 <- setYuima(model=mod1, sampling=samp1)
ic1 <- IC(yuima1, data=Xt, start=para.init, upper=para.upp, lower=para.low, rcpp=TRUE)
ic1

## Ex.2.2
diff.coef.matrix2 <- matrix(c("beta1+1", "beta3*x1", "-beta2*x1", "beta4+1"), 2, 2)
drif.coef.vec2 <- c("alpha1*x1", "alpha2*x2")
mod2 <- setModel(drift = drif.coef.vec2, diffusion = diff.coef.matrix2,
               state.variable = c("x1", "x2"), solve.variable = c("x1", "x2"))
samp2 <- setSampling(Terminal=Ter, n = N)
yuima2 <- setYuima(model=mod2, sampling=samp2)

```

```
ic2 <- IC(yuima2, data=Xt, start=para.init, upper=para.upp, lower=para.low, rcpp=TRUE)
ic2

## End(Not run)
```

---

info.Map-class	<i>Class for information about Map/Operators</i>
----------------	--

---

### Description

Auxiliar class for definition of an object of class [yuima.Map](#). see the documentation of [yuima.Map](#) for more details.

---

info.Ppr	<i>Class for information about Point Process</i>
----------	--

---

### Description

Auxiliar class for definition of an object of class [yuima.Ppr](#) and [yuima.Hawkes](#). see the documentation for more details.

---

Integral.sde	<i>Class for the mathematical description of integral of a stochastic process</i>
--------------	---

---

### Description

Auxiliar class for definition of an object of class [yuima.Integral](#). see the documentation of [yuima.Integral](#) for more details.

---

Integrand	<i>Class for the mathematical description of integral of a stochastic process</i>
-----------	---

---

### Description

Auxiliar class for definition of an object of class [yuima.Integral](#). see the documentation of [yuima.Integral](#) for more details.



---

Intensity.PPR                      *Intensity Process for the Point Process Regression Model*

---

**Description**

This function returns the intensity process of a Point Process Regression Model

**Usage**

```
Intensity.PPR(yuimaPPr, param)
```

**Arguments**

yuimaPPr	An object of class <code>yuima.ppr</code>
param	Model parameters

**Value**

On object of class `yuima.data`

**Author(s)**

YUIMA TEAM

**Examples**

```
#INSERT HERE AN EXAMPLE
```

---

lambdaFromData                      *Intensity of a Point Process Regression Model*

---

**Description**

This function returns the intensity process of a Ppr model when covariates and counting processes are observed on discrete time

**Usage**

```
lambdaFromData(yuimaPPr, PprData = NULL, parLambda = list())
```

**Arguments**

yuimaPPr	Mathematical Description of Ppr model
PprData	Observed data
parLambda	Values of intensity parameters

**Details**

...

**Value**

...

**Note**

...

**Author(s)**

YUIMA TEAM

**References**

...

**See Also**

...

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 <code>optim</code> method. See Examples.

**Details**

lasso behaves more likely the standard `qmls` 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)
```

**Description**

Methods for yuima.law

**Usage**

```

rand(object, n, param, ...)
dens(object, x, param, log = FALSE, ...)
cdf(object, q, param, ...)
quant(object, p, param, ...)

```

**Arguments**

object	...
n	...
param	...
...	...
x	...
log	...
q	...
p	...

**Value**

Methods for an object of `yuima.law`-class

**Note**

Insert additional info

**Author(s)**

YUIMA TEAM

---

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 asymptotic normality of theta estimators. this program is limited to 1-dimension-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))

## theoretical 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 = (ci || ccor),
     grid, psd = TRUE, plot = ci, ccor = ci, ci = FALSE, alpha = 0.01,
     fisher = TRUE, bw)
```

**Arguments**

x	an object of <a href="#">yuima-class</a> or <a href="#">yuima.data-class</a> .
verbose	whether llag returns matrices or not. The default is FALSE.
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. If ci is also TRUE, the pointwise confidence intervals (under the null hypothesis that the corresponding correlation is zero) are also plotted. The default is FALSE.
ccor	logical. If TRUE, the estimated cross-correlation functions are returned. This argument is ignored if verbose is FALSE. The default is FALSE.
ci	logical. If TRUE, (pointwise) confidence intervals of the estimated cross-correlation functions and p-values for the significance of the correlations at the estimated lead-lag parameters are calculated. Note that the confidence intervals are only plotted when plot=TRUE.
alpha	a positive number indicating the significance level of the confidence intervals for the cross-correlation functions.
fisher	logical. If TRUE, the p-values and the confidence intervals for the cross-correlation functions is evaluated after applying the Fisher z transformation. This argument is only meaningful if pval = "corr".
bw	bandwidth parameter to compute the asymptotic variances. See 'Details' and <a href="#">hyavar</a> for details.

## Details

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

Let  $X^{i_{t_{i_0}}}, X^{i_{t_{i_1}}}, \dots, X^{i_{t_{i_n(i)}}}$  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  $U_{ij}(\theta)$  of the observations  $X^i$  and  $X^j$  ( $i < j$ ) is defined by the same way as in Hoffmann et al. (2013), which corresponds to their cross-covariance function. The lead-lag parameter  $\theta_{ij}$  is defined as a maximizer of  $|U_{ij}(\theta)|$ .  $U_{ij}(\theta)$  is evaluated on a finite grid  $G_{ij}$  defined below. Thus  $\theta_{ij}$  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  $C := (U_{kl}(\theta))_{k,l \in i,j}$  is converted to  $(C\%*\%C)^{(1/2)}$  for ensuring the positive semi-definiteness, and  $U_{ij}(\theta)$  is redefined as the (1, 2)-component of the converted  $C$ . Here,  $U_{kk}(\theta)$  is set to the realized volatility of  $X^k$ . In this case  $\theta_{ij}$  is given as a maximizer of the cross-correlation functions.

The grid  $G_{ij}$  is defined as follows. First, if `grid` is missing,  $G_{ij}$  is given by

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

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`),  $a = -(t_{j_n(j)} - t_{i_0})$  (resp.  $b = t_{i_n(i)} - t_{j_0}$ ) is used, while if the corresponding component of `division` is FALSE,  $N = \text{round}(2\max(n(i), n(j))) + 1$  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,  $G_{ij}$  is given by `grid`. If `grid` is a list of numeric vectors,  $G_{ij}$  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  $(\theta_{ij})_{i,j=1,\dots,d}$ . If `verbose` is TRUE, the covariance matrix  $(U_{ij}(\theta_{ij}))_{i,j=1,\dots,d}$  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  $U_{ij}(\theta)/\text{sqrt}(U_{ii}(\theta) * U_{jj}(\theta))$  of class `zoo` indexed by  $G_{ij}$ .

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

If `ci` is TRUE, the asymptotic variances of the cross-correlations are calculated at each point of the grid by using the naive kernel approach described in Section 8.2 of Hayashi and Yoshida (2011). The implementation is the same as that of `hyavar` and more detailed description is found there.

## Value

If `verbose` is FALSE, a skew-symmetric matrix corresponding to the estimated lead-lag parameters is returned. Otherwise, an object of class `"yuima.llag"`, which is a list with the following components, is returned:

<code>lagcce</code>	a skew-symmetric matrix corresponding to the estimated lead-lag parameters.
<code>covmat</code>	a covariance matrix corresponding to the estimated lead-lag parameters.
<code>cormat</code>	a correlation matrix corresponding to the estimated lead-lag parameters.
<code>LLR</code>	a matrix consisting of lead-lag ratios. See Huth and Abergel (2014) for details.

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

`p.values` a matrix of p-values for the significance of the correlations corresponding to the estimated lead-lag parameters.

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

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

`avar` a list of computed asymptotic variances of the cross-correlations (if `ci = TRUE`).

### Note

The default `grid` usually contains too many points, so it is better for users to specify this argument in order to reduce the computational time. See 'Examples' below for an example of the specification.

The evaluated p-values should carefully be interpreted because they are calculated based on *point-wise confidence intervals* rather than *simultaneous confidence intervals* (so there would be a multiple testing problem). Evaluation of p-values based on the latter will be implemented in the future extension of this function: Indeed, so far no theory has been developed for this. However, it is conjectured that the error distributions of the estimated cross-correlation functions are asymptotically independent if the grid is not dense too much, so p-values evaluated by this function will still be meaningful as long as sufficiently low significance levels are used.

### Author(s)

The YUIMA Project Team

### References

Hayashi, T. and Yoshida, N. (2011) Nonsynchronous covariation process and limit theorems, *Stochastic processes and their applications*, **121**, 2416–2454.

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

Huth, N. and Abergel, F. (2014) High frequency lead/lag relationships — Empirical facts, *Journal of Empirical Finance*, **26**, 41–58.

### See Also

[cce](#), [hyavar](#), [mllag](#)

### 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, plot=TRUE)

## estimated lead-lag parameter
result

## computing pointwise confidence intervals
llag(psample, ci = TRUE)

## In practice, it is better to specify the grid because the default grid contains too many points.
## Here we give an example for how to specify it.

## We search lead-lag parameters on the interval [-0.1, 0.1] with step size 0.01
G <- seq(-0.1,0.1,by=0.01)
```

```
## lead-lag estimation (with computing confidence intervals)
result <- llag(psample, grid = G, ci = TRUE)

## Since the true lead-lag parameter 0.12 between x1 and x3 is not contained
## in the searching grid G, we see that the corresponding cross-correlation
## does not exceed the confidence interval

## detailed output
## the p-value for the (1,3)-th component is high
result

## Finally, we can examine confidence intervals of other significant levels
## and/or without the Fisher z-transformation via the plot-method defined
## for yuima.llag-class objects as follows
plot(result, alpha = 0.001)
plot(result, fisher = FALSE)

par(mfcol=c(1,1))
```

---

LogSPX

*Five minutes Log SPX prices*

---

## Description

Intraday five minutes Standard and Poor 500 Log-prices data ranging from 09 july 2012 to 01 april 2015.

## Usage

```
data(LogSPX)
```

## Details

The dataset is composed by a list where the element `Data$all0bs` contains the intraday five minutes Standard and Poor cumulative Log-return data computed as  $\text{Log}(P_t) - \text{Log}(P_0)$  and  $P_0$  is the open SPX price at 09 july 2012. `Data$logdayprice` contains daily SPX log prices and. Each day we have the same number of observation and the value is reported in `Data$obsinday`. The original data are freely available from the <http://thebonnotgang.com/>

## Examples

```
data(LogSPX)
```

---

lseBayes	<i>Adaptive Bayes estimator for the parameters in sde model by using LSE functions</i>
----------	--

---

**Description**

Adaptive Bayes estimator for the parameters in a specific type of sde by using LSE functions.

**Usage**

```
lseBayes(yuima, start, prior, lower, upper, method = "mcmc", mcmc = 1000,
rate = 1, algorithm = "randomwalk")
```

**Arguments**

yuima	a 'yuima' object.
start	initial suggestion for parameter values
prior	a list of prior distributions for the parameters specified by 'code'. Currently, dunif(z, min, max), dnorm(z, mean, sd), dbeta(z, shape1, shape2), dgamma(z, shape, rate) are available.
lower	a named list for specifying lower bounds of parameters
upper	a named list for specifying upper bounds of parameters
method	nomcmc requires package cubature
mcmc	number of iteration of Markov chain Monte Carlo method
rate	a thinning parameter. Only the first $n^{\text{rate}}$ observation will be used for inference.
algorithm	Logical value when method = mcmc. If algorithm = "randomwalk" (default), the random-walk Metropolis algorithm will be performed. If algorithm = "MPCN", the Mixed preconditioned Crank-Nicolson algorithm will be performed.

**Details**

lseBayes is always performed by Rcpp code. Calculate the Bayes estimator for stochastic processes by using Least Square Estimate functions. The calculation is performed by the Markov chain Monte Carlo method. Currently, the Random-walk Metropolis algorithm and the Mixed preconditioned Crank-Nicolson algorithm is implemented. In lseBayes, the LSE function for estimating diffusion parameter differs from the LSE function for estimating drift parameter. lseBayes is similar to adaBayes, but lseBayes calculate faster than adaBayes because of LSE functions.

**Value**

vector	a vector of the parameter estimate
--------	------------------------------------

**Note**

algorithm = "nomcmc" is unstable. nomcmc is going to be stopped.

**Author(s)**

Yuto Yoshida with YUIMA project Team

**References**

Yoshida, N. (2011). Polynomial type large deviation inequalities and quasi-likelihood analysis for stochastic differential equations. *Annals of the Institute of Statistical Mathematics*, 63(3), 431-479.

