Package ‘yuima’

February 29, 2024

Type Package
Title The YUIMA Project Package for SDEs
Version 1.15.27
Depends R(>= 2.10.0), methods, zoo, stats4, utils, expm, cubature, mvtnorm
Imports Rcpp (>= 0.12.1), boot (>= 1.3-2), glasstoFast, coda, calculus (>= 0.2.0), statmod,
Author YUIMA Project Team
Maintainer Stefano M. Iacus <siacus@iq.harvard.edu>
Description Simulation and Inference for SDEs and Other Stochastic Processes.
License GPL-2
URL https://yuimaproject.com
BugReports https://github.com/yuimaproject/yuima/issues
LinkingTo Rcpp, RcppArmadillo
NeedsCompilation yes
Repository CRAN
Date/Publication 2024-02-29 21:30:03 UTC

R topics documented:

  adaBayes .......................................................... 4
  ae .............................................................. 6
  aeCharacteristic ................................................. 8
  aeDensity ......................................................... 9
  aeExpectation .................................................... 10
  aeKurtosis ......................................................... 11
  aeMarginal ....................................................... 12
  aeMean .......................................................... 13
  aeMoment ......................................................... 14
  aeSd ............................................................. 15
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>aeSkewness</td>
<td>16</td>
</tr>
<tr>
<td>asymptotic_term</td>
<td>17</td>
</tr>
<tr>
<td>bns.test</td>
<td>19</td>
</tr>
<tr>
<td>carma.info-class</td>
<td>21</td>
</tr>
<tr>
<td>carmaHawkes.info-class</td>
<td>22</td>
</tr>
<tr>
<td>CarmaNoise</td>
<td>23</td>
</tr>
<tr>
<td>cce</td>
<td>25</td>
</tr>
<tr>
<td>cce.factor</td>
<td>34</td>
</tr>
<tr>
<td>Class for Quasi Maximum Likelihood Estimation of Point Process Regression Models</td>
<td>40</td>
</tr>
<tr>
<td>cogarch.est.-class</td>
<td>41</td>
</tr>
<tr>
<td>cogarch.est.incr-class</td>
<td>41</td>
</tr>
<tr>
<td>cogarch.info-class</td>
<td>42</td>
</tr>
<tr>
<td>cogarchNoise</td>
<td>43</td>
</tr>
<tr>
<td>CPoint</td>
<td>44</td>
</tr>
<tr>
<td>DataPPR</td>
<td>48</td>
</tr>
<tr>
<td>Diagnostic.Carma</td>
<td>50</td>
</tr>
<tr>
<td>Diagnostic.Cogarch</td>
<td>51</td>
</tr>
<tr>
<td>estimation_LRM</td>
<td>52</td>
</tr>
<tr>
<td>EstimCarmaHawkes</td>
<td>53</td>
</tr>
<tr>
<td>fitCIR</td>
<td>55</td>
</tr>
<tr>
<td>FromCF2yuima_law</td>
<td>56</td>
</tr>
<tr>
<td>get.counting.data</td>
<td>57</td>
</tr>
<tr>
<td>gmm</td>
<td>60</td>
</tr>
<tr>
<td>hyavar</td>
<td>62</td>
</tr>
<tr>
<td>IC</td>
<td>65</td>
</tr>
<tr>
<td>info.Map-class</td>
<td>69</td>
</tr>
<tr>
<td>info.PPR</td>
<td>69</td>
</tr>
<tr>
<td>Integral.sde</td>
<td>69</td>
</tr>
<tr>
<td>Integrand</td>
<td>70</td>
</tr>
<tr>
<td>Intensity.PPR</td>
<td>70</td>
</tr>
<tr>
<td>JBtest</td>
<td>71</td>
</tr>
<tr>
<td>lambdaFromData</td>
<td>72</td>
</tr>
<tr>
<td>lasso</td>
<td>73</td>
</tr>
<tr>
<td>LawMethods</td>
<td>74</td>
</tr>
<tr>
<td>limiting.gamma</td>
<td>75</td>
</tr>
<tr>
<td>llag</td>
<td>77</td>
</tr>
<tr>
<td>llag.test</td>
<td>81</td>
</tr>
<tr>
<td>lm.jumptest</td>
<td>84</td>
</tr>
<tr>
<td>LogSPX</td>
<td>85</td>
</tr>
<tr>
<td>lseBayes</td>
<td>86</td>
</tr>
<tr>
<td>mllag</td>
<td>88</td>
</tr>
<tr>
<td>mmfrac</td>
<td>92</td>
</tr>
<tr>
<td>model.parameter-class</td>
<td>93</td>
</tr>
<tr>
<td>mpy</td>
<td>94</td>
</tr>
<tr>
<td>MWK151</td>
<td>96</td>
</tr>
<tr>
<td>noisy.sampling</td>
<td>97</td>
</tr>
<tr>
<td>ntv</td>
<td>99</td>
</tr>
<tr>
<td>param.Integral</td>
<td>101</td>
</tr>
</tbody>
</table>
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>param.Map-class</td>
<td>102</td>
</tr>
<tr>
<td>phi.test</td>
<td>102</td>
</tr>
<tr>
<td>poisson.random.sampling</td>
<td>103</td>
</tr>
<tr>
<td>pz.test</td>
<td>104</td>
</tr>
<tr>
<td>qyz</td>
<td>106</td>
</tr>
<tr>
<td>qmle</td>
<td>108</td>
</tr>
<tr>
<td>qmleLevy</td>
<td>116</td>
</tr>
<tr>
<td>rconst</td>
<td>119</td>
</tr>
<tr>
<td>rng</td>
<td>120</td>
</tr>
<tr>
<td>setCarma</td>
<td>124</td>
</tr>
<tr>
<td>setCarmaHawkes</td>
<td>127</td>
</tr>
<tr>
<td>setCharacteristic</td>
<td>129</td>
</tr>
<tr>
<td>setCogarch</td>
<td>130</td>
</tr>
<tr>
<td>setData</td>
<td>132</td>
</tr>
<tr>
<td>setFunctional</td>
<td>133</td>
</tr>
<tr>
<td>setHawkes</td>
<td>135</td>
</tr>
<tr>
<td>setIntegral</td>
<td>136</td>
</tr>
<tr>
<td>setLaw</td>
<td>138</td>
</tr>
<tr>
<td>setLaw_th</td>
<td>139</td>
</tr>
<tr>
<td>setLRM</td>
<td>140</td>
</tr>
<tr>
<td>setMap</td>
<td>141</td>
</tr>
<tr>
<td>setModel</td>
<td>142</td>
</tr>
<tr>
<td>setPoisson</td>
<td>145</td>
</tr>
<tr>
<td>setPPR</td>
<td>147</td>
</tr>
<tr>
<td>setSampling</td>
<td>148</td>
</tr>
<tr>
<td>setYuima</td>
<td>150</td>
</tr>
<tr>
<td>simBmllag</td>
<td>152</td>
</tr>
<tr>
<td>simCIR</td>
<td>155</td>
</tr>
<tr>
<td>simFunctional</td>
<td>156</td>
</tr>
<tr>
<td>simulate</td>
<td>157</td>
</tr>
<tr>
<td>snr</td>
<td>164</td>
</tr>
<tr>
<td>spectralcov</td>
<td>166</td>
</tr>
<tr>
<td>subsampling</td>
<td>170</td>
</tr>
<tr>
<td>toLatex</td>
<td>172</td>
</tr>
<tr>
<td>variable.Integral</td>
<td>173</td>
</tr>
<tr>
<td>wllag</td>
<td>174</td>
</tr>
<tr>
<td>ybook</td>
<td>177</td>
</tr>
<tr>
<td>yuima-class</td>
<td>177</td>
</tr>
<tr>
<td>yuima.ae-class</td>
<td>178</td>
</tr>
<tr>
<td>yuima.carma-class</td>
<td>179</td>
</tr>
<tr>
<td>yuima.carma.qml-class</td>
<td>180</td>
</tr>
<tr>
<td>yuima.carmaHawkes-class</td>
<td>181</td>
</tr>
<tr>
<td>yuima.characteristic-class</td>
<td>182</td>
</tr>
<tr>
<td>yuima.cogarch-class</td>
<td>182</td>
</tr>
<tr>
<td>yuima.CP.qml-class</td>
<td>183</td>
</tr>
<tr>
<td>yuima.data-class</td>
<td>184</td>
</tr>
<tr>
<td>yuima.functional-class</td>
<td>185</td>
</tr>
<tr>
<td>yuima.Hawkes</td>
<td>186</td>
</tr>
</tbody>
</table>
**Description**

The `adaBayes` class is a class of the `yuima` package that extends the `mle-class`.

**Usage**

```r
adaBayes(yuima, start, prior, lower, upper, method = "mcmc", iteration = NULL, mcmc, rate = 1, rcpp = TRUE, algorithm = "randomwalk", center = NULL, sd = NULL, rho = NULL, path = FALSE)
```

**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.
- `iteration`: number of iteration of Markov chain Monte Carlo method.
- `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.
- `rcpp`: Logical value. If `rcpp = TRUE` (default), Rcpp code will be performed. Otherwise, usual R code will be performed.
- `algorithm`: If `algorithm = "randomwalk"` (default), the random-walk Metropolis algorithm will be performed. If `algorithm = "MpCN"`, the Mixed preconditioned Crank-Nicolson algorithm will be performed.
### adaBayes

- **center**: A list of parameters used to center MpCN algorithm.
- **sd**: A list for specifying the standard deviation of proposal distributions.
- **path**: Logical value when `method = "mcmc"`. If `path=TRUE`, then the sample path for each variable will be included in the MCMC object in the output.
- **rho**: A parameter used for MpCN algorithm.

**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.

**Slots**

- **mcmc**: is a list of MCMC objects for all estimated parameters.
- **accept_rate**: is a list acceptance rates for diffusion and drift parts.
- **call**: is an object of class `language`.
- **fullcoef**: is an object of class `list` that contains estimated parameters.
- **vcov**: is an object of class `matrix`.
- **coefficients**: is an object of class `vector` that contains estimated parameters.

**Note**

`algorithm = nomcmc` is unstable.

**Author(s)**

Kengo Kamatani with YUIMA project Team

**References**


**Examples**

```r
# 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)
```
ae

Asymptotic Expansion

Description

Asymptotic expansion of uni-dimensional and multi-dimensional diffusion processes.

Usage

```
ae(
  model, 
  xinit, 
  order = 1L, 
  ...
)
```
true.parameter = list(),
sampling = NULL,
eps.var = "eps",
solver = "rk4",
verbose = FALSE
)

Arguments

model an object of yuima-class or yuima.model-class.
xinit initial value vector of state variables.
order integer. The asymptotic expansion order. Higher orders lead to better approxi-
mations but longer computational times.
true.parameter named list of parameters.
sampling a yuima.sampling-class object.
eps.var character. The perturbation variable.
solver the solver for ordinary differential equations. One of "rk4" (more accurate) or
"euler" (faster).
verbose logical. Print on progress? Default FALSE.

Details

If sampling is not provided, then model must be an object of yuima-class with non-empty
sampling.

if eps.var does not appear in the model specification, then it is internally added in front of the
diffusion matrix to apply the asymptotic expansion scheme.

Value

An object of yuima.ae-class

Author(s)

Emanuele Guidotti <emanuele.guidotti@unine.ch>

Examples

## Not run:
# model
gbm <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# settings
xinit <- 100
par <- list(mu = 0.01, sigma = 0.2)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 4, true.parameter = par, xinit = xinit)
# exact density
x <- seq(50, 200, by = 0.1)
extact <- dlnorm(x = x, meanlog = log(xinit)+(par$mu-0.5*par$sigma^2)*1, sdlog = par$sigma*sqrt(1))

# compare
plot(x, exact, type = "l", ylab = "Density")
lines(x, aeDensity(x = x, ae = approx, order = 1), col = 2)
lines(x, aeDensity(x = x, ae = approx, order = 2), col = 3)
lines(x, aeDensity(x = x, ae = approx, order = 3), col = 4)
lines(x, aeDensity(x = x, ae = approx, order = 4), col = 5)

## End(Not run)

---

**aeCharacteristic**

Asymptotic Expansion - Characteristic Function

**Description**

Asymptotic Expansion - Characteristic Function

**Usage**

```r
aeCharacteristic(..., ae, eps = 1, order = NULL)
```

**Arguments**

- `...`: named argument, data.frame, list, or environment specifying the grid to evaluate the characteristic function. See examples.
- `ae`: an object of class `yuima.ae-class`.
- `eps`: numeric. The intensity of the perturbation.
- `order`: integer. The expansion order. If NULL (default), it uses the maximum order used in `ae`.

**Value**

Characteristic function evaluated on the given grid.

**Examples**

```r
## Not run:
# model
gbm <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# settings
xinit <- 100
par <- list(mu = 0.01, sigma = 0.2)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)
```
# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 4, true.parameter = par, xinit = xinit)

# The following are all equivalent methods to specify the grid via ....
# Notice that the character 'u1' corresponds to the 'u.var' of the ae object.
approx@u.var

# 1) named argument
u1 <- seq(0, 1, by = 0.1)
psi <- aeCharacteristic(u1 = u1, ae = approx, order = 4)

# 2) data frame
df <- data.frame(u1 = seq(0, 1, by = 0.1))
psi <- aeCharacteristic(df, ae = approx, order = 4)

# 3) environment
e <- new.env()
e$u1 <- seq(0, 1, by = 0.1)
psi <- aeCharacteristic(env, ae = approx, order = 4)

# 4) list
lst <- list(u1 = seq(0, 1, by = 0.1))
psi <- aeCharacteristic(lst, ae = approx, order = 4)

## End(Not run)

---

### aeDensity

#### Asymptotic Expansion - Density

**Description**

Asymptotic Expansion - Density

**Usage**

aeDensity(..., ae, eps = 1, order = NULL)

**Arguments**

- **...**
  - named argument, data.frame, list, or environment specifying the grid to evaluate the density. See examples.
- **ae**
  - an object of class `yuima.ae-class`.
- **eps**
  - numeric. The intensity of the perturbation.
- **order**
  - integer. The expansion order. If NULL (default), it uses the maximum order used in ae.

**Value**

Probability density function evaluated on the given grid.
Examples

```r
## Not run:
# model
gbm <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# settings
xinit <- 100
par <- list(mu = 0.01, sigma = 0.2)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 4, true.parameter = par, xinit = xinit)

# The following are all equivalent methods to specify the grid via ....
# Notice that the character 'x' corresponds to the solve.variable of the yuima model.

# 1) named argument
x <- seq(50, 200, by = 0.1)
density <- aeDensity(x = x, ae = approx, order = 4)

# 2) data frame
df <- data.frame(x = seq(50, 200, by = 0.1))
density <- aeDensity(df, ae = approx, order = 4)

# 3) environment
env <- new.env()
env$x <- seq(50, 200, by = 0.1)
density <- aeDensity(env, ae = approx, order = 4)

# 4) list
lst <- list(x = seq(50, 200, by = 0.1))
density <- aeDensity(lst, ae = approx, order = 4)

# exact density
exact <- dlnorm(x = x, meanlog = log(xinit)+(par$mu-0.5*par$sigma^2)*1, sdlog = par$sigma*sqrt(1))

# compare
plot(x = exact, y = density, xlab = "Exact", ylab = "Approximated")

## End(Not run)
```

---

### Description

Compute the expected value of functionals.

### Usage

```r
aeExpectation(f, bounds, ae, eps = 1, order = NULL, ...)
```
### Arguments

- **f** character. The functional.
- **bounds** named list of integration bounds in the form `list(x = c(xmin, xmax), y = c(ymin, ymax), ...)`
- **ae** an object of class `yuima.ae-class`.
- **eps** numeric. The intensity of the perturbation.
- **order** integer. The expansion order. If NULL (default), it uses the maximum order used in `ae`.
- `...` additional arguments passed to `cubintegrate`.

### Value

return value of `cubintegrate`. The expectation of the functional provided.

### Examples

```r
## Not run:
# model
gbm <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# settings
xinit <- 100
par <- list(mu = 0.01, sigma = 0.2)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 4, true.parameter = par, xinit = xinit)

# compute the mean via integration
aeExpectation(f = 'x', bounds = list(x = c(0,1000)), ae = approx)

# compare with the mean computed by differentiation of the characteristic function
aeMean(approx)
## End(Not run)
```

### Description

Asymptotic Expansion - Kurtosis

### Usage

```r
aeKurtosis(ae, eps = 1, order = NULL)
```
Arguments

ae an object of class `yuima.ae-class`.
eps numeric. The intensity of the perturbation.
order integer. The expansion order. If NULL (default), it uses the maximum order used in ae.

Value

numeric.

Examples

```r
## Not run:
# model
gbm <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# settings
xinit <- 100
par <- list(mu = 0.01, sigma = 0.2)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 4, true.parameter = par, xinit = xinit)

# expansion order max
aeKurtosis(ae = approx)

# expansion order 1
aeKurtosis(ae = approx, order = 1)

## End(Not run)
```

---

**aeMarginal**

Asymptotic Expansion - Marginals

**Description**

Asymptotic Expansion - Marginals

**Usage**

`aeMarginal(ae, var)`

**Arguments**

| ae     | an object of class `yuima.ae-class`. |
| var    | variables of the marginal distribution to compute. |
Value

An object of *yuima.ae-class*

Examples

```r
## Not run:
# multidimensional model
gbm <- setModel(drift = c('mu*x1','mu*x2'),
    diffusion = matrix(c('sigma1*x1',0,0,'sigma2*x2'), nrow = 2),
    solve.variable = c('x1','x2'))

# settings
xinit <- c(100,100)
par <- list(mu = 0.01, sigma1 = 0.2, sigma2 = 0.1)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 3, true.parameter = par, xinit = xinit)

# extract marginals
margin1 <- aeMarginal(ae = approx, var = "x1")
margin2 <- aeMarginal(ae = approx, var = "x2")

# compare with exact solution for marginal 1
x1 <- seq(50, 200, by = 0.1)
exact <- dlnorm(x = x1, meanlog = log(xinit[1])+(par$mu-0.5*par$sigma1^2), sdlog = par$sigma1)
plot(x1, exact, type = 'p', ylab = "Density")
lines(x1, aeDensity(x1 = x1, ae = margin1, order = 3), col = 2)

# compare with exact solution for marginal 2
x2 <- seq(50, 200, by = 0.1)
exact <- dlnorm(x = x2, meanlog = log(xinit[2])+(par$mu-0.5*par$sigma2^2), sdlog = par$sigma2)
plot(x2, exact, type = 'p', ylab = "Density")
lines(x2, aeDensity(x2 = x2, ae = margin2, order = 3), col = 2)

## End(Not run)
```

---

**aeMean**

### Asymptotic Expansion - Mean

**Description**

Asymptotic Expansion - Mean

**Usage**

```r
aeMean(ae, eps = 1, order = NULL)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ae</td>
<td>an object of class \texttt{yuima.ae-class}.</td>
</tr>
<tr>
<td>eps</td>
<td>numeric. The intensity of the perturbation.</td>
</tr>
<tr>
<td>order</td>
<td>integer. The expansion order. If NULL (default), it uses the maximum order used in ae.</td>
</tr>
</tbody>
</table>

Value

numeric.

Examples

```r
## Not run:
# model
gbm <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# settings
xinit <- 100
par <- list(mu = 0.01, sigma = 0.2)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 4, true.parameter = par, xinit = xinit)

# expansion order max
aeMean(ae = approx)

# expansion order 1
aeMean(ae = approx, order = 1)

## End(Not run)
```

Description

Asymptotic Expansion - Moments

Usage

```r
aeMoment(ae, m = 1, eps = 1, order = NULL)
```
Arguments

ae an object of class `yuima.ae-class`.
m integer. The moment order. In case of multidimensional processes, it is possible to compute cross-moments by providing a vector of the same length as the state variables.
eps numeric. The intensity of the perturbation.
order integer. The expansion order. If NULL (default), it uses the maximum order used in ae.

Value

numeric.

Examples

```r
## Not run:
# model
gbm <- setModel(drift = "mu*x", diffusion = "sigma*x", solve.variable = 'x')

# settings
xinit <- 100
par <- list(mu = 0.01, sigma = 0.2)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 4, true.parameter = par, xinit = xinit)

# second moment, expansion order max
aeMoment(ae = approx, m = 2)

# second moment, expansion order 3
aeMoment(ae = approx, m = 2, order = 3)

# second moment, expansion order 2
aeMoment(ae = approx, m = 2, order = 2)

# second moment, expansion order 1
aeMoment(ae = approx, m = 2, order = 1)

## End(Not run)
```
Usage

aeSd(ae, eps = 1, order = NULL)

Arguments

- `ae`: an object of class `yuima.ae-class`.
- `eps`: numeric. The intensity of the perturbation.
- `order`: integer. The expansion order. If NULL (default), it uses the maximum order used in `ae`.

Value

numeric.

Examples

```r
## Not run:
# model
gbm <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# settings
xinit <- 100
par <- list(mu = 0.01, sigma = 0.2)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 4, true.parameter = par, xinit = xinit)

# expansion order max
aeSd(ae = approx)

# expansion order 1
aeSd(ae = approx, order = 1)

## End(Not run)
```

---

**aeSkewness**

**Asymptotic Expansion - Skewness**

Description

Asymptotic Expansion - Skewness

Usage

```r
aeSkewness(ae, eps = 1, order = NULL)
```
Arguments

- **ae**: an object of class `yuima.ae-class`.
- **eps**: numeric. The intensity of the perturbation.
- **order**: integer. The expansion order. If NULL (default), it uses the maximum order used in ae.

Value

numeric.

Examples

```
## Not run:
# model
gbm <- setModel(drift = 'mu*x', diffusion = 'sigma*x', solve.variable = 'x')

# settings
xinit <- 100
par <- list(mu = 0.01, sigma = 0.2)
sampling <- setSampling(Initial = 0, Terminal = 1, n = 1000)

# asymptotic expansion
approx <- ae(model = gbm, sampling = sampling, order = 4, true.parameter = par, xinit = xinit)

# expansion order max
aeSkewness(ae = approx)

# expansion order 1
aeSkewness(ae = approx, order = 1)
```

---

**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 + 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

```r
## Not run:
# 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)
asymp <- asymptotic_term(yuima, block=10, rho, g)
asymp
sum(asymp$d0 + e * asymp$d1)

### An example of multivariate case: Heston model
## a <- 1; C <- 1; d <- 10; R<-.1
## diff.matrix <- matrix( c("x1*sqrt(x2)*e", "e*R*sqrt(x2)",0,"sqrt(x2*(1-R^2))*e"), 2,2)
## model <- setModel(drift = c("a*x1","C*(10-x2)"),
## diffusion = diff.matrix, solve.variable=c("x1","x2"), state.variable=c("x1","x2"))
## call option is evaluated by averaging
## 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)
##
## End(Not run)

---

**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 component.
Usage

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

Arguments

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

Details

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

Value

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

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

Note

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

Author(s)

Yuta Koike with YUIMA Project Team

References


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

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Number of autoregressive coefficients.</td>
</tr>
<tr>
<td>q</td>
<td>Number of moving average coefficients.</td>
</tr>
<tr>
<td>loc.par</td>
<td>Label of location coefficient.</td>
</tr>
<tr>
<td>scale.par</td>
<td>Label of scale coefficient.</td>
</tr>
<tr>
<td>ar.par</td>
<td>Label of autoregressive coefficients.</td>
</tr>
<tr>
<td>ma.par</td>
<td>Label of moving average coefficients.</td>
</tr>
<tr>
<td>lin.par</td>
<td>Label of linear coefficients.</td>
</tr>
<tr>
<td>Carma.var</td>
<td>Label of the observed process.</td>
</tr>
<tr>
<td>Latent.var</td>
<td>Label of the unobserved process.</td>
</tr>
<tr>
<td>XinExpr</td>
<td>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.</td>
</tr>
</tbody>
</table>

Author(s)

The YUIMA Project Team

---

carmaHawkes.info-class

*Class for information on the Hawkes process with a CARMA(p,q) intensity*

Description

The carmaHawkes.info-class is a class of the yuima package.

Details

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

Slots

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Number of autoregressive coefficients.</td>
</tr>
<tr>
<td>q</td>
<td>Number of moving average coefficients.</td>
</tr>
<tr>
<td>base.Int</td>
<td>Label of baseline Intensity parameter.</td>
</tr>
<tr>
<td>ar.par</td>
<td>Label of autoregressive coefficients.</td>
</tr>
</tbody>
</table>
ma.par: Label of moving average coefficients.
Intensity.var: Label of the Intensity 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.
Type.Jump: Logical variable. If XinExpr=TRUE, the jump size is deterministic

Author(s)
The YUIMA Project Team
Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>

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

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.

```r
cce
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)
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, fregion = 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 `yuima-class` or `yuima.data-class`.
- **method**: the method to be used. See ‘Details’.
- **theta**: a numeric vector or matrix. If it is a matrix, each of its components indicates the tuning parameter which determines the pre-averaging window lengths `kn` to be used for estimating the corresponding component. If it is a numeric vector, it is converted to a matrix as `(C+t(C))/2`, where `C=matrix(theta,d,d)` and `d=dim(x)`. The default value is 0.15 for the method “PHY” or “PTHY” following Christensen et al. (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 `ceiling(c.two*n^(2/3))`, where `n` is the number of the refresh times associated with the data minus 1.
- **c.two**: a positive number indicating the tuning parameter which determines the scale of the large time-scale parameter `K`. The default value is the average of the numeric vector each of whose components is the roughly estimated optimal value in the sense of the minimizer of the theoretical asymptotic variance of the estimator of...
the corresponding diagonal component. The theoretical asymptotic variance is considered in the standard case and given by Eq.(63) of Zhang et al. (2005).

\( J \)

a positive integer indicating the small time-scale parameter.

\( \text{c.multi} \)
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 \((C + \text{t}(C))/2\), where \(C=\text{matrix}(\text{c.multi},d,d)\) 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).

\( \text{kernel} \)
a function indicating the kernel function to be used. The default value is the Parzen kernel, which is recommended in Barndorff-Nielsen et al. (2009, 2011).

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

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

\( \eta \)
a positive number indicating the tuning parameter which determines the order of the bandwidth parameter \(H\).

\( m \)
a positive integer indicating the number of the end points to be jittered.

\( \text{ftregion} \)
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.

\( \text{vol.init} \)
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.

\( \text{covol.init} \)
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.

\( \text{nvar.init} \)
a numeric vector each of whose components indicates the initial value to be used to estimate the variance of noise of the corresponding component, which is passed to the optimizer.

\( \text{ncov.init} \)
a numeric matrix each of whose columns indicates the initial value to be used to estimate the covariance of noise of the corresponding component, which is passed to the optimizer.

\( \text{mn} \)
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.
alpha  
a positive number indicating the order of \( mn \).

frequency  
a positive integer indicating the frequency (seconds) of the calendar time sampling to be used.

avg  
logical. If TRUE, the averaged subsampling estimator is calculated. Otherwise the simple sparsely subsampled estimator is calculated.

threshold  
a numeric vector or list indicating the threshold parameter(s). Each of its components indicates the threshold parameter or process to be used for estimating the corresponding component. If it is a numeric vector, the elements in threshold are recycled if there are two few elements in threshold. The default value is determined following Koike (2014) (for the method "THY") and Koike (2015) (for the method "PTHY").

utime  
a positive number indicating what seconds the interval \([0,1]\) corresponds to. The default value is the difference between the maximum and the minimum of the sampling times, multiplied by 23,400. Here, 23,400 seconds correspond to 6.5 hours, hence if the data is sampled on the interval \([0,1]\), then the sampling interval is regarded as 6.5 hours.

cce(x,psd=FALSE)  
cce(x,method="PHY",theta,tkn,kg,refreshing=TRUE,cwise=TRUE,psd=FALSE)  
cce(x,method="MRC",theta,tkn,kg,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,tkn,kg,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 $k_n$ changes depending on the values of refreshing and cwise. If both refreshing and cwise are TRUE (the default), the default value of $k_n$ is given by the matrix $\text{ceiling}(\theta N)$, 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 $k_n$ is given by $\text{ceiling}(\text{mean}(\theta)\sqrt{n})$, where $n$ is the number of the refresh times associated with the data minus 1. If refreshing is FALSE while cwise is TRUE, the default value of $k_n$ is given by the matrix $\text{ceiling}(\theta N_0)$, 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 $k_n$ is given by $\text{ceiling}(\text{mean}(\theta)\sqrt{\sum(\text{length}(x)-1)})$ (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:

- covmat: the estimated covariance matrix
- cormat: the estimated correlation matrix

Note

The example shows the central limit theorem for the nonsynchronous covariance estimator. Estimation of the asymptotic variance can be implemented by `hyavar`. The second-order correction will be provided in a future version of the package.

Author(s)

Yuta Koike with YUIMA Project Team

References


See Also

setModel, setData, hyavar, lmm, cce.factor

Examples