Uchida, M., & Yoshida, N. (2014). Adaptive Bayes type estimators of ergodic diffusion processes from discrete observations. *Statistical Inference for Stochastic Processes*, 17(2), 181-219.

Kamatani, K. (2017). Ergodicity of Markov chain Monte Carlo with reversible proposal. *Journal of Applied Probability*, 54(2).

**Examples**

```
## Not run:
####2-dim model
set.seed(123)

b <- c("-theta1*x1+theta2*sin(x2)+50", "-theta3*x2+theta4*cos(x1)+25")
a <- matrix(c("4+theta5*sin(x1)^2", "1", "1", "2+theta6*sin(x2)^2"), 2, 2)

true = list(theta1 = 0.5, theta2 = 5, theta3 = 0.3,
            theta4 = 5, theta5 = 1, theta6 = 1)
lower = list(theta1=0.1, theta2=0.1, theta3=0,
            theta4=0.1, theta5=0.1, theta6=0.1)
upper = list(theta1=1, theta2=10, theta3=0.9,
            theta4=10, theta5=10, theta6=10)
start = list(theta1=runif(1),
            theta2=rnorm(1),
            theta3=rbeta(1,1,1),
            theta4=rnorm(1),
            theta5=rgamma(1,1,1),
            theta6=rexp(1))

yuimamodel <- setModel(drift=b,diffusion=a,state.variable=c("x1", "x2"),solve.variable=c("x1","x2"))
yuimasamp <- setSampling(Terminal=50,n=50*100)
yuima <- setYuima(model = yuimamodel, sampling = yuimasamp)
yuima <- simulate(yuima, xinit = c(100,80),
                true.parameter = true,sampling = yuimasamp)

prior <-
  list(
    theta1=list(measure.type="code",df="dunif(z,0,1)"),
    theta2=list(measure.type="code",df="dnorm(z,0,1)"),
    theta3=list(measure.type="code",df="dbeta(z,1,1)"),
    theta4=list(measure.type="code",df="dgamma(z,1,1)"),
    theta5=list(measure.type="code",df="dnorm(z,0,1)"),
    theta6=list(measure.type="code",df="dnorm(z,0,1)")
  )
```

```

mle <- qmle(yuima, start = start, lower = lower, upper = upper, method = "L-BFGS-B",rcpp=TRUE)
print(mle@coef)
set.seed(123)
bayes1 <- lseBayes(yuima, start=start, prior=prior,
                  method="mcmc",
                  mcmc=1000,lower = lower, upper = upper,algorithm = "randomwalk")

bayes1@coef
set.seed(123)
bayes2 <- lseBayes(yuima, start=start, prior=prior,
                  method="mcmc",
                  mcmc=1000,lower = lower, upper = upper,algorithm = "MpCN")

bayes2@coef

## End(Not run)

```

---

mllag

*Multiple Lead-Lag Detector*


---

## Description

Detecting the lead-lag parameters of discretely observed processes by picking time shifts at which the Hayashi-Yoshida cross-correlation functions exceed thresholds, which are constructed based on the asymptotic theory of Hayashi and Yoshida (2011).

## Usage

```

mllag(x, from = -Inf, to = Inf, division = FALSE, grid, psd = TRUE,
      plot = TRUE, alpha = 0.01, fisher = TRUE, bw)

```

## Arguments

x	an object of <a href="#">yuima-class</a> or <a href="#">yuima.data-class</a> or <a href="#">yuima.llag-class</a> (output of <a href="#">llag</a> ) or <a href="#">yuima.mllag-class</a> (output of this function).
from	passed to <a href="#">llag</a> .
to	passed to <a href="#">llag</a> .
division	passed to <a href="#">llag</a> .
grid	passed to <a href="#">llag</a> .
psd	passed to <a href="#">llag</a> .
plot	logical. If TRUE, the estimated cross-correlation functions and the pointwise confidence intervals (under the null hypothesis that the corresponding correlation is zero) as well as the detected lead-lag parameters are plotted.
alpha	a positive number indicating the significance level of the confidence intervals for the cross-correlation functions.
fisher	logical. If TRUE, the p-values and the confidence intervals for the cross-correlation functions is evaluated after applying the Fisher z transformation.
bw	passed to <a href="#">llag</a> .

## Details

The computation method of cross-correlation functions and confidence intervals is the same as the one used in [llag](#). The exception between this function and [llag](#) is how to detect the lead-lag parameters. While [llag](#) only returns the maximizer of the absolute value of the cross-correlations following the theory of Hoffmann et al. (2013), this function returns all the time shifts at which the cross-correlations exceed (so there is also the possibility that *no* lead-lag is returned). Note that this approach is mathematically debatable because there would be a multiple testing problem (see also 'Note' of [llag](#)), so the interpretation of the result from this function should carefully be addressed. In particular, the significance level  $\alpha$  probably does not give the "correct" level.

## Value

An object of class "yuima.mllag", which is a list with the following elements:

mllagc	a list of data.frame-class objects consisting of lagc (lead-lag parameters), p.value and correlation.
LLR	a matrix consisting of lead-lag ratios. See Huth and Abergel (2014) for details.
ccor	a list of computed cross-correlation functions.
avar	a list of computed asymptotic variances of the cross-correlations (if ci = TRUE).
CI	a list of computed confidence intervals.

## Author(s)

The YUIMA Project Team

## References

- Hayashi, T. and Yoshida, N. (2011) Nonsynchronous covariation process and limit theorems, *Stochastic processes and their applications*, **121**, 2416–2454.
- Hoffmann, M., Rosenbaum, M. and Yoshida, N. (2013) Estimation of the lead-lag parameter from non-synchronous data, *Bernoulli*, **19**, no. 2, 426–461.
- Huth, N. and Abergel, F. (2014) High frequency lead/lag relationships — Empirical facts, *Journal of Empirical Finance*, **26**, 41–58.

## See Also

[llag](#), [hyavar](#)

## Examples

```
# The first example is taken from llag

## 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)

## We search lead-lag parameters on the interval [-0.1, 0.1] with step size 0.01
G <- seq(-0.1,0.1,by=0.01)

## lead-lag estimation by mllag
par(mfcol=c(3,1))
result <- mllag(psample, grid = G)

## Since the lead-lag parameter for the pair(x1, x2) is not contained in G,
## no lead-lag parameter is detected for this pair

par(mfcol=c(1,1))

# The second example is a situation where multiple lead-lag effects exist
set.seed(222)

n <- 3600
Times <- seq(0, 1, by = 1/n)
R1 <- 0.6
R2 <- -0.3

dW1 <- rnorm(n + 10)/sqrt(n)
dW2 <- rnorm(n + 5)/sqrt(n)
dW3 <- rnorm(n)/sqrt(n)

```

```

x <- zoo(diffinv(dW1[-(1:10)] + dW2[1:n]), Times)
y <- zoo(diffinv(R1 * dW1[1:n] + R2 * dW2[-(1:5)] +
               sqrt(1- R1^2 - R2^2) * dW3), Times)

## In this setting, both x and y have a component leading to the other,
## but x's leading component dominates y's one

yuima <- setData(list(x, y))

## Lead-lag estimation by llag
G <- seq(-30/n, 30/n, by = 1/n)
est <- llag(yuima, grid = G, ci = TRUE)

## The shape of the plotted cross-correlation is evidently bimodal,
## so there are likely two lead-lag parameters

## Lead-lag estimation by mllag
mllag(est) # succeeds in detecting two lead-lag parameters

## Next consider a non-synchronous sampling case
psample <- poisson.random.sampling(yuima, n = n, rate = c(0.8, 0.7))

## Lead-lag estimation by mllag
est <- mllag(psample, grid = G)
est # detects too many lead-lag parameters

## Using a lower significant level
mllag(est, alpha = 0.001) # insufficient

## As the plot reveals, one reason is because the grid is too dense
## In fact, this phenomenon can be avoided by using a coarser grid
mllag(psample, grid = seq(-30/n, 30/n, by=5/n)) # succeeds!

```

---

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 <a href="#">qgv</a> .



**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  $X_{t_0}^i, X_{t_1}^i, \dots, X_{t_n}^i$  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

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

where  $\mu_p$  is the p-th absolute moment of the standard normal distribution and  $\Delta X_{t_j}^i = X_{t_j}^i - X_{t_{j-1}}^i$ . 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=TRUE)` is defined as the d-dimensional vector with i-th element equal to

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

where  $r_1^i, \dots, r_{k_i}^i$  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

<ftp://ftp.ncdc.noaa.gov/pub/data/paleo/treering/measurements/northamerica/usa/ca535.rwl>

## 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	<i>Noisy Observation Generator</i>
----------------	------------------------------------

---

## 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 <a href="#">yuima-class</a> or <a href="#">yuima.data-class</a> .
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 <code>as.list(double(dim(x)))</code> .

## Details

This function simulates microstructure noise and adds it to the path of  $x$ . Currently, this function can deal with Kalnina and Linton (2011) type microstructure noise. See 'Examples' below for more details.

**Value**

an object of `yuima.data-class`.

**Author(s)**

The YUIMA Project Team

**References**

Kalnina, I. and Linton, O. (2011) Estimating quadratic variation consistently in the presence of endogenous and diurnal measurement error, *Journal of Econometrics*, **147**, 47–59.

**See Also**

[cce](#), [lmm](#)

**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.psampl2 <- noisy.sampling(psampl, var.adj=Omega, rng="runif", min=-sqrt(3), max=sqrt(3))
plot(noisy.psampl2)

## Contaminate the observation data by centered exponentially distributed noise
## with the variance matrix equal to 1% of the diffusion
noisy.psampl3 <- noisy.sampling(psampl, var.adj=Omega, rng="rexp", rate=1, mean.adj=1)
plot(noisy.psampl3)

## 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.psampl4 <- noisy.sampling(psampl, end.coef=-0.1, n=2340*c(1/3, 1/6))
plot(noisy.psampl4)

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

```

---

param.Integral	<i>Class for the mathematical description of integral of a stochastic process</i>
----------------	---

---

### Description

Auxiliar class for definition of an object of class [yuima.Integral](#). see the documentation of [yuima.Integral](#) for more details.

---

param.Map-class	<i>Class for information about Map/Operators</i>
-----------------	--

---

### Description

Auxiliar class for definition of an object of class [yuima.Map](#). see the documentation of [yuima.Map](#) for more details.

---

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 <a href="#">qml</a> 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 object of class phi.test.

**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 =
"NoIncr", aggregation = TRUE, threshold = NULL, rcpp =
FALSE, ...)
quasilogl(yuima, param, print = FALSE, rcpp = 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 <code>yuima.carma-class</code> or <code>yuima.cogarch-class</code> the qmle returns an object of <code>yuima.carma.qmle-class</code> , <code>cogarch.est.incr-class</code> , <code>cogarch.est-class</code> or object of class <code>mle-class</code> . By default <code>Est.Incr="NoIncr"</code> , alternative values are <code>IncrPar</code> and <code>Incr</code> .
aggregation	If <code>aggregation=TRUE</code> , 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 <code>optim</code> method. See Examples.
rcpp	use C++ code?

**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 value of 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 <code>yuima.carma-class</code> , qmle returns an object <code>yuima.carma.qmle-class</code>
cogarchopt	if the model is an object of <code>yuima.cogarch-class</code> , qmle returns an object of class <code>cogarch.est-class</code> . The estimates are obtained by maximizing the pseudo-loglikelihood function as shown in Iacus et al. (2015)

**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

**References****## Non-ergodic diffusion**

Genon-Catalot, V., & Jacod, J. (1993). On the estimation of the diffusion coefficient for multi-dimensional diffusion processes. In *Annales de l'IHP Probabilités et statistiques*, 29(1), 119-151.

Uchida, M., & Yoshida, N. (2013). Quasi likelihood analysis of volatility and nondegeneracy of statistical random field. *Stochastic Processes and their Applications*, 123(7), 2851-2876.

**## Ergodic diffusion**

Kessler, M. (1997). Estimation of an ergodic diffusion from discrete observations. *Scandinavian Journal of Statistics*, 24(2), 211-229.

**## Jump diffusion**

Shimizu, Y., & Yoshida, N. (2006). Estimation of parameters for diffusion processes with jumps from discrete observations. *Statistical Inference for Stochastic Processes*, 9(3), 227-277.

Ogihara, T., & Yoshida, N. (2011). Quasi-likelihood analysis for the stochastic differential equation with jumps. *Statistical Inference for Stochastic Processes*, 14(3), 189-229.

**## COGARCH**

Iacus S. M., Mercuri L. and Rroji E.(2015) Discrete time approximation of a COGARCH (p, q) model and its estimation. <http://arxiv.org/abs/1511.00253>

**## CARMA**

Iacus S. M., Mercuri L. (2015) Implementation of Levy CARMA model in Yuima package. *Comp. Stat.* (30) 1111-1141. <http://link.springer.com/article/10.1007/s00180-015-0569-7>

**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.q1(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.q1(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)

# 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)

# 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)

# Simulation and Estimation of COGARCH(1,1) with CP driven noise
```

```
# Model parameters
eta<-0.053
b1 <- eta
beta <- 0.04
a0 <- beta/b1
phi<- 0.038
a1 <- phi

# Definition

cog11<-setCogarch(p = 1,q = 1,
  measure = list(intensity = "1",
                 df = list("dnorm(z, 0, 1)")),
  measure.type = "CP",
  XinExpr=TRUE)

# Parameter
paramCP11 <- list(a1 = a1, b1 = b1,
                 a0 = a0, y01 = 50.31)
# Sampling scheme
samp11 <- setSampling(0, 3200, n=64000)

# Simulation
set.seed(125)

SimTime11 <- system.time(
  sim11 <- simulate(object = cog11,
    true.parameter = paramCP11,
    sampling = samp11,
    method="mixed")
)

plot(sim11)

# Estimation

timeComp11<-system.time(
  res11 <- qmle(yuima = sim11,
    start = paramCP11,
    grideq = TRUE,
    method = "Nelder-Mead")
)

timeComp11

unlist(paramCP11)

coef(res11)

# COGARCH(2,2) model driven by CP

cog22 <- setCogarch(p = 2,q = 2,
```

```
measure = list(intensity = "1",
               df = list("dnorm(z, 0, 1)")),
measure.type = "CP",
XinExpr=TRUE)

# Parameter

paramCP22 <- list(a1 = 0.04, a2 = 0.001,
                 b1 = 0.705, b2 = 0.1, a0 = 0.1, y01 = (1 + 2 / 3),
                 y02=0)

# Use diagnostic.cog for checking the stat and positivity

check22 <- Diagnostic.Cogarch(cog22, param = paramCP22)

# Sampling scheme

samp22 <- setSampling(0, 3600, n = 64000)

# Simulation

set.seed(125)
SimTime22 <- system.time(
  sim22 <- simulate(object = cog22,
                   true.parameter = paramCP22,
                   sampling = samp22,
                   method = "Mixed")
)

plot(sim22)

timeComp22 <- system.time(
  res22 <- qmle(yuima = sim22,
              start = paramCP22,
              grideq=TRUE,
              method = "Nelder-Mead")
)

timeComp22

unlist(paramCP22)

coef(res22)

## End(Not run)
```