```r
## Not run:
## 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.
```
## True value of the quadratic covariation.

cce.th <- function(T, sigma1, sigma2, rho) {
  tmp <- function(t) return(sigma1(t) * sigma2(t) * rho(t))
  integrate(tmp, 0, T)
}

theta <- cce.th(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

cat("Set seed(111)\n")
yuima.samp <- setSampling(Terminal = 1, n = 10000)
yuima <- setYuima(model = cor.mod, sampling = yuima.samp)
yuima <- simulate(yuima, xinit=c(2,3))
plot(yuima)

cce(yuima)

## We use the function poisson.random.sampling to generate nonsynchronous
## observations by Poisson sampling.
p-sample <- 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
```r
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
```r
set.seed(116)
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
```r
psample <- poisson.random.sampling(yuima, rate = c(1/5,1/5), n = 11700)
```

## Hayashi-Yoshida estimator consistetly 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
```r
suppressWarnings(s1 <- cce(subsampling(psample, sampling = setSampling(n = 4680)))$cormat[1,2])
s1
```

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

## Synchronization on the grid with 20 seconds steps
```r
suppressWarnings(s3 <- cce(subsampling(psample, sampling = setSampling(n = 1170)))$cormat[1,2])
s3
```
### Synchronization on the grid with 30 seconds steps

```r
cce(subsampling(psample, sampling = setSampling(n = 780)))$cormat[1,2]
```

### Synchronization on the grid with 1 minute steps

```r
cce(subsampling(psample, sampling = setSampling(n = 390)))$cormat[1,2]
```

```r
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

```r
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)
```

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

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

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

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

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

## End(Not run)

---

**Description**

This function estimates the covariance and precision matrices of a high-dimensional Ito process by factor modeling and regularization when it is observed at discrete times possibly nonsynchronously with noise.

**Usage**

```r
cce.factor(yuima, method = "HY", factor = NULL, PCA = FALSE,
```
Arguments

yuima  
an object of yuima-class or yuima.data-class.

method  
the method to be used in cce.

factor  
an integer or character vector indicating which components of yuima are factors. If NULL, no factor structure is taken account of.

PCA  
logical. If TRUE, a principal component analysis is performed to construct factors.

nfactor  
the number of factors constructed when PCA is TRUE. If nfactor = "interactive", the scree plot of the principal component analysis is depicted and the user can set this argument interactively.

regularize  
the regularizaton method to be used. Possible choices are "glasso" (the default), "tapering", "thresholding" and "eigen.cleaning". See 'Details'.

taper  
the tapering matrix used when regularize = "tapering". If missing, the tapering matrix is constructed according to group. See 'Details'.

group  
an integer vector having the length equal to dim(yuima)-length(factor).

lambda  
the penalty parameter used when regularize = "glasso". If it is "aic" (resp. "bic"), it is selected by minimizing the formally defined AIC (resp. BIC). See 'Details'.

weight  
logical. If TRUE, a weighted version is used for regularize = "glasso" as in Koike (2020).

nlambda  
a positive integer indicating the number of candidate penalty parameters for which AIC or BIC is evaluated when lambda is "aic" or "bic".

ratio  
a positive number indicating the ratio of the largest and smallest values in candidate penalty parameters for which AIC or BIC is evaluated when lambda is "aic" or "bic". See 'Details'. The default value is sqrt(log(d)/N), where d is the dimension of yuima.

N  
a positive integer indicating the "effective" sampling size, which is necessary to evaluate AIC and BIC when lambda is "aic" or "bic". In a standard situation, it is equal to the sample size − 1, but it might be different when the data are observed nonsynchronously and/or with noise. If missing, it is automatically determined according to method.

thr.type  
a character string indicating the type of the thresholding method used when regularize = "thresholding". Possible choices are "hard", "soft", "alasso" and "scad". See Section 2.3 of Dai et al. (2019) for the definition of each method.

thr  
a numeric matrix indicating the threshold levels used when regularize = "thresholding". Its entries indicate the threshold levels for the corresponding entries of the covariance matrix (values for λ in the notation of Dai et al. (2019)). A single
number is converted to the matrix with common entries equal to that number. If NULL, it is determined according to tau. See ‘Details’.

tau

A number between 0 and 1 used to determine the threshold levels used when regularize = "thresholding" and thr=NULL (a value for \( \tau \) in the notation of Dai et al. (2019)). If NULL, it is determined by a grid search procedure as suggested in Section 4.3 of Dai et al. (2019). See ‘Details’.

par.lasso

The tuning parameter for thr.type = "lasso" (a value for \( \eta \) in the notation of Dai et al. (2019)).

par.scad

The tuning parameter for thr.type = "scad" (a value for \( a \) in the notation of Dai et al. (2019)).

thr.delta

A positive number indicating the step size used in the grid search procedure to determine tau.

frequency

Passed to cce.

utime

Passed to cce.

... Passed to cce.

Details

One basic approach to estimate the covariance matrix of high-dimensional time series is to take account of the factor structure and perform regularization for the residual covariance matrix. This function implements such an estimation procedure for high-frequency data modeled as a discretely observed semimartingale. Specifically, let \( Y \) be a \( d \)-dimensional semimartingale which describes the dynamics of the observation data. We consider the following continuous-time factor model:

\[
Y_t = \beta X_t + Z_t, \quad 0 \leq t \leq T,
\]

where \( X \) is an \( r \)-dimensional semimartingale (the factor process), \( Z \) is a \( d \)-dimensional semimartingale (the residual process), and \( \beta \) is a constant \( d \times r \) matrix (the factor loading matrix). We assume that \( X \) and \( Z \) are orthogonal in the sense that \( \langle X, Z \rangle_T = 0 \). Then, the quadratic covariation matrix of \( Y \) is given by

\[
\langle Y, Y \rangle_T = \beta \langle X, X \rangle_T \beta^T + \langle Z, Z \rangle_T.
\]

Also, \( \beta \) can be written as \( \beta = [Y, X]_T [X, X]_T^{-1} \). Thus, if we have observation data both for \( Y \) and \( X \), we can construct estimators for \( \langle Y, Y \rangle_T \), \( \langle X, X \rangle_T \) and \( \beta \) by cce. Moreover, plugging these estimators into the above equation, we can also construct an estimator for \( \langle Z, Z \rangle_T \). Since this estimator is often poor due to the high-dimensionality, we regularize it by some method. Then, by plugging the regularized estimator for \( \langle Z, Z \rangle_T \) into the above equation, we obtain the final estimator for \( \langle Y, Y \rangle_T \).

Even if we do not have observation data for \( X \), we can (at least formally) construct a pseudo factor process by performing principal component analysis for the initial estimator of \( \langle Y, Y \rangle_T \). See Ait-Sahalia and Xiu (2017) and Dai et al. (2019) for details.

Currently, the following four options are available for the regularization method applied to the residual covariance matrix estimate:

1. regularize = "glasso" (the default).

This performs the graphical Lasso. When weight=TRUE (the default), a weighted version of the graphical Lasso is performed as in Koike (2020). Otherwise, the standard graphical Lasso is performed as in Brownlees et al. (2018).
If \( \text{lambda} = \text{"aic" (resp. \text{"bic")}, the penalty parameter for the graphical Lasso is selected by minimizing the formally defined AIC \text{ resp. BIC). The minimization is carried out by grid search, where the grid is determined as in Section 5.1 of Koike (2020). The optimization problem in the graphical Lasso is solved by the GLASSOFAST algorithm of Sustik and Calderhead (2012), which is available from the package \text{glassoFast}.}

2. \text{regularize = \text{"tapering"}.}

This performs tapering, i.e. taking the entry-wise product of the residual covariance matrix estimate and a tapering matrix specified by \text{taper}. See Section 3.5.1 of Pourahmadi (2011) for an overview of this method.

If \text{taper} is missing, it is constructed according to \text{group} as follows: \text{taper} is a 0-1 matrix and the \((i,j)\)-th entry is equal to 1 if and only if \text{group}[i]==\text{group}[j]. Thus, by default it makes the residual covariance matrix diagonal.

3. \text{regularize = \text{"thresholding"}.}

This performs thresholding, i.e. entries of the residual covariance matrix are shrunk toward 0 according to a thresholding rule (specified by \text{thr.type}) and a threshold level (specified by \text{thr}).

If \text{thr=\text{NULL}}, the \((i,j)\)-th entry of \text{thr} is given by \(\tau \sqrt{[Z^i_i, Z^i_i][Z^j_j, Z^j_j]}\), where \([Z^i_i, Z^i_i][Z^j_j, Z^j_j]_T\) denotes the \(i\)-th \text{ resp. } \(j\)-th) diagonal entry of the non-regularized estimator for the residual covariance matrix \([Z, Z]'_T\), and \(\tau\) is a tuning parameter specified by \text{tau}.

When \text{tau=\text{NULL}}, the value of \(\tau\) is set to the smallest value in the grid with step size \text{thr.delta} such that the regularized estimate of the residual covariance matrix becomes positive definite.

4. \text{regularize = \text{"eigen.cleaning"}.}

This performs the eigenvalue cleaning algorithm described in Hautsch et al. (2012).

Value

A list with components:

- \text{covmat.y} the estimated covariance matrix
- \text{premat.y} the estimated precision matrix
- \text{beta.hat} the estimated factor loading matrix
- \text{covmat.x} the estimated factor covariance matrix
- \text{covmat.z} the estimated residual covariance matrix
- \text{premat.z} the estimated residual precision matrix
- \text{sigma.z} the estimated residual covariance matrix before regularization
- \text{pc} the variances of the principal components (it is \text{NULL} if \text{PCA = FALSE})

Author(s)

Yuta Koike with YUIMA project Team
References


See Also

`cce, lmm, glassoFast`

Examples

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

## Simulating a factor process (Heston model)
drift <- c("mu*S", "-theta*(V-v)"
diffusion <- matrix(c("sqrt(max(V,0))*S", "gamma*sqrt(max(V,0))*rho", 0, "gamma*sqrt(max(V,0))*sqrt(1-rho^2)"),
                   2,2)
mod <- setModel(drift = drift, diffusion = diffusion,
                state.variable = c("S", "V"))
n <- 2340
samp <- setSampling(n = n)
heston <- setYuima(model = mod, sampling = samp)
param <- list(mu = 0.03, theta = 3, v = 0.09,
               gamma = 0.3, rho = -0.6)
result <- simulate(heston, xinit = c(1, 0.1),
                   true.parameter = param)

zdata <- get.zoo.data(result) # extract the zoo data
X <- log(zdata[[1]]) # log-price process
V <- zdata[[2]] # squared volatility process