**Description**

Calculate the Gaussian quasi-likelihood and Gaussian quasi-likelihood estimators of Levy driven SDE.

**Usage**

```
qmleLevy(yuima, start, lower, upper, joint = FALSE, third = FALSE)
```

**Arguments**

yuima	a yuima object.
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.
joint	perform joint estimation or two stage estimation? by default joint=FALSE. If there exists an overlapping parameter, joint=TRUE does not work for the theoretical reason
third	perform third estimation? by default third=FALSE. If there exists an overlapping parameter, third=TRUE does not work for the theoretical reason.

**Details**

This function performs Gaussian quasi-likelihood estimation for Levy driven SDE.

**Value**

first	estimated values of first estimation (scale parameters)
second	estimated values of second estimation (drift parameters)
third	estimated values of third estimation (scale parameters)

**Note**

The function `qmleLevy` uses the function `qmle` internally. It can be applied only for the standardized Levy noise whose moments of any order exist. In present `yuima` package, bilateral gamma (`bgamma`) process, normal inverse Gaussian (NIG) process, variance gamma (VG) process, and normal tempered stable process are such candidates. In the current version, the standardization condition on the driving noise is internally checked only for the one-dimensional noise. The standardization condition for the multivariate noise is given in

<https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbX5dW1hdWVoYXJhMTkyOHxneDo3Z>

or

<https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbX5dW1hdWVoYXJhMTkyOHxneDo3Z>

They also contain more precise explanation of this function.

**Author(s)**

The YUIMA Project Team

Contacts: Yuma Uehara <y-uehara@math.kyushu-u.ac.jp>

**References**

Masuda, H. (2013). Convergence of Gaussian quasi-likelihood random fields for ergodic Levy driven SDE observed at high frequency. *The Annals of Statistics*, 41(3), 1593-1641.

Masuda, H. and Uehara, Y. (2017). On stepwise estimation of Levy driven stochastic differential equation (Japanese) ., *Proc. Inst. Statist. Math.*, accepted.

**Examples**

```
## One-dimensional case
dri<-"-theta0*x" ## set drift
jum<-"theta1/(1+x^2)^(-1/2)" ## set jump
yuima<-setModel(drift = dri
                ,jump.coeff = jum
                ,solve.variable = "x",state.variable = "x"
                ,measure.type = "code"
                ,measure = list(df="rbgamma(z,1,sqrt(2),1,sqrt(2))")) ## set true model

n<-3000
T<-30 ## terminal
hn<-T/n ## stepsize

sam<-setSampling(Terminal = T, n=n) ## set sampling scheme
yuima<-setYuima(model = yuima, sampling = sam) ## model

true<-list(theta0 = 1,theta1 = 2) ## true values
upper<-list(theta0 = 4, theta1 = 4) ## set upper bound
lower<-list(theta0 = 0.5, theta1 = 1) ## set lower bound
set.seed(123)
yuima<-simulate(yuima, xinit = 0, true.parameter = true,sampling = sam) ## generate a path
start<-list(theta0 = runif(1,0.5,4), theta1 = runif(1,1,4)) ## set initial values
qmleLevy(yuima,start=start,lower=lower,upper=upper, joint = TRUE)

## Multi-dimensional case

lambda<-1/2
alpha<-1
beta<-c(0,0)
mu<-c(0,0)
Lambda<-matrix(c(1,0,0,1),2,2) ## set parameters in noise

dri<-c("1-theta0*x1-x2","-theta1*x2")
jum<-matrix(c("x1*theta2+1","0","0","1"),2,2) ## set coefficients

yuima <- setModel(drift=dri,
                 solve.variable=c("x1","x2"),state.variable = c("x1","x2"),
                 jump.coeff=jum, measure.type="code",
                 measure=list(df="rvgamma(z, lambda, alpha, beta, mu, Lambda
```

```

    )")

n<-3000 ## the number of total samples
T<-30 ## terminal
hn<-T/n ## stepsize

sam<-setSampling(Terminal = T, n=n) ## set sampling scheme
yuima<-setYuima(model = yuima, sampling = sam) ## model

true<-list(theta0 = 1,theta1 = 2,theta2 = 3,lambda=lambda, alpha=alpha,
beta=beta,mu=mu, Lambda=Lambda) ## true values
upper<-list(theta0 = 4, theta1 = 4, theta2 = 5) ## set upper bound
lower<-list(theta0 = 0.5, theta1 = 1, theta2 = 1) ## set lower bound
set.seed(123)
yuima<-simulate(yuima, xinit = c(0,0), true.parameter = true,sampling = sam) ## generate a path
plot(yuima)
start<-list(theta0 = runif(1,0.5,4), theta1 = runif(1,1,4),
theta2 = runif(1,1,5)) ## set initial values
qmlLevy(yuima,start=start,lower=lower,upper=upper,joint = FALSE,third=TRUE)

```

---

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 numbers and densities*


---

**Description**

simulate function can use the specific random number generators to generate Levy paths.

**Usage**

```
rGIG(x, lambda, delta, gamma)
dGIG(x, lambda, delta, gamma)
rGH(x, lambda, alpha, beta, delta, mu, Lambda)
dGH(x, lambda, alpha, beta, delta, mu, Lambda)
rIG(x, delta, gamma)
dIG(x, delta, gamma)
rNIG(x, alpha, beta, delta, mu, Lambda)
dNIG(x, alpha, beta, delta, mu, Lambda)
rvgamma(x, lambda, alpha, beta, mu, Lambda)
dvgamma(x, lambda, alpha, beta, mu, Lambda)
rbgamma(x, delta.plus, gamma.plus, delta.minus, gamma.minus)
dbgamma(x, delta.plus, gamma.plus, delta.minus, gamma.minus)
rstable(x, alpha, beta, sigma, gamma)
rpts(x, alpha, a, b)
rnsts(x, alpha, a, b, beta, mu, Lambda)
```

**Arguments**

x	Number of R.Ns to be generated.
a	parameter
b	parameter
delta	parameter
gamma	parameter
mu	parameter
Lambda	parameter
alpha	parameter
lambda	parameter
sigma	parameter

beta	parameter
delta.plus	parameter
gamma.plus	parameter
delta.minus	parameter
gamma.minus	parameter

### Details

GIG (generalized inverse Gaussian): The density function of GIG distribution is expressed as:

$$f(x) = 1/2 * (\gamma/\delta)^{\lambda} * 1/bK_{\lambda}(\gamma * \delta) * x^{\lambda - 1} * \exp(-1/2 * (\delta^2/x + \gamma^2 * x))$$

where  $bK_{\lambda}()$  is the modified Bessel function of the third kind with order  $\lambda$ . The parameters  $\lambda$ ,  $\delta$  and  $\gamma$  vary within the following regions:

$$\delta \geq 0, \gamma > 0, \lambda > 0,$$

$$\delta > 0, \gamma > 0, \lambda = 0,$$

$$\delta > 0, \gamma \geq 0, \lambda < 0.$$

The corresponding Levy measure is given in Eberlein, E., & Hammerstein, E. A. V. (2004) (it contains IG).

GH (generalized hyperbolic): Generalized hyperbolic distribution is defined by the normal mean-variance mixture of generalized inverse Gaussian distribution. The parameters  $\alpha$ ,  $\beta$ ,  $\delta$ ,  $\mu$  express heaviness of tails, degree of asymmetry, scale and location, respectively. Here the parameter  $\Lambda$  is supposed to be symmetric and positive definite with  $\det(\Lambda) = 1$  and the parameters vary within the following region:

$$\delta \geq 0, \alpha > 0, \alpha^2 > \beta^T \Lambda \beta,$$

$$\delta > 0, \alpha > 0, \alpha^2 > \beta^T \Lambda \beta = 0,$$

$$\delta > 0, \alpha \geq 0, \alpha^2 \geq \beta^T \Lambda \beta < 0.$$

The corresponding Levy measure is given in Eberlein, E., & Hammerstein, E. A. V. (2004) (it contains NIG and vgamma).

IG (inverse Gaussian (the element of GIG)):  $\delta$  and  $\gamma$  are positive (the case of  $\gamma = 0$  corresponds to the positive half stable, provided by the "rstable").

NIG (normal inverse Gaussian (the element of GH)): Normal inverse Gaussian distribution is defined by the normal mean-variance mixture of inverse Gaussian distribution. The parameters  $\alpha$ ,  $\beta$ ,  $\delta$  and  $\mu$  express the heaviness of tails, degree of asymmetry, scale and location, respectively. They satisfy the following conditions:  $\Lambda$  is symmetric and positive definite with  $\det(\Lambda) = 1$ ;  $\delta > 0$ ;  $\alpha > 0$  with  $\alpha^2 - \beta^T \Lambda \beta > 0$ .

vgamma (variance gamma (the element of GH)): Variance gamma distribution is defined by the normal mean-variance mixture of gamma distribution. The parameters satisfy the following conditions:  $\Lambda$  is symmetric and positive definite with  $\det(\Lambda) = 1$ ;  $\lambda > 0$ ;  $\alpha > 0$  with  $\alpha^2 - \beta^T \Lambda \beta > 0$ . Especially in the case of  $\beta = 0$  it is variance gamma distribution.

bgamma (bilateral gamma): Bilateral gamma distribution is defined by the difference of independent gamma distributions  $\text{Gamma}(\delta, \gamma)$  and  $\text{Gamma}(\delta, \gamma)$ . Its Levy density  $f(z)$  is given by:  $f(z) = \delta/z * \exp(-\gamma * z) * \text{ind}(z > 0) +$



$\text{delta.minus}/|z| * \exp(-\text{gamma.minus} * |z|) * \text{ind}(z < 0)$ , where the function  $\text{ind}()$  denotes an indicator function.

**stable (stable):** Parameters  $\alpha$ ,  $\beta$ ,  $\sigma$  and  $\gamma$  express stability, degree of skewness, scale and location, respectively. They satisfy the following condition:  $0 < \alpha \leq 2$ ;  $-1 \leq \beta \leq 1$ ;  $\sigma > 0$ ;  $\gamma$  is a real number.

**pts (positive tempered stable):** Positive tempered stable distribution is defined by the tilting of positive stable distribution. The parameters  $\alpha$ ,  $a$  and  $b$  express stability, scale and degree of tilting, respectively. They satisfy the following condition:  $0 < \alpha < 1$ ;  $a > 0$ ;  $b > 0$ . Its Levy density  $f(z)$  is given by:  $f(z) = az^{\alpha-1} \exp(-bz)$ .

**nts (normal tempered stable):** Normal tempered stable distribution is defined by the normal mean-variance mixture of positive tempered stable distribution. The parameters  $\alpha$ ,  $a$ ,  $b$ ,  $\beta$ ,  $\mu$  and  $\Lambda$  express stability, scale, degree of tilting, degree of asymmetry, location and degree of mixture, respectively. They satisfy the following condition:  $\Lambda$  is symmetric and positive definite with  $\det(\Lambda) = 1$ ;  $0 < \alpha < 1$ ;  $a > 0$ ;  $b > 0$ . In one-dimensional case, its Levy density  $f(z)$  is given by:  $f(z) = 2a/(2\pi)^{1/2} * \exp(\beta * z) * (z^2/(2b + \beta^2))^{\alpha-1/2} * bK_{\alpha-1/2}(z\sqrt{2b + \beta^2})$ .

### Value

rXXX	Collection of random numbers or vectors
dXXX	Density dunction

### Note

Some density-plot functions are still missing: as for the non-Gaussian stable densities, one can use, e.g., `stabledist` package. The rejection-acceptance method is used for generating `pts` and `nts`. It should be noted that its acceptance rate decreases at exponential order as  $a$  and  $b$  become larger: specifically, the rate is given by  $\exp(a * \text{gamma}(-\alpha) * b^{\alpha})$

### Author(s)

The YUIMA Project Team

Contacts: Hiroki Masuda <hiroki@math.kyushu-u.ac.jp> and Yuma Uehara <y-uehara@math.kyushu-u.ac.jp>

### References

## rGIG, dGIG, rIG, dIG

Chhikara, R. (1988). *The Inverse Gaussian Distribution: Theory: Methodology, and Applications* (Vol. 95). CRC Press.

Hörmann, W., & Leydold, J. (2014). Generating generalized inverse Gaussian random variates. *Statistics and Computing*, 24(4), 547-557. <http://onlinelibrary.wiley.com/doi/10.1111/1467-9469.00045/abstract>

Jørgensen, B. (2012). *Statistical properties of the generalized inverse Gaussian distribution* (Vol. 9). Springer Science & Business Media. <http://www.springer.com/la/book/9780387906652>

Michael, J. R., Schucany, W. R., & Haas, R. W. (1976). Generating random variates using transformations with multiple roots. *The American Statistician*, 30(2), 88-90. <https://www.jstor.org/stable/2683801>

## rGH, dGH, rNIG, dNIG, rvgamma, dvgamma

Barndorff-Nielsen, O. (1977). Exponentially decreasing distributions for the logarithm of particle size. In Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences (Vol. 353, No. 1674, pp. 401-419). The Royal Society. <http://rspa.royalsocietypublishing.org/content/353/1674/401>

Barndorff-Nielsen, O. E. (1997). Processes of normal inverse Gaussian type. Finance and stochastics, 2(1), 41-68. <http://link.springer.com/article/10.1007/s007800050032>

Eberlein, E. (2001). Application of generalized hyperbolic Lévy motions to finance. In Lévy processes (pp. 319-336). Birkhäuser Boston. [http://link.springer.com/chapter/10.1007/978-1-4612-0197-7\\_14](http://link.springer.com/chapter/10.1007/978-1-4612-0197-7_14)

Eberlein, E., & Hammerstein, E. A. V. (2004). Generalized hyperbolic and inverse Gaussian distributions: limiting cases and approximation of processes. In Seminar on stochastic analysis, random fields and applications IV (pp. 221-264). Birkhäuser Basel. [http://link.springer.com/chapter/10.1007/978-3-0348-7943-9\\_15](http://link.springer.com/chapter/10.1007/978-3-0348-7943-9_15)

Madan, D. B., Carr, P. P., & Chang, E. C. (1998). The variance gamma process and option pricing. European finance review, 2(1), 79-105. <http://rof.oxfordjournals.org/content/2/1/79.short>

## rbgamma, dbgamma

Küchler, U., & Tappe, S. (2008). Bilateral Gamma distributions and processes in financial mathematics. Stochastic Processes and their Applications, 118(2), 261-283. <http://www.sciencedirect.com/science/article/pii/S0304>

Küchler, U., & Tappe, S. (2008). On the shapes of bilateral Gamma densities. Statistics & Probability Letters, 78(15), 2478-2484. <http://www.sciencedirect.com/science/article/pii/S0167715208001521>

## rstable

Chambers, John M., Colin L. Mallows, and B. W. Stuck. (1976) A method for simulating stable random variables, Journal of the american statistical association, 71(354), 340-344. <http://amstat.tandfonline.com/doi/abs/10.1080>

Weron, Rafał. (1996) On the Chambers-Mallows-Stuck method for simulating skewed stable random variables, Statistics & probability letters, 28.2, 165-171. <http://www.sciencedirect.com/science/article/pii/016771529500>

Weron, Rafał. (2010) Correction to: " On the Chambers-Mallows-Stuck Method for Simulating Skewed Stable Random Variables", No. 20761, University Library of Munich, Germany. <https://ideas.repec.org/p/pramprapa>

## rpts

Kawai, R., & Masuda, H. (2011). On simulation of tempered stable random variates. Journal of Computational and Applied Mathematics, 235(8), 2873-2887. <http://www.sciencedirect.com/science/article/pii/S0377042710>

## rnts

Barndorff-Nielsen, O. E., & Shephard, N. (2001). Normal modified stable processes. Aarhus: MaPhySto, Department of Mathematical Sciences, University of Aarhus.

## Examples

```
set.seed(123)

# Ex 1. (One-dimensional standard Cauchy distribution)
# The value of parameters is alpha=1,beta=0,sigma=1,gamma=0.
# Choose the values of x.
x<-10 # the number of r.n
rstable(x,1,0,1,0)

# Ex 2. (One-dimensional Levy distribution)
```

```
# Choose the values of sigma, gamma, x.
# alpha = 0.5, beta=1
x<-10 # the number of r.n
beta <- 1
sigma <- 0.1
gamma <- 0.1
rstable(x,0.5,beta,sigma,gamma)

# Ex 3. (Symmetric bilateral gamma)
# delta=delta.plus=delta.minus, gamma=gamma.plus=gamma.minus.
# Choose the values of delta and gamma and x.
x<-10 # the number of r.n
rbgamma(x,1,1,1,1)

# Ex 4. ((Possibly skewed) variance gamma)
# lambda, alpha, beta, mu
# Choose the values of lambda, alpha, beta, mu and x.
x<-10 # the number of r.n
rvgamma(x,2,1,-0.5,0)

# Ex 5. (One-dimensional normal inverse Gaussian distribution)
# Lambda=1.
# Choose the parameter values and x.
x<-10 # the number of r.n
rNIG(x,1,1,1,1)

# Ex 6. (Multi-dimensional normal inverse Gaussian distribution)
# Choose the parameter values and x.
beta<-c(.5,.5)
mu<-c(0,0)
Lambda<-matrix(c(1,0,0,1),2,2)
x<-10 # the number of r.n
rNIG(x,1,beta,1,mu,Lambda)

# Ex 7. (Positive tempered stable)
# Choose the parameter values and x.
alpha<-0.7
a<-0.2
b<-1
x<-10 # the number of r.n
rpts(x,alpha,a,b)

# Ex 8. (Generalized inverse Gaussian)
# Choose the parameter values and x.
lambda<-0.3
delta<-1
gamma<-0.5
x<-10 # the number of r.n
rGIG(x,lambda,delta,gamma)

# Ex 9. (Multi-variate generalized hyperbolic)
# Choose the parameter values and x.
lambda<-0.4
```

```

alpha<-1
beta<-c(0,0.5)
delta<-1
mu<-c(0,0)
Lambda<-matrix(c(1,0,0,1),2,2)
x<-10 # the number of r.n
rGH(x,lambda,alpha,beta,delta,mu,Lambda)

```

---

setCarma

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


---

### Description

'setCarma' describes the following model:

$$Vt = c0 + \text{sigma} (b0 Xt(0) + \dots + b(q) Xt(q))$$

$$dXt(0) = Xt(1) dt$$

...

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

$$dXt(p-1) = (-a(p) Xt(0) - \dots - a(1) Xt(p-1))dt + (\text{gamma}(0) + \text{gamma}(1) Xt(0) + \dots + \text{gamma}(p) Xt(p-1))dt$$

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

$$\text{gamma}(0) = 1, \text{gamma}(1) = \text{gamma}(2) = \dots = \text{gamma}(p) = 0.$$

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, Cogarch=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.
Cogarch	a logical variable. The default value Cogarch=FALSE implies that the parameters are specified according to Brockwell (2000).
...	Arguments to be passed to 'setCarma', such as the slots of <a href="#">yuima.model-class</a>
	measure Levy measure of jump variables.
	measure.type type specification for Levy measure.
	xinit a vector of expressions identifyng 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:
#  $V_t = c_0 + \alpha_0 X_{0t} + \alpha_1 X_{1t}$ 
#  $dX_{0t} = X_{1t} dt$ 
#  $dX_{1t} = X_{2t} dt$ 
#  $dX_{2t} = (-\beta_3 X_{0t} - \beta_2 X_{1t} - \beta_1 X_{2t}) dt + dW_t$ 
# 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:
#  $V_t = c_0 + \alpha_0 X_{0t} + \alpha_1 X_{1t}$ 
#  $dX_{0t} = X_{1t} dt$ 
#  $dX_{1t} = X_{2t} dt$ 
#  $dX_{2t} = (-\beta_3 X_{0t} - \beta_2 X_{1t} - \beta_1 X_{2t}) dt + (c_0 + \alpha_0 X_{0t}) dW_t$ 
# 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:
#  $V_t = c_0 + \alpha_0 X_{0t} + \alpha_1 X_{1t}$ 
#  $dX_{0t} = X_{1t} dt$ 
#  $dX_{1t} = X_{2t} dt$ 
#  $dX_{2t} = (-\beta_3 X_{0t} - \beta_2 X_{1t} - \beta_1 X_{2t}) dt + dz_t$ 
# 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)
#  $V_t = c_0 + \alpha_0 X_{0t} + \alpha_1 X_{1t}$ 
#  $dX_{0t} = X_{1t} dt$ 
#  $dX_{1t} = X_{2t} dt$ 
#  $dX_{2t} = (-\beta_3 X_{1t} - \beta_2 X_{2t} - \beta_1 X_{3t}) dt + (c_0 + \alpha_0 X_{0t}) dz_t$ 
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

---

setCogarch	<i>Continuous-time GARCH (p,q) process</i>
------------	--

---

**Description**

setCogarch describes the Cogarch(p,q) model introduced in Brockwell et al. (2006):

$$dG_t = \sqrt{V_t} dZ_t$$

$$V_t = a_0 + (a_1 Y_t(1) + \dots + a(p) Y_t(p))$$

$$dY_t(1) = Y_t(2) dt$$

...

$$dY_t(q-1) = Y_t(q) dt$$

$$dY_t(q) = (-b(q) Y_t(1) - \dots - b(1) Y_t(q))dt + (a_0 + (a_1 Y_t(1) + \dots + a(p) Y_t(p))d[Z_t Z_t]^{\{q\}}$$

**Usage**

```
setCogarch(p, q, ar.par = "b", ma.par = "a", loc.par = "a0", Cogarch.var = "g",
  V.var = "v", Latent.var = "y", jump.variable = "z", time.variable = "t",
  measure = NULL, measure.type = NULL, XinExpr = FALSE, startCogarch = 0,
  work = FALSE, ...)
```

**Arguments**

p	a non negative integer that is the number of the moving average coefficients of the Variance process.
q	a non-negative integer that indicates the number of the autoregressive coefficients of the Variance process.
ar.par	a character-string that is the label of the autoregressive coefficients.
ma.par	a character-string that is the label of the autoregressive coefficients.
loc.par	the location coefficient.
Cogarch.var	a character-string that is the label of the observed cogarch process.
V.var	a character-string that is the label of the latent variance process.
Latent.var	a character-string that is the label of the latent process in the state space representation for the variance process.
jump.variable	the jump variable.
time.variable	the time variable.
measure	Levy measure of jump variables.
measure.type	type specification for Levy measure.
XinExpr	a vector of expressions identifying the starting conditions for Cogarch model.
startCogarch	Start condition for the Cogarch process
work	Internal Variable. In the final release this input will be removed.
...	Arguments to be passed to setCogarch such as the slots of the <a href="#">yuima.model-class</a>



## Details

We remark that `yuima` describes a `Cogarch(p,q)` model using the formulation proposed in Brockwell et al. (2006). This representation has the `Cogarch(1,1)` model introduced in Kluppelberg et al. (2004) as a special case. Indeed, by choosing  $\beta = a_0 b_1$ ,  $\eta = b_1$  and  $\phi = a_1$ , we obtain the `Cogarch(1,1)` model proposed in Kluppelberg et al. (2004) defined as the solution of the SDEs:

$$dG_t = \sqrt{V_t} dZ_t$$

$$dV_t = (\beta - \eta V_t) dt + \phi V_t d[Z_t Z_t]^{\{q\}}$$

Please refer to the vignettes and the examples.