## Simulating a residual process (correlated BM)
d <- 100 # dimension
Q <- 0.1 * toeplitz(0.7^(1:d-1)) # residual covariance matrix
dZ <- matrix(rnorm(n*d),n,d) %*% chol(Q)/sqrt(n)
```
Z <- zoo(apply(dZ, 2, "diffinv"), samp@grid[[1]])

## Constructing observation data
b <- runif(d, 0.25, 2.25) # factor loadings
Y <- X %o% b + Z
yuima <- setData(cbind(X, Y))

# We subsample yuima to construct observation data
yuima <- subsampling(yuima, setSampling(n = 78))

## Estimating the covariance matrix (factor is known)
cmat <- tcrossprod(b) * mean(V[-1]) + Q # true covariance matrix
pmat <- solve(cmat) # true precision matrix

# (1) Regularization method is glasso (the default)
est <- cce.factor(yuima, factor = 1)
norm(est$covmat.y - cmat, type = "2")
norm(est$premat.y - pmat, type = "2")

# (2) Regularization method is tapering
est <- cce.factor(yuima, factor = 1, regularize = "tapering")
norm(est$covmat.y - cmat, type = "2")
norm(est$premat.y - pmat, type = "2")

# (3) Regularization method is thresholding
est <- cce.factor(yuima, factor = 1, regularize = "thresholding")
norm(est$covmat.y - cmat, type = "2")
norm(est$premat.y - pmat, type = "2")

# (4) Regularization method is eigen.cleaning
est <- cce.factor(yuima, factor = 1, regularize = "eigen.cleaning")
norm(est$covmat.y - cmat, type = "2")
norm(est$premat.y - pmat, type = "2")

## Estimating the covariance matrix (factor is unknown)
yuima2 <- setData(Y)

# We subsample yuima to construct observation data
yuima2 <- subsampling(yuima2, setSampling(n = 78))

# (A) Ignoring the factor structure (regularize = "glasso")
est <- cce.factor(yuima2)
norm(est$covmat.y - cmat, type = "2")
norm(est$premat.y - pmat, type = "2")

# (B) Estimating the factor by PCA (regularize = "glasso")
est <- cce.factor(yuima2, PCA = TRUE, nfactor = 1) # use 1 factor
norm(est$covmat.y - cmat, type = "2")
norm(est$premat.y - pmat, type = "2")
Class for Quasi Maximum Likelihood Estimation of Point Process Regression Models

# One can interactively select the number of factors
# after implementing PCA (the scree plot is depicted)
# Try: est <- cce.factor(yuima2, PCA = TRUE)

## End(Not run)

---

Description

The yuima.PPR.qmle class is a class of the yuima package that extends the mle-class of the stats4 package.

Slots

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.PPR-class.

Methods

Methods mle  All methods for mle-class are available.

Author(s)

The YUIMA Project Team
cogarch.est.-class

Class for Generalized Method of Moments Estimation for COGARCH(p,q) model

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 qmle 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 qmle.
**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 class `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 `qmle` 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**

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

**Arguments**

- **yuima**: a `yuima` object or an object of `yuima.cogarch-class`.
- **data**: an object of class `yuima.data-class` contains the observations available at uniformly spaced time. If `data=NULL`, the default, the `cogarchNoise` uses the data in an object of `yuima.data-class`.
- **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.
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**

**Examples**

# Insert here some examples

---

### CPoint

**Volatility structural change point estimator**

**Description**

Volatility structural change point estimator

**Usage**

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

**Arguments**

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

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

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

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

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

Value

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

Author(s)

The YUIMA Project Team

Examples

```r
## 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=0.1, theta1.2=0.2)
t2 <- list(theta1.1=0.6, theta1.2=0.6)

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
# CPoint(yuima, param1=t1, param2=t2, plot=TRUE)

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

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

tmp2 <- qmleR(yuima, start=list(theta1.1=0.3, theta1.2=0.5), t=8.5,
               lower=low, method="L-BFGS-B")
tmp2

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

# second stage estimation of parameters given first stage
# change point estimator
# CPoint(yuima, param1=coef(tmp1), param2=coef(tmp2))
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
# qmleL(yuima, start= list(theta1=0.5), t=.15, lower=lower, upper=upper, method="L-BFGS-B")
tmp1 <- qmleL(yuima, start= list(theta1=0.5), t=.15, lower=lower, upper=upper, method="L-BFGS-B")
tmp1

tmp2 <- qmleR(yuima, start= list(theta1=0.5), t=.85, lower=lower, upper=upper, 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$tau-0.1, lower=low, upper=upp, method="L-BFGS-B")
tmp11
DataPPR <- qmleR(yuima, start=as.list(coef(tmp2)), t=t.est$tau+0.1,
        lower=low, upper=upp, method="L-BFGS-B")

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

## End(Not run)

### Description

The function converts an object of class zoo to an object of class `yuima.PPR`.

### Usage

DataPPR(CountVar, yuimaPPR, samp)

### Arguments

- **CountVar**: An object of class `zoo` that contains counting variables and covariates. `index(CountVar)` returns the arrival times.
- **yuimaPPR**: An object of class `yuima.PPR` that contains a mathematical description of the point process regression model assumed to be the generator of the observed data.
- **samp**: An object of class `yuima.sampling`.

### Value

The function returns an object of class `yuima.PPR` where the slot `model` contains the Point process described in `yuimaPPR@model`, the slot `data` contains the counting variables and the covariates observed on the grid in `samp`.

### Examples

```r
## Not run:
# In this example we generate a dataset contains the Counting Variable N
# and the Covariate X.
# The covariate X is an OU driven by a Gamma process.

# Values of parameters.
mu <- 2
alpha <- 4
beta <- 5

# Law definition
```
my.rKern <- function(n,t){
  res0 <- t(t(rgamma(n, 0.1*t)))
  res1 <- t(t(rep(1,n)))
  res <- cbind(res0,res1)
  return(res)
}

Law.PPRKern <- setLaw(rng = my.rKern)

# Point Process definition
modKern <- setModel(drift = c("0.4*(0.1-X)","0"),
diffusion = c("0","0"),
jump.coeff = matrix(c("1","0","0","1"),2,2),
measure = list(df = Law.PPRKern),
measure.type = c("code","code"),
solve.variable = c("X","N"),
xinit=c("0.25","0"))

gFun <- "exp(mu*log(1+X))"

Kernel <- "alpha*exp(-beta*(t-s))"

prvKern <- setPPR(yuima = modKern,
counting.var="N", gFun=gFun,
Kernel = as.matrix(Kernel),
lambda.var = "lambda", var.dx = "N",
lower.var="0", upper.var = "t")

# Simulation
Term<-200
seed<-1
n<-20000

true.parKern <- list(mu=mu, alpha=alpha, beta=beta)

set.seed(seed)

# Simulation

time.simKern <- system.time(
  simprvKern <- simulate(object = prvKern, true.parameter = true.parKern,
sampling = setSampling(Terminal =Term, n=n))
)

plot(simprvKern,main ="Counting Process with covariates" ,cex.main=0.9)

# Using the function get.counting.data we extract from an object of class
# yuima.PPR the counting process N and the covariate X at the arrival times.
CountVar <- get.counting.data(simprvKern)
plot(CountVar)

# We convert the zoo object in the yuima.PPR object.

sim2 <- DataPPR(CountVar, yuimaPPR=simprvKern, samp=simprvKern@sampling)

## End(Not run)

---

**Diagnostic.Carma**  
*Diagnostic Carma model*

---

**Description**

This function verifies if the condition of stationarity is satisfied.

**Usage**

```r
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**

```r
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)
```
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

```r
## Not run:
# Definition of the COGARCH(1,1) process driven by a Variance Gamma nois:
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",
                      ...)```
estimation_LRM

# 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)

---

estimation_LRM  

Estimation of the t-Levy Regression Model

Description

The function estimates a t-Levy Regression Model
Usage

estimation_LRM(start, model, data, upper, lower, PT = 500, n_obs1 = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Initial values to be passed to the optimizer.</td>
</tr>
<tr>
<td>model</td>
<td>A <code>yuima.LevyRM-class</code> that contains the mathematical representation of the (t)-Levy Regression Model. Its slot <code>@data</code> can contain either real or simulated data.</td>
</tr>
<tr>
<td>data</td>
<td>An object of class <code>yuima.data-class</code> contains the observations available at uniformly spaced time. If <code>data=NULL</code>, the default, the function uses the data in the object <code>model</code>.</td>
</tr>
<tr>
<td>upper</td>
<td>A named list for specifying upper bounds of parameters.</td>
</tr>
<tr>
<td>lower</td>
<td>A named list for specifying lower bounds of parameters.</td>
</tr>
<tr>
<td>PT</td>
<td>The number of the data for the estimation of the regressor coefficients and the scale parameter.</td>
</tr>
<tr>
<td>n_obs1</td>
<td>The number of data used in the estimation of the degree of freedom. As default the number of the whole data is used in this part</td>
</tr>
</tbody>
</table>

Details

A two-step estimation procedure. Regressor coefficients and scale parameters are obtained by maximizing the quasi-likelihood function based on the Cauchy density. The degree of freedom is estimated used the unitary increment of the \(t\)-noise.

Value

Estimated parameters

Author(s)

The YUIMA Project Team

Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>

---

\(\text{EstimCarmaHawkes}\) \(\text{Estimation Methods for a CARMA}(p,q)\)-Hawkes Counting Process

Description

The function provides two estimation procedures: Maximum Likelihood Estimation and Matching Empirical Correlation

Usage

EstimCarmaHawkes(yuima, start, est.method = "qmle", method = "BFGS", lower = NULL, upper = NULL, lags = NULL, display = FALSE)
Arguments

- `yuima`: a yuima object.
- `start`: initial values to be passed to the optimizer.
- `est.method`: The method used to estimate the parameters. The default `est.method = "qmle"` indicates the MLE while the alternative approach is based on the minimization of the empirical and theoretical autocorrelation.
- `method`: The optimization method to be used. See `optim`.
- `lower`: Lower Bounds.
- `upper`: Upper Bounds.
- `lags`: Number of lags used in the autocorrelation.
- `display`: you can see a progress of the estimation when `display=TRUE`.

Value

The output contains the estimated parameters.

Author(s)

The YUIMA Project Team

Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>

References


Examples

```r
## Not run:
## MLE For A CARMA(2,1)-Hawkes ##

# Inputs:
a <- c(3,2)
b <- c(1,0.3)
mu<-0.30
true.par<-c(mu,a,b)

# step 1) Model Definition => Constructor 'setCarmaHawkes'
p <- 2
q <- 1
mod1 <- setCarmaHawkes(p = p,q = q)

# step 2) Grid Construction => Constructor 'setSampling'
FinalTime <- 5000
t0 <- 0
samp <- setSampling(t0, FinalTime, n = FinalTime)
```
# step 3) Simulation => method 'simulate'
# We use method 'simulate' to generate our dataset.
# For the estimation from real data,
# we use the constructors 'setData' and
#'setYuima' (input 'model' is an object of
#  'yuima.CarmaHawkes-class').

names(true.par) <- c(mod1@info@base.Int, mod1@info@ar.par, mod1@info@ma.par)

set.seed(1)
system.time(
sim1 <- simulate(object = mod1, true.parameter = true.par,
               sampling = samp)
)

plot(sim1)

# step 4) Estimation using the likelihood function.
system.time(
  res <- EstimCarmaHawkes(yuima = sim1,
               start = true.par)
)

### End(Not run)

---

**fitCIR**

*Calculate preliminary estimator and one-step improvements of a Cox-Ingersoll-Ross diffusion*

---

**Description**

This is a function to simulate the preliminary estimator and the corresponding one step estimators based on the Newton-Raphson and the scoring method of the Cox-Ingersoll-Ross process given via the SDE

\[ dX_t = (\alpha - \beta X_t) dt + \sqrt{\gamma X_t} dW_t \]

with parameters \( \beta > 0, 2\alpha > 5\gamma > 0 \) and a Brownian motion \((W_t)_{t \geq 0}\). This function uses the Gaussian quasi-likelihood, hence requires that data is sampled at high-frequency.

**Usage**

`fitCIR(data)`

**Arguments**

- `data` a numeric matrix containing the realization of \((t_0, X_{t_0}), \ldots, (t_n, X_{t_n})\) with \(t_j\) denoting the \(j\)-th sampling times. `data[1,]` contains the sampling times \(t_0, \ldots, t_n\) and `data[2,]` the corresponding value of the process \(X_{t_0}, \ldots, X_{t_n}\). In other words `data[,j]`=(\(t_j, X_{t_j}\)). The observations should be equidistant.
Details
The estimators calculated by this function can be found in the reference below.

Value
A list with three entries each contain a vector in the following order: The result of the preliminary estimator, Newton-Raphson method and the method of scoring.
If the sampling points are not equidistant the function will return 'Please use equidistant sampling points'.

Author(s)
Nicole Hufnagel
Contacts: <nicole.hufnagel@math.tu-dortmund.de>

References

Examples
#You can make use of the function simCIR to generate the data
data <- simCIR(alpha=3,beta=1, gamma=1, n=5000, h=0.05, equi.dist=TRUE)
results <- fitCIR(data)

FromCF2yuima_law
From a Characteristic Function to an yuima.law-object.

Description
This function returns an object of yuima.law-class and requires the characteristic function as the only input. Density, Random Number Generator, Cumulative Distribution Function and quantile function are internally constructed.

Usage
FromCF2yuima_law(myfun, time.names = "t", var_char = "u", up = 45, low = -45, N_grid = 50001, N_Fourier = 2^10)

Arguments
myfun A string that is the name of the characteristic function defined by Users.
time.names Label of time.
var_char Argument of the characteristic function.
up Upper bound for the internal integration.
get.counting.data

<table>
<thead>
<tr>
<th>Low</th>
<th>Lower bound for the internal integration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>N_grid</td>
<td>Observation grid.</td>
</tr>
<tr>
<td>N_Fourier</td>
<td>Number of points for the Fourier Inversion.</td>
</tr>
</tbody>
</table>

**Details**

The density function is obtained by means of the Fourier Transform.

**Value**

An object of `yuima.law-class`.

**Author(s)**

The YUIMA Project Team

Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>

---

**get.counting.data**

*Extract arrival times from an object of class yuima.PPR*

**Description**

This function extracts arrival times from an object of class `yuima.PPR`.

**Usage**

`get.counting.data(yuimaPPR,type="zoo")`

**Arguments**

- `yuimaPPR` An object of class `yuima.PPR`.
- `type` By default `type="zoo"` the function returns an object of class `zoo`. Other values are `yuima.PPR` and `matrix`.