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

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

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

## Value

`model` an object of `yuima.cogarch-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., Chadraa, E. and Lindner, A. (2006) Continuous-time GARCH processes, *The Annals of Applied Probability*, **16**, 790-826.

Kluppelberg, C., Lindner, A., & Maller, R. (2004) A continuous-time GARCH process driven by a Levy process: Stationarity and second-order behaviour, *Journal of Applied Probability*, **41**, 601-622.

## Examples

```
# Ex 1. (Continuous time GARCH process driven by a compound poisson process)
prova<-setCogarch(p=1,q=3,work=FALSE,
  measure=list(intensity="1", df=list("dnorm(z, 0, 1)")),
  measure.type="CP",
  Cogarch.var="y",
  V.var="v",
  Latent.var="x")
```

---

setData	<i>Set and access data of an object of type "yuima.data" or "yuima".</i>
---------	--

---

### 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, t0=0)
get.zoo.data(x)
```

### Arguments

<code>original.data</code>	some type of data, usually some sort of time series. The function always tries to convert to the input data into an object of <code>zoo</code> -type. See Details.
<code>x</code>	an object of type <code>yuima.data-class</code> or <code>yuima-class</code> .
<code>delta</code>	If there is the need to redefine on the fly the <code>delta</code> increment of the data to make it consistent to statistical theory. See Details.
<code>t0</code>	the time origin for the internal <code>zoo.data</code> slot, defaults to 0.

### 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 <code>yuima.data-class</code> for <code>setData</code> . The content of the <code>zoo.data</code> slot for <code>get.zoo.data</code>
-------	---

**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)

# changing delta on the fly to 1/252 and shifting time to t0=1
mysde2 <- setYuima(data=setData(X, delta=1/252, t0=1))
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:
#  $dX_t^e = X_t^e * dt + e * X_t^e * dW_t$ 
diff.matrix <- matrix( c("x*e"), 1,1)
model <- setModel(drift = c("x"), diffusion = diff.matrix)
# call option is evaluated by averaging
#  $\max\{ (1/T) * \int_0^T X_t^e dt, 0\}$ , 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)
```

---

`setHawkes`*Constructor of Hawkes model*

---

**Description**

...

**Usage**

```
setHawkes(lower.var = "0", upper.var = "t", var.dt = "s",  
          process = "N", dimension = 1, intensity = "lambda",  
          ExpKernParm1 = "c", ExpKernParm2 = "a", const = "nu",  
          measure = NULL, measure.type = NULL)
```

**Arguments**

<code>lower.var</code>	...
<code>upper.var</code>	...
<code>var.dt</code>	...
<code>process</code>	...
<code>dimension</code>	...
<code>intensity</code>	...
<code>ExpKernParm1</code>	...
<code>ExpKernParm2</code>	...
<code>const</code>	...
<code>measure</code>	...
<code>measure.type</code>	...

**Details**

...

**Value**

...

**Note**

...

**Author(s)**

...

**References**

...

**See Also**

...

---

`setIntegral`*Integral of Stochastic Differential Equation*

---

**Description**

'setIntegral' is the constructor of an object of class [yuima.Integral](#)

**Usage**

```
setIntegral(yuima, integrand, var.dx, lower.var, upper.var,
  out.var = "", nrow = 1, ncol = 1)
```

**Arguments**

<code>yuima</code>	an object of class <a href="#">yuima.model</a> that is the SDE.
<code>integrand</code>	A matrix or a vector of strings that describe each component of the integrand.
<code>var.dx</code>	A label that indicates the variable of integration
<code>lower.var</code>	A label that indicates the lower variable in the support of integration, by default <code>lower.var = 0</code> .
<code>upper.var</code>	A label that indicates the upper variable in the support of integration, by default <code>upper.var = t</code> .
<code>out.var</code>	Label for the output
<code>nrow</code>	Dimension of output if <code>integrand</code> is a vector of string.
<code>ncol</code>	Dimension of output if <code>integrand</code> is a vector of string.

**Value**

The constructor returns an object of class [yuima.Integral](#).

**Author(s)**

The YUIMA Project Team

**References**

Yuima Documentation

**Examples**

```

## Not run:
# Definition Model

Mod1<-setModel(drift=c("a1"), diffusion = matrix(c("s1"),1,1),
  solve.variable = c("X"), time.variable = "s")

# In this example we define an integral of SDE such as
# \[
# I=\int^{t}_{0} b*\exp(-a*(t-s))*(X_s-a1*s)dX_s
# \]

integ <- matrix("b*exp(-a*(t-s))*(X-a1*s)",1,1)

Integral <- setIntegral(yuima = Mod1,integrand = integ,
  var.dx = "X", lower.var = "0", upper.var = "t",
  out.var = "", nrow =1 ,ncol=1)

# Structure of slots

is(Integral)
# Function h in the above definition
Integral@Integral@Integrand@IntegrandList
# Dimension of Intgrand
Integral@Integral@Integrand@dimIntegrand

# all parameters are $\left(b,a,a1,s1\right)$
Integral@Integral@param.Integral@allparam

# the parameters in the integrand are $\left(b,a,a1\right)$ \newline
Integral@Integral@param.Integral@Integrandparam

# common parameters are $a1$
Integral@Integral@param.Integral@common

# integral variable dX_s
Integral@Integral@variable.Integral@var.dx
Integral@Integral@variable.Integral@var.time

# lower and upper vars
Integral@Integral@variable.Integral@lower.var
Integral@Integral@variable.Integral@upper.var

## End(Not run)

```

**Description**

Constructor of a random variable

**Usage**

```
setLaw(rng = function(n, ...) {  
  NULL  
}, density = function(x, ...) {  
  NULL  
}, cdf = function(q, ...) {  
  NULL  
}, quant = function(p, ...) {  
  NULL  
}, characteristic = function(u, ...) {  
  NULL  
}, time.var = "t", dim = NA)
```

**Arguments**

rng	function
density	function
cdf	function
characteristic	function
quant	function
time.var	label
dim	label

**Details**

Insert additional info

**Value**

object of class `yuima.law`

**Note**

Insert additional info

**Author(s)**

YUIMA TEAM



setMap

*Map of a Stochastic Differential Equation***Description**

'setMap' is the constructor of an object of class `yuima.Map` that describes a map of a SDE

**Usage**

```
setMap(func, yuima, out.var = "", nrow = 1, ncol = 1)
```

**Arguments**

<code>func</code>	a matrix or a vector of strings that describe each component of the map.
<code>yuima</code>	an object of class <code>yuima.model</code> that is the SDE.
<code>out.var</code>	label for the output
<code>nrow</code>	dimension of Map if <code>func</code> is a vector of string.
<code>ncol</code>	dimension of output if <code>func</code> is a vector of string.

**Value**

The constructor returns an object of class `yuima.Map`.

**Author(s)**

The YUIMA Project Team

**References**

Yuima Documentation

**Examples**

```
## Not run:
# Definition of a yuima model
mod <- setModel(drift=c("a1", "a2"),
  diffusion = matrix(c("s1", "0", "0", "s2"),2,2),
  solve.variable = c("X", "Y"))

# Definition of a map
my.Map <- matrix(c("(X+Y)", "-X-Y",
  "a*exp(X-a1*t)", "b*exp(Y-a2*t)"),
  nrow=2, ncol=2)

# Construction of yuima.Map

yuimaMap <- setMap(func = my.Map, yuima = mod,
```

```

out.var = c("f11","f21","f12","f22")

# Simulation of a Map

set.seed(123)
samp <- setSampling(0, 100,n = 1000)
mypar <- list(a=1, b=1, s1=0.1, s2=0.2, a1=0.1, a2=0.1)
sim1 <- simulate(object = yuimaMap, true.parameter = mypar,
  sampling = samp)

# plot

plot(sim1, ylab = yuimaMap@Output@param@out.var,
  main = "simulation Map", cex.main = 0.8)

## End(Not run)

```

---

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:

$$dX_t = a(t, X_t, \alpha)dt + b(t, X_t, \beta)dW_t + c(t, X_t, \gamma)dZ_t, \quad X_0 = x_0$$

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
#  $dX_t = -3X_t dt + (1/(1+X_t^2+t))dW_t$ ,
# 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
#  $dX_{1t} = -3X_{1t}dt + dW_{1t} + X_{2t}dW_{3t}$ ,
#  $dX_{2t} = -(X_{1t} + 2X_{2t})dt + X_{1t}dW_{1t} + 3dW_{2t}$ ,
# 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 (compound Poisson process))
# To describe
#  $dX_t = -\theta X_t dt + \sigma dZ_t$ 
mod3 <- setModel(drift=c("-theta*x"), diffusion="sigma",
jump.coef="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 jumps (stable process))
# To describe
#  $dX_t = -\theta X_t dt + \sigma dZ_t$ 
```

```

mod4 <- setModel(drift=c("-theta*x"), diffusion="sigma",
jump.coeff="1", measure.type="code",measure=list(df="rstable(z,1,0,1,0)"), solve.variable="x")
# Look at the model structure by
str(mod4)
# See rng about other candidate of Levy noises.

# Ex 5. (Two-dimensional stochastic differenatial equation with Levy noise)
# To describe
# dX1t = (1 - X1t - X2t)*dt+dZ1t
# dX2t = (0.5 - X1t - X2t)*dt+dZ2t
beta<-c(.5,.5)
mu<-c(0,0)
Lambda<-matrix(c(1,0,0,1),2,2)
mod5 <- setModel(drift=c("1 - x1-x2",".5 - x1-x2"),
solve.variable=c("x1","x2"), jump.coeff=Lambda, measure.type="code",
measure=list(df="rNIG(z, alpha, beta, delta0, mu, Lambda)"))
# Look at the model structure by
str(mod5)

# Ex 6. (Process with fractional Gaussian noise)
# dYt = 3*Yt*dt + dWt^h
mod6 <- setModel(drift="3*y", diffusion=1, hurst=0.3, solve.variable=c("y"))
# Look at the model structure by
str(mod6)

```

---

setPoisson

*Basic constructor for Compound Poisson processes*


---

### Description

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

$$M_t = m_0 + \sum_{i=0}^{N_t} c * Y_{\tau_i}, \quad M_0 = m_0$$

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

### Usage

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

### Arguments

intensity	either an expression or a numerical value representing the intensity function of the Poisson process $N_t$ .
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 <a href="#">setModel</a>

**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")
```

---

setPpr *Point Process*

---

## Description

Constructor of a Point Process

## Usage

```
setPpr(yuima, counting.var = "N", gFun, Kernel,  
       var.dx = "s", var.dt = "s", lambda.var = "lambda",  
       lower.var = "0", upper.var = "t", nrow = 1, ncol = 1)
```

## Arguments

yuima	...
counting.var	...
gFun	...
Kernel	...
var.dx	...
var.dt	...
lambda.var	...
lower.var	...
upper.var	...
nrow	...
ncol	...

## Details

...

## Value

...

## Note

...

## Author(s)

...

**References**

...

**See Also**

...

---

`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**

<code>Initial</code>	Initial time of the grid.
<code>Terminal</code>	Terminal time of the grid.
<code>n</code>	number of time intervals.
<code>delta</code>	mesh size in case of regular time grid.
<code>grid</code>	a grid of times for the simulation, possibly empty.
<code>random</code>	specify if it is random sampling. See Details.
<code>sdelta</code>	mesh size in case of regular space grid.
<code>sgrid</code>	a grid in space for the simulation, possibly empty.
<code>interpolation</code>	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 "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. of 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  $(\text{Terminal}-\text{Initial})/n$ . If `delta` is specified, the `Terminal` is adjusted to be equal to  $\text{Initial}+n*\text{delta}$ .