**Value**

By default the function returns an object of class `zoo`. The arrival times can be extracted by applying the method `index` to the output.
Examples

```r
## Not run:
##################
# Hawkes Process #
##################

# Values of parameters.
mu <- 2
alpha <- 4
beta <- 5

# Law definition
my.rHawkes <- function(n){
  res <- t(t(rep(1,n)))
  return(res)
}

Law.Hawkes <- setLaw(rng = my.rHawkes)

# Point Process Definition

gFun <- "mu"
Kernel <- "alpha*exp(-beta*(t-s))"

modHawkes <- setModel(drift = c("0"), diffusion = matrix("0",1,1),
  jump.coeff = matrix(c("1"),1,1), measure = list(df = Law.Hawkes),
  measure.type = "code", solve.variable = c("N"),
  xinit=c("0"))

prvHawkes <- setPPR(yuima = modHawkes, counting.var="N", gFun=gFun,
  Kernel = as.matrix(Kernel), lambda.var = "lambda",
  var.dx = "N", lower.var="0", upper.var = "t")

true.par <- list(mu=mu, alpha=alpha, beta=beta)

set.seed(1)
Term<-70
n<-7000

# Simulation trajectory

time.Hawkes <- system.time(
  simHawkes <- simulate(object = prvHawkes, true.parameter = true.par,
    sampling = setSampling(Terminal =Term, n=n))
)

# Arrival times of the Counting process.

DataHawkes <- get.counting.data(simHawkes)
TimeArr <- index(DataHawkes)
```
# Point Process Regression Model

# Values of parameters.
mu <- 2
alpha <- 4
beta <- 5

# Law definition
my.rKern <- function(n,t){
  res0 <- t(t(rgamma(n, 0.1*t)))
  res1 <- t(t(rep(1,n)))
  res <- cbind(res0,res1)
  return(res)
}

Law.PPRKern <- setLaw(rng = my.rKern)

# Point Process definition
modKern <- setModel(drift = c("0.4*(0.1-X)","0"),
  diffusion = c("0","0"),
  jump.coeff = matrix(c("1","0","0","1"),2,2),
  measure = list(df = Law.PPRKern),
  measure.type = c("code","code"),
  solve.variable = c("X","N"),
  xinit=c("0.25","0"))

gFun <- "exp(mu*log(1+X))"
#
Kernel <- "alpha*exp(-beta*(t-s))"

prvKern <- setPPR(yuima = modKern,
  counting.var="N", gFun=gFun,
  Kernel = as.matrix(Kernel),
  lambda.var = "lambda", var.dx = "N",
  lower.var="0", upper.var = "t")

# Simulation
Term<-100
seed<-1
n<-10000

true.parKern <- list(mu=mu, alpha=alpha, beta=beta)

set.seed(seed)
# set.seed(1)

time.simKern <- system.time(
  simprvKern <- simulate(object = prvKern, true.parameter = true.parKern,
```r
sampling = setSampling(Terminal = Term, n=n))

plot(simprvKern, main = "Counting Process with covariates", cex.main=0.9)

# Arrival Times
CountVar <- get.counting.data(simprvKern)
TimeArr <- index(CountVar)

## 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**

```r
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 `yuima.cogarch-class`.
- `data`: an object of class `yuima.data-class` contains the observations available at uniformly spaced time. If `data=NULL`, the default, the function uses the data in an object of `yuima-class`.
- `start`: a list containing the starting values for the optimization routine.
- `method`: a string indicating one of the methods available in `optim`.
- `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 observations.
- `equally.spaced`: Logical variable. If `equally.spaced = TRUE`, the function use the returns of COGARCH(P,Q) evaluated at unitary length for the computation of the empirical autocorrelations. If `equally.spaced = FALSE`, the increments are evaluated on the interval with frequency specified in an object of class `yuima.data-class` that contains the observed time series.
aggregation  If aggregation=TRUE, before the estimation of the levy parameters we aggregate the estimated increments

Est.Incr  a string variable, If Est.Incr = "NoIncr", default value, gmm returns an object of class cogarch.est-class that contains the COGARCH parameters. If Est.Incr = "Incr" or Est.Incr = "IncrPar" the output is an object of class cogarch.est.incr-class. 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. objFun = "L2", default value, the objective function is a quadratic form where the weighting Matrix is the identity one. objFun = "L2CUE" the weighting matrix is estimated using Continuously Updating GMM (L2CUE). objFun = "L1", 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

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
hyavar <- 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**

```r
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 `matrix(bw,d,d)`, 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 `cce`.

**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:

- `covmat` the estimated covariance matrix
- `cormat` the estimated correlation matrix
- `avar.cov` the estimated asymptotic variances for covariances
- `avar.cor` 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)**

Yuta Koike with YUIMA Project Team

**References**


**See Also**

`setData`, `cce`
Examples

## Not run:
## 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)
thehatat <- result$covmat[1,2] # estimate of the quadratic covariation
se <- sqrt(result$avar.cov[1,2]) # estimated standard error
c(lower = thehatat + qnorm(0.025) * se, upper = thehatat + 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)
IC

Information criteria for the stochastic differential equation

Description

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

IC(drif = NULL, diff = NULL, jump.coeff = NULL, data = NULL, Terminal = 1,
    add.settings = list(), start, lower, upper, ergodic = TRUE,
    stepwise = FALSE, weight = FALSE, rcpp = FALSE, ...)

Arguments

drif  a character vector that each element presents the candidate drift coefficient.
diff  a character vector that each element presents the candidate diffusion coefficient.
jump.coeff  a character vector that each element presents the candidate scale coefficient.
data  the data to be used.
Terminal  terminal time of the grid.
add.settings  details of model settings (see setModel).
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.
ergodic  whether the candidate models are ergodic SDEs or not (default ergodic = TRUE).
stepwise  specifies joint procedure or stepwise procedure (default stepwise = FALSE).
weight  calculate model weight? (default weight = FALSE)
rcpp  use C++ code? (default rcpp = FALSE)
...  passed to qmle

Details

Calculate the information criteria BIC, QBIC, and CIC for stochastic processes. The calculation and model selection are performed by joint procedure or stepwise procedure.

Value

BIC  values of BIC for all candidates.
QBIC  values of QBIC for all candidates.
AIC  values of AIC-type information criterion for all candidates.
model  information of all candidate models.
par  quasi-maximum likelihood estimator for each candidate.
weight  model weights for all candidates.
selected  selected model number and selected drift and diffusion coefficients

Note

The function IC uses the function qmle with method = "L-BFGS-B" internally.
References

### AIC, BIC


### BIC, Quasi-BIC


### CIC


### Model weight


Examples

### Not run:

#### Ex.1

```r
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])
}
```

#### Candidate coefficients

diffusion <- c("exp((theta11*cos(x)+theta12*sin(x)+theta13)/2)",
             "exp((theta11*cos(x)+theta12*sin(x))/2)",)
"exp((theta11*cos(x)+theta13)/2), "exp((theta12*sin(x)+theta13)/2)"

drift <- c("theta21*x + theta22", "theta21*x")

## Parameter settings
para.init <- list(theta11=runif(1,max=5,min=-5), theta12=runif(1,max=5,min=-5), theta13=runif(1,max=5,min=-5), theta21=runif(1,max=-0.5,min=-1.5), theta22=runif(1,max=-0.5,min=-1.5))
para.low <- list(theta11=-10, theta12=-10, theta13=-10, theta21=-5, theta22=-5)
para.upp <- list(theta11=10, theta12=10, theta13=10, theta21=-0.001, theta22=-0.001)

## Ex.1.1 Joint
ic1 <- IC(drif=drift, diff=diffusion, data=Xt, Terminal=Ter, start=para.init, lower=para.low,
upper=para.upp, stepwise = FALSE, weight = FALSE, rcpp = TRUE)

ic1

## Ex.1.2 Stepwise
ic2 <- IC(drif=drift, diff=diffusion, data=Xt, Terminal=Ter,
start=para.init, lower=para.low, upper=para.upp,
stepwise = TRUE, weight = FALSE, rcpp = TRUE)

ic2

### Ex.2 (multidimension 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*x1+beta2*x2+beta3", "1", "-1", "beta1*x1+beta2*x2+beta3"), 2, 2)

mod <- setModel(drift = c("alpha1*x1", "alpha2*x2"),
state.variable = c("x1", "x2"), solve.variable = c("x1", "x2"))

samp <- setSampling(Ter, n = N)

yuima <- setYuima(model = mod, sampling = samp)
simu.yuima <- simulate(yuima, xinit = c(1,1), true.parameter = list(alpha1=-2, alpha2=-1, beta1=-1, beta3=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,
}

## Candidate coefficients
diffusion <- list(matrix(c("beta1*x1+beta2*x2+beta3", "1", "-1", "beta1*x1+beta2*x2+beta3"), 2, 2),
matrix(c("beta1*x1+beta2*x2", "1", "-1", "beta1*x1+beta2*x2"), 2, 2),
matrix(c("beta1*x1+beta3", "1", "-1", "beta1*x1+beta3"), 2, 2),
matrix(c("beta2*x2+beta3", "1", "-1", "beta2*x2+beta3"), 2, 2),
matrix(c("beta1*x1", "1", "-1", "beta1*x1"), 2, 2),
matrix(c("beta2*x2", "1", "-1", "beta2*x2"), 2, 2),
matrix(c("beta3", "1", "-1", "beta3"), 2, 2))

modsettings <- list(state.variable = c("x1", "x2"), solve.variable = c("x1", "x2"))
## Parameter settings

```r
para.init <- list(alpha1 = runif(1, min=-3, max=-1), alpha2 = runif(1, min=-2, max=0),
                beta1 = runif(1, min=-2, max=0), beta2 = runif(1, min=0, max=2),
                beta3 = runif(1, min=1, max=3))
para.low <- list(alpha1 = -5, alpha2 = -5, beta1 = -5, beta2 = -5, beta3 = 1)
para.upp <- list(alpha1 = 0.01, alpha2 = -0.01, beta1 = 5, beta2 = 5, beta3 = 10)
```

## Ex.2.1 Joint

```r
ic3 <- IC(drif=drift, diff=diffusion, data=Xt, Terminal=Ter, add.settings=modsettings,
          start=para.init, lower=para.low, upper=para.upp,
          weight=FALSE, rcpp=FALSE)
ic3
```

## Ex.2.2 Stepwise

```r
ic4 <- IC(drif=drift, diff=diffusion, data=Xt, Terminal=Ter, add.settings=modsettings,
          start=para.init, lower=para.low, upper=para.upp,
          stepwise = TRUE, weight=FALSE, rcpp=FALSE)
ic4
```

## End(Not run)

---

### info.Map-class

*Class for information about Map/Operators*

**Description**

Auxiliar class for definition of an object of class `yuima.Map`. see the documentation of `yuima.Map` for more details.

---

### info.PPR

*Class for information about Point Process*

**Description**

Auxiliar class for definition of an object of class `yuima.PPR` and `yuima.Hawkes`. see the documentation for more details.

---

### Integral.sde

*Class for the mathematical description of integral of a stochastic process*

**Description**

Auxiliar class for definition of an object of class `yuima.Integral`. see the documentation of `yuima.Integral` for more details.
Integrand

Class for the mathematical description of integral of a stochastic process

Description

Auxiliary class for definition of an object of class \texttt{yuima.Integral}. See the documentation of \texttt{yuima[Integral]} for more details.

Intensity.PPR

\textit{Intensity Process for the Point Process Regression Model}

Description

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

Usage

\texttt{Intensity.PPR(yuimaPPR, param)}

Arguments

- \texttt{yuimaPPR} An object of class \texttt{yuima.PPR}
- \texttt{param} Model parameters

Value

- On object of class \texttt{yuima.data}

Author(s)

- YUIMA TEAM

Examples

- #INSERT HERE AN EXAMPLE
Remove jumps and calculate the Gaussian quasi-likelihood estimator based on the Jarque-Bera normality test

Usage

```r
JBtest(yuima,start,lower,upper,alpha,skewness=TRUE,kurtosis=TRUE,withdrift=FALSE)
```

Arguments

- `yuima`: a yuima object (diffusion with compound Poisson jumps).
- `lower`: a named list for specifying lower bounds of parameters.
- `upper`: a named list for specifying upper bounds of parameters.
- `alpha`: Insert Description Here.
- `start`: initial values to be passed to the optimizer.
- `skewness`: use third moment information? by default, skewness=TRUE
- `kurtosis`: use fourth moment information? by default, kurtosis=TRUE
- `withdrift`: use drift information for constructing self-normalized residuals or not? by default, withdrift = FALSE

Details

This function removes large increments which are regarded as jumps based on the iterative Jarque-Bera normality test, and after that, calculates the Gaussian quasi maximum likelihood estimator.

Value

- `Removed`: Removed jumps and jump times
- `OGQMLE`: Gaussian quasi maximum likelihood estimator before jump removal
- `JRGQMLE`: Gaussian quasi maximum likelihood estimator after jump removal
- `Figures`: For visualization, the jump points are presented. In addition, the histogram of the jump removed self-normalized residuals, transition of the estimators and the logarithm of Jarque-Bera statistics are given as figures

Author(s)

The YUIMA Project Team

Contacts: Yuma Uehara <y-uehara@ism.ac.jp>
References


Examples

```r
## Not run:
set.seed(123)
mod <- setModel(drift="10-3*x",
  diffusion="theta*(2+x^2)/(1+x^2)",
  jump.coeff="1",
  measure=list(intensity="1",df=list("dunif(z, 3, 5)")),
  measure.type="CP")

T <- 10 ## Terminal
n <- 5000 ## generation size
samp <- setSampling(Terminal=T, n=n) ## define sampling scheme
yuima <- setYuima(model = mod, sampling = samp)

yuima <- simulate(yuima, xinit=1,true.parameter=list(theta=sqrt(2)), sampling = samp)

JBtest(yuima,start=list(theta=0.5),upper=c(theta=100)
  ,lower=c(theta=0),alpha=0.01)

## End(Not run)
```

---

### 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**

```r
lambdaFromData(yuimaPPR, PPRData = NULL, parLambda = list())
```

**Arguments**

- `yuimaPPR`: Mathematical Description of PPR model
- `PPRData`: Observed data
- `parLambda`: Values of intensity parameters
lasso

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 optim method. See Examples.

Details
lasso behaves more likely the standard qml 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

## Not run:
## 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)

## End(Not run)
**Usage**

```r
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 assymptotic normality of theta estimators.

**Usage**

```r
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 assymptotic 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="(-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**

```r
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, tol = 1e-6)
```

**Arguments**

- `x`: an object of `yuima-class` or `yuima.data-class`.
- `verbose`: logical. If FALSE, llag returns lead-lag time estimates only. 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 `hyavar` for details.
tol 

tolerance parameter to avoid numerical errors in comparison of time stamps. All time stamps are divided by tol and rounded to integers. Note that the values of grid are also divided by tol and rounded to integers. A reasonable choice of tol is the minimum unit of time stamps. The default value 1e-6 supposes that the minimum unit of time stamps is greater or equal to 1 micro-second.

Details

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

Let $X^i_{t^i_0}, X^i_{t^i_1}, \ldots, X^i_{t^i_{n(i)}}$ 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 $C := (U_{kk}(\theta))_{k,l\in I^{i,j}}$ 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

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

where $a$, $b$ and $N$ are the $(d(i-1) - (i-1)i/2 + (j-i))$-th components of from to and division respectively. If the corresponding component of from (resp. to) is $-\text{Inf}$ (resp. $\text{Inf}$), $a = -(t^i_{n(i)} - t^i_0)$ (resp. $b = t^i_{n(i)} - t^i_0$) is used, while if the corresponding component of division is FALSE, $N = \text{round}(2\text{max}(n(i), n(j))) + 1$ is used. Missing components are filled with $-\text{Inf}$ (resp. $\text{Inf}$, FALSE). The default value $-\text{Inf}$ (resp. $\text{Inf}$, FALSE) means that all components are $-\text{Inf}$ (resp. $\text{Inf}$, 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 $(\theta_{ij})_{i,j=1,\ldots,d}$. If verbose is TRUE, the covariance matrix $(U_{ij}(\theta_{ij}))_{i,j=1,\ldots,d}$ corresponding to the estimated lead-lag parameters, the corresponding correlation matrix and the computed contrast functions are also returned. If further ccor is TRUE the computed cross-correlation functions are returned as a list with the length $d(d-1)/2$. For $i < j$, the $(d(i-1) - (i-1)i/2 + (j-i))$-th component of the list consists of an object $U_{ij}(\theta)/\sqrt{U_{ii}(\theta)U_{jj}(\theta)}$ 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.1lag", which is a list with the following components, is returned:

Details

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

Let $X^i_{t^i_0}, X^i_{t^i_1}, \ldots, X^i_{t^i_{n(i)}}$ 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 $C := (U_{kk}(\theta))_{k,l\in I^{i,j}}$ 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

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

where $a$, $b$ and $N$ are the $(d(i-1) - (i-1)i/2 + (j-i))$-th components of from to and division respectively. If the corresponding component of from (resp. to) is $-\text{Inf}$ (resp. $\text{Inf}$), $a = -(t^i_{n(i)} - t^i_0)$ (resp. $b = t^i_{n(i)} - t^i_0$) is used, while if the corresponding component of division is FALSE, $N = \text{round}(2\text{max}(n(i), n(j))) + 1$ is used. Missing components are filled with $-\text{Inf}$ (resp. $\text{Inf}$, FALSE). The default value $-\text{Inf}$ (resp. $\text{Inf}$, FALSE) means that all components are $-\text{Inf}$ (resp. $\text{Inf}$, 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 $(\theta_{ij})_{i,j=1,\ldots,d}$. If verbose is TRUE, the covariance matrix $(U_{ij}(\theta_{ij}))_{i,j=1,\ldots,d}$ corresponding to the estimated lead-lag parameters, the corresponding correlation matrix and the computed contrast functions are also returned. If further ccor is TRUE the computed cross-correlation functions are returned as a list with the length $d(d-1)/2$. For $i < j$, the $(d(i-1) - (i-1)i/2 + (j-i))$-th component of the list consists of an object $U_{ij}(\theta)/\sqrt{U_{ii}(\theta)U_{jj}(\theta)}$ 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.1lag", which is a list with the following components, is returned:
lagcce  a skew-symmetric matrix corresponding to the estimated lead-lag parameters.
covmat  a covariance matrix corresponding to the estimated lead-lag parameters.
cormat  a correlation matrix corresponding to the estimated lead-lag parameters.
LLR     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 multi-
ple 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 con-
jected 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)

Yuta Koike with YUIMA Project Team

References

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

non-synchronous data, Bernoulli, 19, no. 2, 426–461.

of Empirical Finance, 26, 41–58.

See Also

cce, hyavar, mllag, llag.test
## 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
cc(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))

---

### llag.test

#### Wild Bootstrap Test for the Absence of Lead-Lag Effects

**Description**

Tests the absence of lead-lag effects (time-lagged correlations) by the wild bootstrap procedure proposed in Koike (2017) for each pair of components.

**Usage**

```r
llag.test(x, from = -Inf, to = Inf, division = FALSE, grid, R = 999,
parallel = "no", ncpus = getOption("boot.ncpus", 1L),
c1 = NULL, tol = 1e-06)
```

**Arguments**

- `x`: an object of `yuima-class` or `yuima.data-class`. 
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' of llag.
R a single positive integer indicating the number of bootstrap replicates.
parallel passed to boot.
cmpus passed to boot.
cl passed to boot.
tol tolelance parameter to avoid numerical errors in comparison of time stamps. All time stamps are divided by tol and rounded to integers. Note that the values of grid are also divided by tol and rounded to integers. A reasonable choice of tol is the minimum unit of time stamps. The default value 1e-6 supposes that the minimum unit of time stamps is greater or equal to 1 micro-second.

Details

For each pair of components, this function performs the wild bootstrap procedure proposed in Koike (2017) to test whether there is a (possibly) time-lagged correlation. The null hypothesis of the test is that there is no time-lagged correlation and the alternative is its negative. The test rejects the null hypothesis if the maximum of the absolute values of cross-covariances is too large. The critical region is constructed by a wild bootstrap procedure with Rademacher variables as the multiplier variables.

Value

p.values a matrix whose components indicate the bootstrap p-values for the corresponding pair of components.
max.cov a matrix whose components indicate the maxima of the absolute values of cross-covariances for the corresponding pair of components.
max.corr a matrix whose components indicate the maxima of the absolute values of cross-correlations for the corresponding pair of components.

Author(s)

Yuta Koike with YUIMA Project Team

References


See Also

cce, hyavar, mllag, 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"))
```

## 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)
```

## perform lead-lag test
```
llag.test(psample, grid = G, R = 999)
```

## Since the lead-lag parameter for the pair(x1, x3) is not contained in G, 
## the null hypothesis is not rejected for this pair

## End(Not run)
lm.jumptest

Lee and Mykland's Test for the Presence of Jumps Using Normalized Returns

Description

Performs a test for the null hypothesis that the realized path has no jump following Lee and Mykland (2008).

Usage

lm.jumptest(yuima, K)

Arguments

- **yuima**: an object of `yuima-class` or `yuima.data-class`.
- **K**: a positive integer indicating the window size to compute local variance estimates. It can be specified as a vector to use different window sizes for different components. The default value is $K = \text{pmin}(\text{floor}(\sqrt{252*n}), n)$ with $n = \text{length}(\text{yuima}) - 1$, following Lee and Mykland (2008) as well as Dumitru and Urga (2012).

Value

A list with the same length as `dim(yuima)`. Each component of the list has class "htest" and contains the following components:

- **statistic**: the value of the test statistic of the corresponding component of `yuima`.
- **p.value**: an approximate p-value for the test of the corresponding component.
- **method**: the character string "Lee and Mykland jump test".
- **data.name**: the character string "xi", where $i$ is the number of the component.

Author(s)

Yuta Koike with YUIMA Project Team

References


See Also

bns.test, minrv.test, medrv.test, pz.test

Examples

set.seed(123)

# One-dimensional case
## Model: dX_t = t dW_t + t dz_t,
## where z_t is a compound Poisson process with intensity 5 and jump sizes distribution N(0,1).

model <- setModel(drift=0, diffusion="t", jump.coeff="t", measure.type="CP",
                   measure=list(intensity=5, df="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

lm.jumptest(yuima) # p-value is very small, so the path would have a jump
lm.jumptest(yuima, K = floor(sqrt(390))) # different value of K

# Multi-dimensional case
## Model: Bivariate standard BM + CP
## Only the first component has jumps

mod <- setModel(drift = c(0, 0), diffusion = diag(2),
                jump.coeff = diag(c(1, 0)),
                measure = list(intensity = 5,
                                df = "dmvnorm(z,c(0,0),diag(2))"),
                jump.variable = c("z"), measure.type=c("CP"),
                solve.variable=c("x1","x2"))

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

lm.jumptest(yuima) # test is performed component-wise

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$allObs` contains the intraday five minutes Standard and Poor cumulative Log-return data computed as \( \log(P_t) - \log(P_0) \) 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`.

Examples

data(LogSPX)

---

lseBayes

Adaptive Bayes estimator for the parameters in sde model by using LSE functions

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


Examples

```r
## 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))
```
mllag <- setModel(drift=b,diffusion=a,state.variable=c("x1", "x2"),solve.variable=c("x1","x2"))

yuimamasamp <- setSampling(Terminal=50,n=50*100)
yuima <- setYuima(model = yuimamodel, sampling = yuimamasamp)
yuima <- simulate(yuima, xinit = c(100,80),
true.parameter = true,sampling = yuimamasamp)

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 `yuima-class` or `yuima.data-class` or `yuima.llag-class` (output of `llag`) or `yuima.mllag-class` (output of this function).
- **from**: passed to `llag`.
- **to**: passed to `llag`.
- **division**: passed to `llag`.
- **grid**: passed to `llag`.
- **psd**: passed to `llag`.
- **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 `llag`.

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:

- **mlagcce**: a list of data.frame-class objects consisting of lagcce (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)

Yuta Koike with YUIMA Project Team
References


See Also

llag, hyavar, llag.test

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

\[
G \leftarrow \text{seq}(-0.1, 0.1, \text{by}=0.01)
\]

lead-lag estimation by mllag

\[
\text{par(mfcol=c(3,1))}
\]

\[
\text{result} \leftarrow \text{mllag(psample, grid=G)}
\]

Since the lead-lag parameter for the pair\((x_1, x_3)\) is not contained in \(G\),
no lead-lag parameter is detected for this pair

\[
\text{par(mfcol=c(1,1))}
\]

The second example is a situation where multiple lead-lag effects exist

\[
n \leftarrow 3600
\]

\[
\text{Times} \leftarrow \text{seq}(0, 1, \text{by}=1/n)
\]

\[
R1 \leftarrow 0.6
\]

\[
R2 \leftarrow -0.3
\]

\[
dW1 \leftarrow \text{rnorm}(n+10)/\sqrt{n}
\]

\[
dW2 \leftarrow \text{rnorm}(n+5)/\sqrt{n}
\]

\[
dW3 \leftarrow \text{rnorm}(n)/\sqrt{n}
\]

\[
x \leftarrow \text{zoo}(\text{diffinv}(dW1[-(1:10)] + dW2[1:n]), \text{Times})
\]

\[
y \leftarrow \text{zoo}(\text{diffinv}(R1 \times dW1[1:n] + R2 \times dW2[-(1:5)] +
\sqrt{1- R1^2 - R2^2} \times dW3), \text{Times})
\]

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

\[
yuima \leftarrow \text{setData(list}(x, y))
\]

Lead-lag estimation by llag

\[
G \leftarrow \text{seq}(-30/n, 30/n, \text{by}=1/n)
\]

\[
est \leftarrow \text{llag}(yuima, \text{grid=G, ci=TRUE})
\]

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

\[
\text{psample} \leftarrow \text{poisson.random.sampling}(yuima, n=n, \text{rate}=c(0.8, 0.7))
\]

Lead-lag estimation by mllag

\[
est \leftarrow mllag(psample, \text{grid=G})
\]

\[
est \# detects too many lead-lag parameters
\]

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

#### Description

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

#### Usage

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

#### Arguments

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

#### Details

Estimates the drift of a 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


#### See Also

See also `qgv`. 
model.parameter-class

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 component.

Usage

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

Arguments

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

Details

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

Let $X_{i_0}^i, X_{i_1}^i, \ldots, X_{i_{n_i}}^i$ 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, $\text{mpv}(yuima, r, \text{normalize}=\text{TRUE})$ is defined as the $d$-dimensional vector with $i$-th element equal to

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

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 $\mu_{r[1]}^{-1} \cdot \mu_{r[k]}^{-1}$.

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

$$
\mu_{r_{i_1}}^{-1} \cdots \mu_{r_{k_i}}^{-1} n^{\sum_{j=1}^{n-k_i+1} |\Delta X_{t_j}^i|^{r_{i_1,j}} |\Delta X_{t_{j+1}}^i|^{r_{i_2,j}} \cdots |\Delta X_{t_{j+k_i-1}}^i|^{r_{i_{k_i},j}}},
$$

where $r_{i_1,j}, \ldots, r_{i_{k_i},j}$ is the $i$-th component of $r$. If normalize is FALSE the result is not multiplied by $\mu_{r_{i_1}}^{-1} \cdots \mu_{r_{k_i}}^{-1}$.

Value

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

Author(s)
Yuta Koike with YUIMA Project Team

References

See Also
setData, cce, minrv, medrv

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

# One-dimensional case
## Model: dX_t = t dW_t + t 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)
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: dX_{kt} = t dW_{kt} (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)

## End(Not run)

<table>
<thead>
<tr>
<th>MWK151</th>
<th>Graybill - Methuselah Walk - PILO - ITRDB CA535</th>
</tr>
</thead>
</table>

**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 [https://www.ncei.noaa.gov/products/paleoclimatology](https://www.ncei.noaa.gov/products/paleoclimatology).

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**

doi:10.25921/ppqjxy48

**References**


**Examples**

data(MWK151)
noisy.sampling

**Noisy Observation Generator**

**Description**

Generates a new observation data contaminated by noise.

**Usage**

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

**Arguments**

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

**Details**

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**

## Examples

### Set a model (a two-dimensional normal model sampled by a Poisson random sampling)

```r
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

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

### Plot the path without noise

```r
plot(psample)
```

### Set a matrix as the variance of noise

```r
Omega <- 0.01 * diffusion %*% t(diffusion)
```

### Contaminate the observation data by centered normal distributed noise

```r
noisy.psample1 <- noisy.sampling(psample, var.adj = Omega)
plot(noisy.psample1)
```

### Contaminate the observation data by centered uniformly distributed noise

```r
noisy.psample2 <- noisy.sampling(psample, var.adj = Omega, rng = "runif", min = -sqrt(3), max = sqrt(3))
plot(noisy.psample2)
```

### Contaminate the observation data by centered exponentially distributed noise

```r
noisy.psample3 <- noisy.sampling(psample, var.adj = Omega, rng = "rexp", rate = 1, mean.adj = 1)
plot(noisy.psample3)
```

### Contaminate the observation data by its return series

```r
noisy.psample4 <- noisy.sampling(psample, end.coef = -0.1, n = 2340 * c(1/3, 1/6))
plot(noisy.psample4)
```
## An application:
## Adding a compound Poisson jumps to the observation data

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

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

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

---

### Description

`minrv` and `medrv` respectively compute the MinRV and MedRV estimators introduced in Andersen, Dobrev and Schaumburg (2012).