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	<i>Creates a "yuima" object by combining "model", "data", "sampling", "characteristic" and "functional" slots.</i>
----------	--

---

### Description

setYuima constructs an object of [yuima-class](#).

### Usage

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

### Arguments

data	an object of <a href="#">yuima.data-class</a> .
model	an object of <a href="#">yuima.model-class</a> .
sampling	an object of <a href="#">yuima.sampling-class</a> .
characteristic	an object of <a href="#">yuima.characteristic-class</a> .
functional	an object of class <a href="#">yuima.functional-class</a> .

### 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
#  $dX_t^e = -\theta_2 * X_t^e * dt + \theta_1 * dW_t$ 
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	<i>Calculate the value of functional</i>
---------------	--

---

**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:
#  $dX_t^e = X_t^e * dt + e * X_t^e * dW_t$ 
diff.matrix <- matrix( c("x*e"), 1,1)
model <- setModel(drift = c("x"), diffusion = diff.matrix)
# call option is evaluated by averaging
#  $\max\{ (1/T) * \int_0^T X_t^e dt, 0\}$ , 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, method = "euler", hurst, methodfGn = "WoodChan",
  sampling=sampling, subsampling=subsampling, ...)

```

**Arguments**

object	an <a href="#">yuima-class</a> , <a href="#">yuima.model-class</a> or <a href="#">yuima.carma-class</a> 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.

method	string Variable for simulation scheme. The default value method=euler uses the euler discretization for the simulation of a sample path.
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 <code>setSampling</code> to create a sampling
sampling	a <code>yuima.sampling-class</code> object.
subsampling	a <code>yuima.sampling-class</code> 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).

For a COGARCH(p,q) process setting `method=mixed` implies that the simulation scheme is based on the solution of the state space process. For the case in which the underlying noise is a compound poisson Levy process, the trajectory is build firstly by simulation of the jump time, then the quadratic variation and the increments noise are simulated exactly at jump time. For the others Levy process, the simulation scheme is based on the discretization of the state space process solution.

### Value

`yuima` a `yuima-class` object.

### Note

In the simulation of multi-variate Levy processes, the values of parameters have to be defined outside of `simulate` function in advance (see examples below).

### Author(s)

The YUIMA Project Team

**Examples**

```

set.seed(123)

# Path-simulation for 1-dim diffusion process.
#  $dX_t = -0.3 \cdot X_t \cdot dt + dW_t$ 
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.
#  $dX_t = \theta \cdot X_t \cdot dt + dW_t$ 
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:
#  $X=(X_1, X_2, X_3)$ ;  $dX_t = U(t, X_t)dt + V(t)dW_t$ 
# For drift coefficient
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)
# dx_t = a*(c-y)*dt + b*sqrt(X_t)*dW_t
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
##  $dX_t = -0.3 * X_t * dt + dW_t$ 

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:
##  $X=(X_1, X_2, X_3)$ ;  $dX_t = U(t, X_t)dt + V(t)dW_t$ 
## For drift coefficient
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)

```

---

spectralcov

*Spectral Method for Cumulative Covariance Estimation*


---

### Description

This function implements the local method of moments proposed in Bibinger et al. (2014) to estimate the cummulative covariance matrix of a non-synchronously observed multi-dimensional Ito process with noise.

### Usage

```

lmm(x, block = 20, freq = 50, freq.p = 10, K = 4, interval = c(0, 1),
    Sigma.p = NULL, noise.var = "AMZ", samp.adj = "direct", psd = TRUE)

```

**Arguments**

x	an object of <code>yuima-class</code> or <code>yuima.data-class</code> .
block	a positive integer indicating the number of the blocks which the observation interval is split into.
freq	a positive integer indicating the number of the frequencies used to compute the final estimator.
freq.p	a positive integer indicating the number of the frequencies used to compute the pilot estimator for the spot covariance matrix (corresponding to the number $J_n$ in Eq.(29) from Altmeyer and Bibinger (2015)).
K	a positive integer indicating the number of the blocks used to compute the pilot estimator for the spot covariance matrix (corresponding to the number $K_n$ in Eq.(29) from Altmeyer and Bibinger (2015)).
interval	a vector indicating the observation interval. The first component represents the initial value and the second component represents the terminal value.
Sigma.p	a block by <code>dim(x)</code> matrix giving the pilot estimates of the spot covariance matrix plugged into the optimal weight matrices. If NULL (the default), it is computed by using formula (29) from Altmeyer and Bibinger (2015).
noise.var	character string giving the method to estimate the noise variances. There are several options: "AMZ" (the default) uses equation (3.7) from Gatheral and Oomen (2010), i.e. the quasi-maximum likelihood estimator proposed by Ait-Sahalia et al. (2005) (see also Xiu (2010)). "BR" uses equation (3.9) from Gatheral and Oomen (2010), i.e. the sample average of the squared returns divided by 2, the estimator proposed by Bandi and Russel (2006). "O" uses equation (3.8) from Gatheral and Oomen (2010), i.e. another method-of-moments estimator proposed by Oomen (2006). It is also possible to directly specify the noise variances by setting this argument to a numeric vector. In this case the <i>i</i> -th component of <code>noise.var</code> must indicates the variance of the noise for the <i>i</i> -th component of the observation process.
samp.adj	character string giving the method to adjust the effect of the sampling times on the variances of the spectral statistics for the noise part. The default method "direct" uses the local sums of the squares of the one-skip differences of the sampling times divided by 2, which directly appears in the representation of the variances of the spectral statistics for the noise part. Another choice is "QVT", which uses the local quadratic variations of time as in Altmeyer and Bibinger (2015) and Bibinger et al. (2014).
psd	logical. If TRUE (the default), the estimated covariance matrix and variance-covariance matrix are converted to their spectral absolute values to ensure their positive semi-definiteness. This procedure does not matter in terms of the asymptotic theory.

**Details**

The default implementation is the adaptive version of the local method of moments estimator, which is only based on observation data. It is possible to implement oracle versions of the estimator by setting user-specified `Sigma.p` and/or `noise.var`. An example is given below.

**Value**

An object of class "yuima.specv", which is a list with the following elements:

covmat	the estimated covariance matrix
vcov	the estimated variance-covariance matrix of as.vector(covmat)
Sigma.p	the pilot estimates of the spot covariance matrix

**Author(s)**

The YUIMA Project Team

**References**

- Ait-Sahalia, Y., Mykland, P. A. and Zhang, L. (2005) How often to sample a continuous-time process in the presence of market microstructure noise, *The Review of Financial Studies*, **18**, 351–416.
- Altmeyer, R. and Bibinger, M. (2015) Functional stable limit theorems for quasi-efficient spectral covolatility estimators, to appear in *Stochastic processes and their applications*, doi:10.1016/j.spa.2015.07.009.
- Bandi, F. M. and Russell, J. R. (2006) Separating microstructure noise from volatility, *Journal of Financial Economics*, **79**, 655–692.
- Bibinger, M., Hautsch, N., Malec, P. and Reiss, M. (2014) Estimating the quadratic covariation matrix from noisy observations: local method of moments and efficiency, *Annals of Statistics*, **42**, 80–114.
- Gatheral J. and Oomen, R. C. A. (2010) Zero-intelligence realized variance estimation, *Finance Stochastics*, **14**, 249–283.
- Oomen, R. C. A. (2006) Properties of realized variance under alternative sampling schemes, *Journal of Business and Economic Statistics*, **24**, 219–237.
- Reiss, M. (2011) Asymptotic equivalence for inference on the volatility from noisy observations, *Annals of Statistics*, **39**, 772–802.
- Xiu, D. (2010) Quasi-maximum likelihood estimation of volatility with high frequency data, *Journal of Econometrics*, **159**, 235–250.

**See Also**

[cce](#), [setData](#)

**Examples**

```
# Example. One-dimensional and regular sampling case
# Here the simulated model is taken from Reiss (2011)

## Set a model
sigma <- function(t) sqrt(0.02 + 0.2 * (t - 0.5)^4)
modI <- setModel(drift = 0, diffusion = "sigma(t)")

## Generate a path of the process
set.seed(117)
n <- 12000
```

```

yuima.samp <- setSampling(Terminal = 1, n = n)
yuima <- setYuima(model = modI, sampling = yuima.samp)
yuima <- simulate(yuima, xinit = 0)

delta <- 0.01 # standard deviation of microstructure noise
yuima <- noisy.sampling(yuima, var.adj = delta^2) # generate noisy observations
plot(yuima)

## Estimation of the integrated volatility
est <- lmm(yuima)
est

## True integrated volatility and theoretical standard error
disc <- seq(0, 1, by = 1/n)
cat("true integrated volatility\n")
print(mean(sigma(disc[-1])^2))
cat("theoretical standard error\n")
print(sqrt(8*delta*mean(sigma(disc[-1])^3))/n^(1/4))

# Plotting the pilot estimate of the spot variance path
block <- 20
G <- seq(0,1,by=1/block)[1:block]
Sigma.p <- sigma(G)^2 # true spot variance
plot(zoo(Sigma.p, G), col = "blue",, xlab = "time",
      ylab = expression(sigma(t)^2))
lines(zoo(est$Sigma.p, G))

## "Oracle" implementation
lmm(yuima, block = block, Sigma.p = Sigma.p, noise.var = delta^2)

# Example. Multi-dimensional case
# We simulate noisy observations of a correlated bivariate Brownian motion
# First we examine the regular sampling case since in this situation the theoretical standard
# error can easily be computed via the formulae given in p.88 of Bibinger et al. (2014)

## Set a model
drift <- c(0,0)

rho <- 0.5 # correlation

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

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

## Generate a path of the latent process
set.seed(123)

## We regard the unit interval as 6.5 hours and generate the path on it
## with the step size equal to 1 seconds

n <- 8000
yuima.samp <- setSampling(Terminal = 1, n = n)

```

```

yuima <- setYuima(model = modII, sampling = yuima.samp)
yuima <- simulate(yuima)

## Generate noisy observations
eta <- 0.05
yuima <- noisy.sampling(yuima, var.adj = diag(eta^2, 2))
plot(yuima)

## Estimation of the integrated covariance matrix
est <- lmm(yuima)
est

## Theoretical standard error
a <- sqrt(4 * eta * (sqrt(1 + rho) + sqrt(1 - rho)))
b <- sqrt(2 * eta * ((1 + rho)^(3/2) + (1 - rho)^(3/2)))
cat("theoretical standard error\n")
print(matrix(c(a,b,b,a),2,2)/n^(1/4))

## "Oracle" implementation
block <- 20
Sigma.p <- matrix(c(1,rho,rho,1),block,4,byrow=TRUE) # true spot covariance matrix
lmm(yuima, block = block, Sigma.p = Sigma.p, noise.var = rep(eta^2,2))

# Next we extract nonsynchronous observations from
# the path generated above by Poisson random sampling
psample <- poisson.random.sampling(yuima, rate = c(1/2,1/2), n = n)

## Estimation of the integrated covariance matrix
lmm(psample)

## "Oracle" implementation
lmm(psample, block = block, Sigma.p = Sigma.p, noise.var = rep(eta^2,2))

## Other choices of tuning parameters (estimated values are not varied so much)
lmm(psample, block = 25)
lmm(psample, freq = 100)
lmm(psample, freq.p = 15)
lmm(psample, K = 8)

```

---

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 available 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
## frequency 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](#), [yuima.carma-class](#) or [yuima.cogarch-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,...)
## S3 method for class 'yuima.cogarch'
toLatex(object,...)