`minrv.test` and `medrv.test` respectively perform Haussman type tests for the null hypothesis that the realized path has no jump using the MinRV and MedRV estimators. See Section 4.4 in Andersen, Dobrev and Schaumburg (2014) for a concise discussion.

### Usage

```r
minrv(yuima)
medrv(yuima)
minrv.test(yuima, type = "ratio", adj = TRUE)
medrv.test(yuima, type = "ratio", adj = TRUE)
```

### Arguments

- `yuima`: an object of `yuima-class` or `yuima.data-class`.
- `type`: type of the test statistic to use. `ratio` 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". See also Section 2.5 in Dumitru and Urga (2012).
Value

\texttt{minrv} and \texttt{medrv} return a numeric vector with the same length as \texttt{dim(yuima)}. Each component of the vector is a volatility estimate for the corresponding component of \texttt{yuima}.

\texttt{minrv.test} and \texttt{medrv.test} return a list with the same length as \texttt{dim(yuima)}. Each component of the list has class \"ltest\" and contains the following components:

- \texttt{statistic} the value of the test statistic of the corresponding component of \texttt{yuima}.
- \texttt{p.value} an approximate p-value for the test of the corresponding component.
- \texttt{method} the character string “Andersen-Dobrev-Schaumburg jump test based on xxx”, where xxx is either MinRV or MedRV.
- \texttt{data.name} the character string “xi”, where i is the number of the component.

Author(s)

Yuta Koike with YUIMA Project Team

References


See Also

\texttt{mpv}, \texttt{cce}, \texttt{bns.test}, \texttt{lm.jumptest}, \texttt{pz.test}

Examples

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

# One-dimensional case
## Model: dX_t = t dW_t + t dZ_t, where Z_t is a compound Poisson process with intensity 5
## and jump sizes distribution N(0,1).

model <- setModel(drift=0, diffusion="t", jump.coef="t", measure.type="CP",
                   measure=list(intensity=5, df=list("dnorm(z,0,1)")),
                   time.variable="t")
yuima.samp <- setSampling(Terminal = 1, n = 390)
```
### param.Integral

**Class for the mathematical description of integral of a stochastic process**

**Description**

Auxilar class for definition of an object of class `yuima.Integral`. See the documentation of `yuima.Integral` for more details.
param.Map-class

Class for information about Map/Operators

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 qmle function.

Details

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

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

Value

ans an object of class phitest.

Author(s)

The YUIMA Project Team
### Examples

```r
## Not run:
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, phi=phi1, method="L-BFGS-B")
phi.test(X, H0=h1, phi=phi1, method="L-BFGS-B")
```

## End(Not run)

---

### Poisson.random.sampling

**Poisson random sampling method**

#### Description

Poisson random sampling method.

#### Usage

```r
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`.
## Set a model

```r
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)"),2,2)
cor.mod <- setModel(drift=c("",""), diffusion=diff.coef.matrix, solve.variable=c("x1", "x2"), xinit=c(3,2))
set.seed(111)
```

```r
## 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)
```

```r
## 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)
```

---

### pz.test

**Podolskij and Ziggel’s Test for the Presence of Jumps Using Power Variation with Perturbed Truncation**

**Description**

Performs a test for the null hypothesis that the realized path has no jump following Podolskij and Ziggel (2010).

**Usage**

```r
pz.test(yuima, p = 4, threshold = "local", tau = 0.05)
```

**Arguments**

- `yuima` : an object of `yuima-class` or `yuima.data-class`.
- `p` : a positive number indicating the exponent of the (truncated) power variation to compute test statistic(s). Theoretically, it must be greater than or equal to 2.
threshold a numeric vector or list indicating the threshold parameter(s). Each of its components indicates the threshold parameter or process to be used for estimating the corresponding component. If it is a numeric vector, the elements in `threshold` are recycled if there are two few elements in `threshold`. Alternatively, you can specify either "PZ" or "local" to automatically select a (hopefully) appropriate threshold. When `threshold="PZ"`, selection is performed following Section 5.1 in Podolskij and Ziggel (2010). When `threshold="local"`, selection is performed following Section 5.1 in Koike (2014). The default is `threshold="local"`.

tau a probability controlling the strength of perturbation. See Section 2.3 in Podolskij and Ziggel (2010) for details. Podolskij and Ziggel (2010) suggests using a relatively small value for `tau`, e.g. `tau=0.1` or `tau=0.05`.

Value
A list with the same length as `dim(yuima)`. Each component of the list has class "htest" and contains the following components:

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

Note
Podolskij and Ziggel (2010) also introduce a pre-averaged version of the test to deal with noisy observations. Such a test will be implemented in the future version of the package.

Author(s)
Yuta Koike with YUIMA Project Team

References


See Also
`bns.test`, `lm.jumptest`, `minrv.test`, `medrv.test`
Examples

```r
## Not run:
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(0,1).

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

#lm.jumptest(yuima) # p-value is very small, so the path would have a jump
#lm.jumptest(yuima, K = floor(sqrt(390))) # different value of K
pz.test(yuima) # p-value is very small, so the path would have a jump
pz.test(yuima, p = 2) # different value of p
pz.test(yuima, tau = 0.1) # different value of tau

# Multi-dimensional case
## Model: Bivariate standard BM + CP
## Only the first component has jumps

mod <- setModel(drift = c(0, 0), diffusion = diag(2),
                 jump.coeff = diag(c(1, 0)),
                 measure = list(intensity = 5,
                               df = "dmvnorm(z,c(0,0),diag(2))"),
                 jump.variable = c("z"), measure.type=c("CP"),
                 solve.variable=c("x1","x2"))

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

pz.test(yuima) # test is performed component-wise

## End(Not run)
```

Description

Estimate the local Holder exponent with quadratic generalized variations method
Usage

\texttt{qgv(yuima, filter.type = "Daubechies", order = 2, a = NULL)}

Arguments

\begin{itemize}
\item \texttt{yuima} A \texttt{yuima} object.
\item \texttt{filter.type} The \texttt{filter.type} can be set to "Daubechies" or "Classical".
\item \texttt{order} The order of the filter \texttt{a} to be chosen
\item \texttt{a} Any other filter
\end{itemize}

Details

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

Value

an object of class \texttt{qgv}

Author(s)

The YUIMA Project Team

References


See Also

See also \texttt{mmfrac}.

Examples

\begin{verbatim}
# 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)
\end{verbatim}
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

```r
qmle(yuima, start, method = "L-BFGS-B", fixed = list(),
print = FALSE, envir = globalenv(), 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.
- `envir`: an environment where the model coefficients are evaluated.
- `method`: see Details.
- `param`: list of parameters for the quasi loglikelihood.
- `lower`: a named list for specifying lower bounds of parameters
- `upper`: a named list for specifying upper bounds of parameters
- `start`: initial values to be passed to the optimizer.
- `fixed`: for conditional (quasi)maximum likelihood estimation.
- `joint`: perform joint estimation or two stage estimation? by default joint=FALSE.
- `Est.Incr`: If the yuima model is an object of `yuima.carma-class` or `yuima.cogarch-class` the qmle returns an object of `yuima.carma.qmle-class`, `cogarch.est.incr-class`, `cogarch.est-class` or object of class `mle-class`. By default Est.Incr="NoIncr", alternative values are IncrPar and Incr.
- `aggregation`: If aggregation=TRUE, before the estimation of the levy parameters we aggregate the increments.
- `threshold`: If the model has Compound Poisson type jumps, the threshold is used to perform thresholding of the increments.
- `...`: passed to optim method. See Examples.
- `rcpp`: use C++ code?
qmle

Details

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

1se 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 yuima.carma-class, qmle returns an object yuima.carma.qmle-class
cogarchopt if the model is an object of yuima.cogarch-class, qmle returns an object of class cogarch.est-class. 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 diffucion


## Ergodic diffusion


## Jump diffusion


## COGARCH

Examples

```r
# dX^e_t = -theta2 * X^e_t * dt + theta1 * dW_t
diff.matrix <- matrix(c("theta1"), 1, 1)
ymodel <- setModel(drift=c("(-1)*theta2*x"), diffusion=diff.matrix,
                   time.variable="t", state.variable="x", solve.variable="x")
n <- 100
ysamp <- setSampling(Terminal=(n)^(1/3), n=n)
yuima <- setYuima(model=ymodel, sampling=ysamp)
set.seed(123)
yuima <- simulate(yuima, xinit=1, true.parameter=list(theta1=0.3,
                 theta2=0.1))
QL <- quasilogl(yuima, param=list(theta2=0.8, theta1=0.7))
## QL <- ql(yuima, 0.8, 0.7, h=1/((n)^(2/3)))
QL

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

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

system.time(
  opt2 <- qmle(yuima, start=list(theta1=0.8, theta2=0.7), lower=list(theta1=0, theta2=0),
                upper=list(theta1=1, theta2=1), method="L-BFGS-B")
)
cat(sprintf("\nTrue param. theta1 = .3, theta2 = .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. theta1 = .3, theta2 = .1\n"))
print(coef(opt3))
```
## perform joint estimation? Non-optimal, just for didactic purposes

```r
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. theta1 = .3, theta2 = .1\n"))
print(coef(opt4))
```

## fix theta1 to the true value

```r
system.time(
  opt5 <- qmle(yuima, start=list(theta2=0.7), lower=list(theta2=0),
                upper=list(theta2=1), fixed=list(theta1=0.3), method="L-BFGS-B")
)
cat(sprintf("\nTrue param. theta1 = .3, theta2 = .1\n"))
print(coef(opt5))
```

## old code

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

## Not run:

### multidimension case

```r
drift.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)
```

```r
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)
```

```r
#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))
```

```r
## 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
```

```r
## 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,
             alpha1=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,
                grid eq = 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)

---

qmleLevy

Gaussian quasi-likelihood estimation for Levy driven SDE

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, Est.Incr = "NoIncr",
          aggregation = TRUE)

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.
**qmleLevy**

**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.

**Est.Incr**

the qmleLevy returns an object of mle-clas, by default Est.Incr="NoIncr", other options as "Inc" or "IncrPar".

**aggregation**

If aggregation=TRUE, the function returns the unit-time Levy increments. If Est.Incr="IncrPar", the function estimates Levy parameters using the unit-time Levy increments.

**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, birateral 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=ZGVmYXVsdGRvbWFpbmx5dWlhWVoYXJhMTkyOHxneDo3ZTdlMTA1OTMyZTBkYjQ2

or

https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbmx5dWlhWVoYXJhMTkyOHxneDo3ZTdlMTA1OTMyZTBkYjQ2

They also contain more precise explanation of this function.

**Author(s)**

The YUIMA Project Team

Contacts: Yuma Uehara <y-uehara@ism.ac.jp>

**References**


Examples

## Not run:
## 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<"(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.coef=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, lambda=lambda, alpha=alpha,
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)
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)
rnts(x, alpha, a, b, beta, mu, Lambda)

Arguments

x Number of R.Ns to be geneated.
a parameter
b parameter
delta parameter written as $\delta$ below
gamma parameter written as $\gamma$ below
mu parameter written as $\mu$ below
Lambda parameter written as $\Lambda$ below
alpha parameter written as $\alpha$ below
lambda parameter written as $\lambda$ below
sigma parameter written as $\sigma$ below
beta parameter written as $\beta$ below
delta.plus parameter written as $\delta_+$ below
gamma.plus parameter written as $\gamma_+$ below
delta.minus parameter written as $\delta_-$ below
gamma.minus parameter written as $\gamma_-$ below
Details

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

\[ f(x) = \frac{1}{2} \cdot \frac{1}{\Gamma(\lambda) \Gamma(\delta)} \lambda^{\lambda} \delta^\delta \exp(-\lambda x + \delta x^2) \]

where \( \Gamma(\lambda) \) and \( \Gamma(\delta) \) are the Gamma functions, \( \lambda \) and \( \delta \) are positive and \( \lambda > \delta \). The parameters \( \lambda, \delta, \alpha, \beta, \mu \) 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:

\[ \delta > 0, \alpha > 0, \beta > 0, \mu > 0 \]


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:

\[ \delta > 0, \alpha > 0, \beta > 0, \mu > 0 \]


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 \( \alpha, \beta, \delta, \mu \) express 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 \( \alpha^2 - \beta^2 \Lambda \beta > 0 \).