```

## Arguments

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

## Details

This method 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 coefficient
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)

# A COGARCH(p=3,q=5) process.
cogarch1<-setCogarch(p=3,q=5,
                    measure=list(df=list("rNIG(z, mu00, bu00, 1, 0)")),
                    measure.type="code")

str(cogarch1)
toLatex(cogarch1)

```

---

variable.Integral	<i>Class for the mathematical description of integral of a stochastic process</i>
-------------------	---

---

**Description**

Auxiliar class for definition of an object of class [yuima.Integral](#). see the documentation of [yuima.Integral](#) for more details.

---

ybook	<i>R code for the Yuima Book</i>
-------	----------------------------------

---

**Description**

Shows the R code corresponding to each chapter in the Yuima Book.

**Usage**

```
ybook(chapter)
```

**Arguments**

chapter	a number in 1:7
---------	-----------------

**Details**

This is an accessory function which open the R code corresponding to Chapter "chapter" in the Yuima Book so that the reader can replicate the code.

**Examples**

```
ybook(1)
```

---

 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.sampling", ...): 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 asynchronous 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.cogarch-class

*Class for the mathematical description of CoGarch(p,q) model*


---

### Description

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

### Objects from the Class

Objects can be created by calls of the function `setCogarch`.

### Slots

`info`: is an `cogarch.info-class` object that describes the structure of the Cogarch(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.

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

`measure`: Levy measure for the jump component.

`measure.type`: Type of specification for Levy measure

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

`state.variable`: the state variable.

`jump.variable`: the jump variable.

`time.variable`: the time variable.

`noise.number`: Object of class "numeric"

**equation.number:** dimension of the stochastic differential equation.  
**dimension:** number of parameters.  
**solve.variable:** the solve variable  
**xinit:** Object of class "expression" that contains the starting function for the SDE.  
**J.flag:** wheather jump.coef include jump.variable.

### Extends

Class "[yuima.model](#)", directly.

### Methods

**simulate** simulation method. For more information see [simulate](#)  
**toLatex** This method converts an object of `yuima.cogarch`-class to character vectors with LaTeX markup.  
**qmle** Quasi maximum likelihood estimation procedure. For more information see [qmle](#).

### 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	<i>Class "yuima.data" for the data slot of a "yuima" class object</i>
------------------	---

---

**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 `initialize` 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 asynchronous 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.Hawkes`

*Class for a mathematical description of a Point Process*

---

**Description**

Insert Here documentation

---

yuima.Integral-class    *Class for the mathematical description of integral of a stochastic process*

---

## Description

The `yuima.Integral` class is a class of the **yuima** package that extends the `yuima-class` it represents a integral of a stochastic process

$$z_t = \int_{t_0}^t h(\theta, X_s, s) \, dX_s$$

## Slots

In the following we report the the additional slots of an object of class `yuima.Integral` with respect to the `yuima-class`:

**Integral:** It is an object of class `Integral.sde` and it is composed by the following slots:

**param.Integral:** it is an object of class `param.Integral` and it is composed by the following slots:

**allparam:** labels of all parameters (model and integral).

**common:** common parameters.

**Integrandparam:** labels of all parameters only in the integral.

**variable.Integral:** it is an object of class `variable.Integral` and it is composed by the following slots:

**var.dx:** integral variable.

**lower.var:** lower bound of support.

**upper.var:** upper bound of support.

**out.var:** labels of output.

**var.time:** label of time.

**Integrand:** it is an object of class `variable.Integral` and it is composed by the following slots:

**IntegrandList:** It is a list that contains the components of integrand  $h(\theta, X_s, s)$ .

**dimIntegrand:** a numeric object that is the dimensions of the output.

## Methods

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

---

yuima.law-class	<i>Class of yuima law</i>
-----------------	---------------------------

---

**Description**

Insert Description Here

**Slots**

...:

**rng** function

**density** function

**cdf** function

**quantile** function

**characteristic** function

**param.measure** ...

**time.var** label

**dim** number

**Methods**

**rand** signature(object = "yuima.law", n = "numeric", param = "list", ...):  
INSERT HERE DESCRIPTION

**dens** signature(object = "yuima.law", x = "numeric", param = "list", log = FALSE, ...):  
INSERT HERE DESCRIPTION

**cdf** signature(object = "yuima.law", q = "numeric", param = "list", ...): INSERT  
HERE DESCRIPTION

**quant** signature(object = "yuima.law", p = "numeric", param = "list", ...):  
INSERT HERE DESCRIPTION

**char** signature(object = "yuima.law", u = "numeric", param = "list", ...): INSERT  
HERE DESCRIPTION

**Author(s)**

The YUIMA Project Team

---

yuima.Map-class	<i>Class for the mathematical description of function of a stochastic process</i>
-----------------	---

---

## Description

The `yuima.Map` class is a class of the **yuima** package that extends the [yuima-class](#) it represents a map of a stochastic process

$$zt = g(\text{theta}, X_t, t) : \mathbb{R}^{\{q \times d \times 1\}} \rightarrow \mathbb{R}^{\{l_1 \times l_2 \times \dots\}}$$

or an operator between two independent stochastic process:

$$zt = h(\text{theta}, X_t, Y_t, t)$$

where  $X_t$  and  $Y_t$  are object of class [yuima.model-class](#) or [yuima-class](#) with the same dimension.

## Slots

Here we report the additional slots of an object of class `yuima.Map` with respect to the [yuima-class](#):

**Output:** It is an object of class `info.Map` and it is composed by the following slots:

**formula:** It is a vector that contains the components of map  $g(\text{theta}, X_t, t)$  or the operator  $h(\text{theta}, X_t, Y_t, t)$

**dimension:** a numeric object that is the dimensions of the Map.

**type:** If `type = "Maps"`, the Map is a map of stochastic process, If `type = "Operator"`, the result is an operator between two independent stochastic process

**param** it is an object of class `param.Map` and it is composed by the following slots:

**out.var:** labels for Map.

**allparam:** labels of all parameters (model and map/operators).

**allparamMap:** labels of map/operator parameters.

**common:** common parameters.

**Input.var:** labels for inputs.

**time.var:** label for time variable.

## Methods

**simulate** simulation method. For more information see [simulate](#).

## Author(s)

The YUIMA Project Team

---

yuima.model-class	<i>Classes for the mathematical description of stochastic differential equations</i>
-------------------	--

---

## 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.multimodel-class

*Class for the mathematical description of Multi dimensional Jump Diffusion processes*

---

## Description

The `yuima.multimodel` 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 containing 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

**Examples**

```

## Not run:
# We define the density function of the underlying Levy

dmyexp <- function(z, sig1, sig2, sig3){
  rep(0,3)
}

# We define the random number generator

rmyexp <- function(z, sig1, sig2, sig3){
  cbind(rnorm(z,0,sig1), rgamma(z,1,sig2), rnorm(z,0,sig3))
}

# Model Definition: in this case we consider only a multi
# compound poisson process with a common intensity as underlying
# noise

mod <- setModel(drift = matrix(c("0","0","0"),3,1), diffusion = NULL,
  jump.coeff = matrix(c("1","0","0","0","1","-1","1","0","0"),3,3),
  measure = list( intensity = "lambda1", df = "dmyexp(z,sig1,sig2,sig3)"),
  jump.variable = c("z"), measure.type=c("CP"),
  solve.variable=c("X1","X2","X3"))

# Sample scheme

samp<-setSampling(0,100,n=1000)
param <- list(lambda1 = 1, sig1 = 0.1, sig2 = 0.1, sig3 = 0.1)

# Simulation

traj <- simulate(object = mod, sampling = samp,
  true.parameter = param)

# Plot

plot(traj, main = " driven noise. Multidimensional CP",
  cex.main = 0.8)

# We construct a multidimensional SDE driven by a multivariate
# levy process without CP components.

# Definition multivariate density

dmyexp1 <- function(z, sig1, sig2, sig3){
  rep(0,3)
}

# Definition of random number generator
# In this case user must define the delta parameter in order to
# control the effect of time interval in the simulation.

```



```

rmyexp1 <- function(z, sig1, sig2, sig3, delta){
  cbind(rexp(z,sig1*delta), rgamma(z,1*delta,sig2), rexp(z,sig3*delta))
}

# Model defintion

mod1 <- setModel(drift=matrix(c("0.1*(0.01-X1)",
  "0.05*(1-X2)", "0.1*(0.1-X3)"),3,1), diffusion=NULL,
  jump.coeff = matrix(c("0.01", "0", "0", "0", "0.01",
  "0", "0", "0", "0.01"),3,3),
  measure = list(df="dmyexp1(z,sig1,sig2,sig3)"),
  jump.variable = c("z"), measure.type=c("code"),
  solve.variable=c("X1", "X2", "X3"),xinit=c("10", "1.2", "10"))

# Simulation sample paths

samp<-setSampling(0,100,n=1000)
param <- list(sig1 = 1, sig2 = 1, sig3 = 1)

# Simulation

set.seed(1)
traj1 <- simulate(object = mod1, sampling = samp,
  true.parameter = param)

# Plot

plot(traj1, main = "driven noise: multi Levy without CP",
  cex.main = 0.8)

# We construct a multidimensional SDE driven by a multivariate
# levy process.

# We consider a mixed situation where some
# noise are driven by a multivariate Compuond Poisson that
# shares a common intensity parameters.

### Multi Levy model

rmyexample2<-function(z,sig1,sig2,sig3, delta){
  if(missing(delta)){
    delta<-1
  }
  cbind(rexp(z,sig1*delta), rgamma(z,1*delta,sig2),
    rexp(z,sig3*delta), rep(1,z),
    rep(1,z))
}

dmyexample2<-function(z,sig1,sig2,sig3){
  rep(0,5)
}

# Definition Model

```

```

mod2 <- setModel(drift=matrix(c("0.1*(0.01-X1)",
  "0.05*(1-X2)", "0.1*(0.1-X3)", "0", "0"),5,1), diffusion=NULL,
  jump.coeff = matrix(c("0.01", "0", "0", "0", "0",
    "0", "0.01", "0", "0", "0",
    "0", "0", "0.01", "0", "0",
    "0", "0", "0", "0.01", "0",
    "0", "0", "0", "0", "0.01"),5,5),
  measure = list(df = "dmyexample2(z, sig1, sig2, sig3)",
    intensity = "lambda1"),
  jump.variable = c("z"),
  measure.type=c("code", "code", "code", "CP", "CP"),
  solve.variable=c("X1", "X2", "X3", "X4", "X5"),
  xinit=c("10", "1.2", "10", "0", "0"))

# Simulation scheme
samp <- setSampling(0, 100, n = 1000)
param <- list(sig1 = 1, sig2 = 1, sig3 = 1, lambda1 = 1)

# Simulation

set.seed(1)
traj2 <- simulate(object = mod2, sampling = samp,
  true.parameter = param)

plot(traj2, main = "driven noise: general multi Levy", cex.main = 0.8)

## End(Not run)

```

---

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 containing 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.Ppr

*Class for a mathematical description of a Point Process*

---

## Description

Insert Here documentation

---

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

## \*Topic **Estimation COGARCH**

[gmm](#), 31

## \*Topic **Information criteria**

[IC](#), 36

## \*Topic **Method of Moments**

[gmm](#), 31

## \*Topic **\textasciitildekw1**

[Diagnostic.Carma](#), 28

[qmlLevy](#), 75

[setCogarch](#), 88

## \*Topic **\textasciitildekw2**

[Diagnostic.Carma](#), 28

[qmlLevy](#), 75

[setCogarch](#), 88

## \*Topic **classes**

[carma.info-class](#), 9

[cogarch.est.-class](#), 21

[cogarch.est.incr-class](#), 22

[cogarch.info-class](#), 23

[model.parameter-class](#), 57

[yuima-class](#), 124

[yuima.carma-class](#), 125

[yuima.carma.qml-class](#), 126

[yuima.characteristic-class](#), 127

[yuima.cogarch-class](#), 127

[yuima.CP.qml-class](#), 128

[yuima.data-class](#), 129

[yuima.functional-class](#), 130

[yuima.law-class](#), 132

[yuima.Map-class](#), 133

[yuima.model-class](#), 134

[yuima.multimodel-class](#), 135

[yuima.poisson-class](#), 138

[yuima.sampling-class](#), 139

## \*Topic **datasets**

[LogSPX](#), 50

[MWK151](#), 60

## \*Topic **misc**

[toLatex](#), 121

[ybook](#), 123

## \*Topic **ts**

[adaBayes](#), 3

[asymptotic\\_term](#), 5

[bns.test](#), 7

[CarmaNoise](#), 10

[cce](#), 13

[cogarchNoise](#), 24

[CPoint](#), 25

[hyavar](#), 33

[lasso](#), 42

[limiting.gamma](#), 44

[llag](#), 46

[lseBayes](#), 51

[mllag](#), 53

[mmfrac](#), 56

[mpv](#), 58

[noisy.sampling](#), 61

[phi.test](#), 64

[poisson.random.sampling](#), 65

[qgv](#), 66

[qml](#), 67

[rconst](#), 78

[rng](#), 79

[setCarma](#), 84

[setCharacteristic](#), 87

[setData](#), 90

[setFunctional](#), 91

[setModel](#), 98

[setPoisson](#), 101

[setSampling](#), 104

[setYuima](#), 106

[simFunctional](#), 107

[simulate](#), 108

[spectralcov](#), 115

[subsampling](#), 119

[adaBayes](#), 3

[adaBayes,yuima-method \(adaBayes\)](#), 3

[arma0](#), 17

- asymptotic\_term, 5
- asymptotic\_term, yuima-method (asymptotic\_term), 5
- bns.test, 7
- bns.test, list-method (bns.test), 7
- bns.test, yuima-method (bns.test), 7
- bns.test, yuima.data-method (bns.test), 7
- CARMA (setCarma), 84
- Carma (setCarma), 84
- carma.info-class, 9
- carma.qmle (yuima.carma.qmle-class), 126
- Carma.Recovering (CarmaNoise), 10
- CarmaNoise, 10, 68, 126
- CarmaRecovNoise (CarmaNoise), 10
- cbind.yuima (setData), 90
- cbind.yuima, yuima-method (yuima-class), 124
- cbind.yuima, yuima.data-method (yuima.data-class), 129
- cce, 13, 34, 35, 48, 59, 62, 65, 117, 124, 130
- cce, yuima-method (yuima-class), 124
- cce, yuima.data-method (yuima.data-class), 129
- cdf (LawMethods), 43
- cdf, yuima.law-method (yuima.law-class), 132
- char (LawMethods), 43
- char, yuima.law-method (yuima.law-class), 132
- COGARCH (setCogarch), 88
- CoGarch (setCogarch), 88
- Cogarch (setCogarch), 88
- cogarch (setCogarch), 88
- cogarch.est-class (cogarch.est.-class), 21
- cogarch.est.-class, 21
- cogarch.est.incr-class, 22
- cogarch.info-class, 23
- cogarch.Recovering (cogarchNoise), 24
- cogarchNoise, 22, 24
- CogarchRecovNoise (cogarchNoise), 24
- CP.qmle (yuima.CP.qmle-class), 128
- CPoint, 25
- Data (LogSPX), 50
- dbgamma (rng), 79
- dconst (rconst), 78
- dens (LawMethods), 43
- dens, yuima.law-method (yuima.law-class), 132
- dGH (rng), 79
- dGIG (rng), 79
- Diagnostic.Carma, 28
- Diagnostic.Cogarch, 29
- dIG (rng), 79
- dim, 90, 130
- dim (setData), 90
- dim, yuima-method (yuima-class), 124
- dim, yuima.data-method (yuima.data-class), 129
- dNIG (rng), 79
- dvgamma (rng), 79
- est.cogarch.incr-class (cogarch.est.incr-class), 22
- F0 (simFunctional), 107
- F0, yuima-method (simFunctional), 107
- Fnorm (simFunctional), 107
- Fnorm, yuima-method (simFunctional), 107
- get.zoo.data (setData), 90
- get.zoo.data, yuima-method (yuima-class), 124
- get.zoo.data, yuima.data-method (yuima.data-class), 129
- gete (setFunctional), 91
- gete, yuima.functional-method (yuima.functional-class), 130
- getF (setFunctional), 91
- getf (setFunctional), 91
- getF, yuima.functional-method (yuima.functional-class), 130
- getf, yuima.functional-method (yuima.functional-class), 130
- getxinit (setFunctional), 91
- getxinit, yuima.functional-method (yuima.functional-class), 130
- gmm, 21, 22, 31
- hyavar, 17, 18, 33, 46–48, 54
- IC, 36
- info.Map (info.Map-class), 40
- info.Map-class, 40
- info.Ppr, 40

- info.Ppr-class (info.Ppr), 40
- initialize, carma.info-method  
(yuima.carma-class), 125
- initialize, cogarch.info-method  
(yuima.cogarch-class), 127
- initialize, info.Map-method  
(info.Map-class), 40
- initialize, info.Ppr-method (info.Ppr),  
40
- initialize, Integral.sde-method  
(Integral.sde), 40
- initialize, Integrand-method  
(Integrand), 40
- initialize, model.parameter-method  
(yuima.model-class), 134
- initialize, param.Integral-method  
(param.Integral), 63
- initialize, param.Map-method  
(param.Map-class), 63
- initialize, variable.Integral-method  
(variable.Integral), 123
- initialize, yuima-method (yuima-class),  
124
- initialize, yuima.carma-method  
(yuima.carma-class), 125
- initialize, yuima.characteristic-method  
(yuima.characteristic-class),  
127
- initialize, yuima.cogarch-method  
(yuima.cogarch-class), 127
- initialize, yuima.data-method  
(yuima.data-class), 129
- initialize, yuima.functional-method  
(yuima.functional-class), 130
- initialize, yuima.Hawkes-method  
(yuima.Hawkes), 130
- initialize, yuima.Integral-method  
(yuima.Integral-class), 131
- initialize, yuima.law-method  
(yuima.law-class), 132
- initialize, yuima.Map-method  
(yuima.Map-class), 133
- initialize, yuima.model-method  
(yuima.model-class), 134
- initialize, yuima.multimodel-method  
(yuima.multimodel-class), 135
- initialize, yuima.poisson-method  
(yuima.poisson-class), 138
- initialize, yuima.Ppr-method  
(yuima.Ppr), 139
- initialize, yuima.sampling-method  
(yuima.sampling-class), 139
- Integral.sde, 40
- Integral.sde-class (Integral.sde), 40
- Integrand, 40
- Integrand-class (Integrand), 40
- Intensity.PPR, 41
- lambdaFromData, 41
- lasso, 42
- LawMethods, 43
- length, 90, 130
- length (setData), 90
- length, yuima-method (yuima-class), 124
- length, yuima.data-method  
(yuima.data-class), 129
- Levy.Carma (CarmaNoise), 10
- Levy.cogarch (cogarchNoise), 24
- limiting.gamma, 44
- limiting.gamma, yuima-method  
(yuima-class), 124
- limiting.gamma, yuima.carma-method  
(yuima.carma-class), 125
- limiting.gamma, yuima.cogarch-method  
(yuima.cogarch-class), 127
- limiting.gamma, yuima.model-method  
(yuima.model-class), 134
- llag, 46, 53, 54, 125, 130
- llag, list-method (llag), 46
- llag, yuima-method (yuima-class), 124
- llag, yuima.data-method  
(yuima.data-class), 129
- lmm, 18, 62
- lmm (spectralcov), 115
- LogSPX, 50
- lse (qml), 67
- LSE, yuima-method (yuima-class), 124
- lseBayes, 51
- lseBayes, yuima-method (lseBayes), 51
- Map of SDE (setMap), 97
- Map of yuima (setMap), 97
- Method of Moment COGARCH (gmm), 31
- ml.ql, yuima-method (yuima-class), 124
- mllag, 48, 53
- mmfrac, 56, 67
- model.parameter-class, 57

- mpv, [8](#), [58](#)
- mpv, yuima-method (mpv), [58](#)
- mpv, yuima.data-method (mpv), [58](#)
- MWK151, [60](#)
- noisy.sampling, [61](#)
- noisy.sampling, yuima-method (noisy.sampling), [61](#)
- noisy.sampling, yuima.data-method (noisy.sampling), [61](#)
- optim, [32](#), [42](#), [68](#)
- param.Integral, [63](#)
- param.Integral-class (param.Integral), [63](#)
- param.Map (param.Map-class), [63](#)
- param.Map-class, [63](#)
- phi.test, [64](#)
- plot, [124](#), [130](#)
- plot, cogarch.est., ANY-method (cogarch.est.-class), [21](#)
- plot, cogarch.est.incr, ANY-method (cogarch.est.incr-class), [22](#)
- plot, yuima, ANY-method (yuima-class), [124](#)
- plot, yuima.carma.qmle, ANY-method (yuima.carma.qmle-class), [126](#)
- plot, yuima.CP.qmle, ANY-method (yuima.CP.qmle-class), [128](#)
- plot, yuima.data, ANY-method (yuima.data-class), [129](#)
- poisson.random.sampling, [65](#)
- poisson.random.sampling, yuima-method (yuima-class), [124](#)
- poisson.random.sampling, yuima.data-method (yuima.data-class), [129](#)
- pseudologlikelihood (qmle), [67](#)
- qgv, [56](#), [57](#), [66](#)
- ql, yuima-method (yuima-class), [124](#)
- qmle, [21](#), [22](#), [25](#), [37](#), [42](#), [64](#), [67](#), [126](#), [128](#), [135](#), [139](#)
- qmle.carma (yuima.carma.qmle-class), [126](#)
- qmle.CP (yuima.CP.qmle-class), [128](#)
- qmleL (CPoint), [25](#)
- qmleLevy, [75](#)
- qmleR (CPoint), [25](#)
- quant (LawMethods), [43](#)
- quant, yuima.law-method (yuima.law-class), [132](#)
- quasilogl (qmle), [67](#)
- rand (LawMethods), [43](#)
- rand, yuima.law-method (yuima.law-class), [132](#)
- rand-method (LawMethods), [43](#)
- rbgamma (rng), [79](#)
- rconst, [78](#)
- Recovering.Noise (CarmaNoise), [10](#)
- Recovering.Noise.cogarch (cogarchNoise), [24](#)
- rGH (rng), [79](#)
- rGIG (rng), [79](#)
- rIG (rng), [79](#)
- rng, [79](#)
- rNIG (rng), [79](#)
- rnts (rng), [79](#)
- rpts (rng), [79](#)
- rql (qmle), [67](#)
- rql, yuima-method (yuima-class), [124](#)
- rstable (rng), [79](#)
- rvgamma (rng), [79](#)
- setCarma, [9](#), [84](#)
- setCharacteristic, [87](#)
- setCogarch, [23](#), [88](#), [127](#)
- setData, [18](#), [35](#), [90](#), [117](#), [129](#)
- setFunctional, [91](#)
- setFunctional, yuima-method (setFunctional), [91](#)
- setFunctional, yuima.model-method (setFunctional), [91](#)
- setHawkes, [93](#)
- setIntegral, [94](#)
- setLaw, [95](#)
- setMap, [97](#)
- setModel, [18](#), [57](#), [59](#), [98](#), [101](#)
- setPoisson, [101](#), [135](#), [138](#)
- setPpr, [103](#)
- setSampling, [104](#), [109](#), [139](#)
- setYuima, [106](#)
- simFunctional, [107](#)
- simFunctional, yuima-method (simFunctional), [107](#)
- simulate, [23](#), [105](#), [108](#), [125](#), [126](#), [128](#), [131](#), [133](#), [135](#), [139](#)
- simulate, cogarch.est.incr-method (cogarch.est.incr-class), [22](#)
- simulate, yuima-method (yuima-class), [124](#)



- simulate, `yuima.carma`-method  
(`yuima.carma`-class), 125
- simulate, `yuima.cogarch`-method  
(`yuima.cogarch`-class), 127
- simulate, `yuima.Hawkes`-method  
(`yuima.Hawkes`), 130
- simulate, `yuima.Integral`-method  
(`yuima.Integral`-class), 131
- simulate, `yuima.Map`-method  
(`yuima.Map`-class), 133
- simulate, `yuima.model`-method  
(`yuima.model`-class), 134
- simulate, `yuima.multimodel`-method  
(`yuima.multimodel`-class), 135
- simulate, `yuima.Ppr`-method (`yuima.Ppr`),  
139
- spectralcov, 115
- subsampling, 119, 139
- subsampling, `yuima`-method (`yuima`-class),  
124
- subsampling, `yuima.data`-method  
(`yuima.data`-class), 129
  
- toLatex, 121
  
- variable.Integral, 123
- variable.Integral-class  
(`variable.Integral`), 123
  
- ybook, 123
- yuima-class, 124
- yuima.carma-class, 125
- yuima.carma.qmle-class, 126
- yuima.characteristic-class, 127
- yuima.cogarch-class, 127
- yuima.CP.qmle-class, 128
- yuima.data-class, 129
- yuima.functional-class, 130
- yuima.Hawkes, 40, 130
- yuima.Hawkes-class (`yuima.Hawkes`), 130
- yuima.Integral, 40, 63, 94, 123
- yuima.Integral (`yuima.Integral`-class),  
131
- yuima.Integral-class, 131
- yuima.law (`yuima.law`-class), 132
- yuima.law-class, 132
- yuima.Map, 40, 63, 97
- yuima.Map (`yuima.Map`-class), 133
- yuima.Map-class, 133
- yuima.model, 94, 97, 128
- yuima.model (`yuima.model`-class), 134
- yuima.model-class, 134
- yuima.multimodel  
(`yuima.multimodel`-class), 135
- yuima.multimodel-class, 135
- yuima.poisson-class, 138
- yuima.Ppr, 40, 139
- yuima.Ppr-class (`yuima.Ppr`), 139
- yuima.qmle-class (`yuima.CP.qmle`-class),  
128
- yuima.sampling-class, 139
  
- zoo, 22, 47, 90, 124, 126, 129, 130