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 \( \det(\Lambda) = 1 \); \( \delta > 0 \); \( \alpha > 0 \) with \( \alpha^2 - \beta^2 \Lambda \beta > 0 \). 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 \( \Gamma(\delta, \gamma_+ \text{and} \Gamma(\delta, \gamma_-) \). Its Levy density \( f(z) \) is given by:

\[ f(z) = \delta_+/z \cdot \exp(-\gamma_+ \cdot z) \cdot \text{ind}(z > 0) + \delta_-/|z| \cdot \exp(-\gamma_- \cdot |z|) \cdot \text{ind}(z < 0) \]

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: \( 0 < \alpha <= 2; -1 <= \beta <= 1; \sigma > 0; \gamma \) 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 Lévy density $f(z)$ is given by:

$$f(z) = 2a/(2\pi)^{1/2} \cdot \exp(\beta \cdot z) \cdot (z^2/(2b+\beta^2))^{1/2} - \alpha/2 - 1/4 \times bK(\alpha+1/2)(z^2(2b+\beta^2)^{1/2})$$

### Value

- `rXXX`: Collection of random numbers or vectors
- `dXXX`: Density function

### 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 $\exp(a \cdot \Gamma(-\alpha) \cdot b/\alpha)$

### Author(s)

The YUIMA Project Team

Contacts: Hiroki Masuda <hmasuda@ms.u-tokyo.ac.jp> and Yuma Uehara <y-uehara@kansai-u.ac.jp>

### References


# rbgamma, dbgamma


# rstable


# rpts


# rnts


Examples

# Not run:
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(0.5, 0.5)
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. (Generarized 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)

## End(Not run)
setCarma

Description

'setCarma' describes the following model:
\[ V_t = c_0 + \sigma (b_0 X_t(0) + \ldots + b(q) X_t(q)) \]
\[ dX_t(0) = X_t(1) \, dt \]
\[ \ldots \]
\[ dX_t(p-2) = X_t(p-1) \, dt \]
\[ dX_t(p-1) = (-a(p) X_t(0) - \ldots - a(1) X_t(p-1))dt + (\gamma(0) + \gamma(1) X_t(0) + \ldots + \gamma(p) X_t(p-1))dz_t \]

The continuous ARMA process using the state-space representation as in Brockwell (2000) is obtained by choosing:
\[ \gamma(0) = 1, \gamma(1) = \gamma(2) = \ldots = \gamma(p) = 0. \]

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 \( c_0 = 0 \).
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 yuima.model-class measure Levy measure of jump variables.
measure.type type specification for Levy measure.
xinit a vector of expressions identifying 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

Amsterdam.

Examples

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

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

# Look at the model structure by
str(mod2)

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

# Look at the model structure by
str(mod3)

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

# Look at the model structure by
str(mod4)
Description

'setCarmaHawkes' describes a self-exciting Hawkes process where the intensity is a CARMA(p,q) process. The model admits the Hawkes process with exponential kernel as a special case but it is able to reproduce a more complex time-dependence structure.

Usage

```
setCarmaHawkes(p, q, law = NULL, base.Int = "mu0", 
ar.par = "a", ma.par = "b", Counting.Process = "N", 
Intensity.var = "lambda", Latent.var = "x", time.var = "t", 
Type.Jump = FALSE, XinExpr = FALSE)
```

Arguments

- `p`: a non-negative integer that indicates the number of the autoregressive coefficients.
- `q`: a non-negative integer that indicates the number of the moving average coefficients.
- `law`: An object of `yuima.law-class` that describes the jump size. If it is NULL, the jump size is unitary.
- `base.Int`: a character-string that is the label of the baseline Intensity parameter. Defaults to "mu0".
- `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".
- `Counting.Process`: a character-string that is the label of the Counting process. Defaults to "N".
- `Intensity.var`: a character-string that is the label of the Intensity process. Defaults to "lambda".
- `Latent.var`: a character-string that is the label of the unobserved process. Defaults to "x".
- `time.var`: the name of the time variable.
- `Type.Jump`: a logical value. If it is TRUE, the jump size is deterministic.
- `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.

Value

Model an object of `yuima.carmaHawkes-class`.

Author(s)

The YUIMA Project Team

Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>
setCharacteristic

References

Examples
```r
## Not run:
# Definition of an Hawkes with exponential Kernel
mod1 <- setCarmaHawkes(p = 1, q = 0)

# Definition of an Hawkes with a CARMA(2,1) Intensity process
mod2 <- setCarmaHawkes(p = 2, q = 1)

## End(Not run)
```

**setCharacteristic**  
*Set characteristic information and create a ‘characteristic’ object.*

**Description**

setCharacteristic is a constructor for characteristic class.

**Usage**

```r
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

Continuous-time GARCH (p,q) process

Description

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

\[
\text{d}G_t = \sqrt{V_t}\text{d}Z_t \\
V_t = a_0 + (a_1 Y_t(1) + \ldots + a(p) Y_t(p)) \\
\text{d}Y_t(1) = Y_t(2) \text{dt} \\
\ldots \\
\text{d}Y_t(q-1) = Y_t(q) \text{dt} \\
\text{d}Y_t(q) = (-b(q) Y_t(1) - \ldots - b(1) Y_t(q)) \text{dt} + (a_0 + (a_1 Y_t(1) + \ldots + a(p) Y_t(p)) \text{d}[Z_tZ_t]^{|q|}
\]

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 yuima.model-class
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 \( \beta = a0 b1, \eta = b1 \) and \( \phi = a1 \), we obtain the Cogarch(1,1) model proposed in Kluppelberg et al. (2004) defined as the solution of the SDEs:

\[
\begin{align*}
dG_t &= \sqrt{V_t}dZ_t \\
dV_t &= (\beta - \eta V_t) \, dt + \phi V_t \, d[Z_tZ_t]^{q}
\end{align*}
\]

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


Examples

```r
# 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

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

Description

setData constructs an object of \texttt{yuima.data-class}.
get.zoo.data returns the content of the \texttt{zoo.data} slot of a \texttt{yuima.data-class} object. (Note: value is a list of \texttt{zoo} objects).
plot plot method for object of \texttt{yuima.data-class} or \texttt{yuima-class}.
dim returns the \texttt{dim} of the \texttt{zoo.data} slot of a \texttt{yuima.data-class} object.
length returns the \texttt{length} of the time series in \texttt{zoo.data} slot of a \texttt{yuima.data-class} object.
cbind.yuima bind \texttt{yuima} data object.

Usage

\begin{verbatim}
setData(original.data, delta=NULL, t0=0)
get.zoo.data(x)
\end{verbatim}

Arguments

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

Details

Objects in the \texttt{yuima.data-class} contain two slots:

\begin{itemize}
\item \texttt{original.data}: The slot \texttt{original.data} contains, as the name suggests, a copy of the original data passed to the function \texttt{setData}. It is intended for backup purposes.
\item \texttt{zoo.data}: the function \texttt{setData} tries to convert \texttt{original.data} into an object of class \texttt{zoo}. The coerced \texttt{zoo} data are stored in the slot \texttt{zoo.data}. If the conversion fails the function exits with an error. Internally, the \texttt{yuima} package stores and operates on \texttt{zoo}-type objects.
\end{itemize}

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

Value

\begin{itemize}
\item \texttt{value} a list of object(s) of \texttt{yuima.data-class} for \texttt{setData}. The content of the \texttt{zoo.data} slot for \texttt{get.zoo.data}
\end{itemize}
setFunctional

Author(s)

The YUIMA Project Team

Examples

```r
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**

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

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 `F` are 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

```r
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)
```
Description

'setHawkes' constructs an object of class yuima.Hawkes that is a mathematical description of a multivariate Hawkes model.

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

- `lower.var`: Lower bound in the integral
- `upper.var`: Upper bound in the integral
- `var.dt`: Time variable
- `process`: Counting process
- `dimension`: An integer that indicates the components of the counting process
- `intensity`: Intensity Process
- `ExpKernParm1`: Kernel parameters
- `ExpKernParm2`: Kernel parameters
- `const`: Constant term in the intensity process
- `measure`: Jump size. By default 1
- `measure.type`: Type. By default code.

Details

By default the object is an univariate Hawkes process.

Value

The function returns an object of class yuima.Hawkes.

Author(s)

YUIMA Team
Examples

```r
## Not run:
# Definition of an univariate hawkes model

provaHawkes2<-setHawkes()
str(provaHawkes2)

# Simulation

true.par <- list(nu1=0.5, c11=3.5, a11=4.5)

simprv1 <- simulate(object = provaHawkes2, true.parameter = true.par,
                      sampling = setSampling(Terminal =70, n=7000))

plot(simprv1)

# Computation of intensity

lambda1 <- Intensity.PPR(simprv1, param = true.par)

plot(lambda1)

# qmle

res1 <- qmle(simprv1, method="Nelder-Mead", start = true.par)

summary(res1)

## End(Not run)
```

**setIntegral**

*Integral of Stochastic Differential Equation*

**Description**

'setIntegral' is the constructor of an object of class `yuima.Integral`

**Usage**

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

**Arguments**

- **yuima**: an object of class `yuima.model` that is the SDE.
- **integrand**: A matrix or a vector of strings that describe each component of the integrand.
- **var.dx**: A label that indicates the variable of integration
### setIntegral

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower.var</td>
<td>A label that indicates the lower variable in the support of integration, by default lower.var = 0.</td>
</tr>
<tr>
<td>upper.var</td>
<td>A label that indicates the upper variable in the support of integration, by default upper.var = t.</td>
</tr>
<tr>
<td>out.var</td>
<td>Label for the output</td>
</tr>
<tr>
<td>nrow</td>
<td>Dimension of output if integrand is a vector of string.</td>
</tr>
<tr>
<td>ncol</td>
<td>Dimension of output if integrand is a vector of string.</td>
</tr>
</tbody>
</table>

**Value**

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

**Author(s)**

The YUIMA Project Team

**References**

Yuima Documentation

**Examples**

```r
## 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
# \[ I=\int_{0}^{t} b*exp(-a*(t-s))*(X_s-a1*s)dX_s \]
#
# 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 Integrand
Integral@Integral@Integrand@dimIntegrand

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

# the parameters in the integrand are $\left(b,a,a1\right)$
```


setLaw

Random variable constructor

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
**setLaw_th**

**Details**
Insert additional info

**Value**
object of class yuima.law

**Note**
Insert additional info

**Author(s)**

YUIMA TEAM

---

**Description**

`setLaw_th` constructs an object of class `yuima.th-class`.

**Usage**

```
setLaw_th(h = 1, method = "LAG", up = 7, low = -7, N = 180, N_grid = 1000, regular_par = NULL, ...)
```

**Arguments**

- **h**: a numeric object that is the time of the intervals
- **method**: Method for the inversion of the characteristic function. Three methods are available: cos, LAG, and FFT.
- **up**: Upper bound for the integration support.
- **low**: Lower bound for the integration support.
- **N**: Integration grid.
- **N_grid**: Number of points in the support.
- **regular_par**: A scalar for controlling the Gibbs effect for the inversion of the characteristic function
- **...**: Additional arguments. See `setLaw` for more details.

**Value**

The function returns an object of class `yuima.th-class`.

**Author(s)**

The YUIMA Project Team

Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>
setLRM

A constructor of a t-Student Regression Model.

Description

This function returns an object of `yuima.LevyRM-class`.

Usage

```r
setLRM(unit_Levy, yuima_regressors, LevyRM = "Y", coeff = c("mu", "sigma0"),
        data = NULL, sampling = NULL, characteristic = NULL, functional = NULL, ...)
```

Arguments

- `unit_Levy`: An object of `yuima.th-class` that describes the t-noise in the regression model.
- `yuima_regressors`: An object of `yuima.model-class` that represents the regressors.
- `LevyRM`: The label of the output variable. Default 'Y'.
- `coeff`: Labels for the regressor coefficients and the scale parameter.
- `data`: An object of `yuima.data-class` that contains simulated or real data.
- `sampling`: An object of `yuima.sampling-class`.
- `characteristic`: An object of `yuima.characteristic-class`.
- `functional`: An object of class `yuima.functional-class`.
- `...`: Additional arguments. See `setYuima`.

Value

An object of `yuima.LevyRM-class`.

Author(s)

The YUIMA Project Team

Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>
Description
'setMap' is the constructor of an object of class \texttt{yuima.Map} that describes a map of a SDE.

Usage

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

Arguments

- **func**: a matrix or a vector of strings that describe each component of the map.
- **yuima**: an object of class \texttt{yuima.model} that is the SDE.
- **out.var**: label for the output
- **nrow**: dimension of Map if \texttt{func} is a vector of string.
- **ncol**: dimension of output if \texttt{func} is a vector of string.

Value

The constructor returns an object of class \texttt{yuima.Map}.

Author(s)

The YUIMA Project Team

References

Yuima Documentation

Examples

```r
## 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,
```

setMap  

\textit{Map of a Stochastic Differential Equation}
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)

---

**setDescription**

*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**

```r
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 \texttt{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

\begin{verbatim}
# Ex 1. (One-dimensional diffusion process)
# To describe
# dX_t = -3*X_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_1_t = -3*X_1_t*dt + dW_1_t +X_2_t*dW_3_t,
# dX_2_t = -(X_1_t + 2*X_2_t)*dt + X_1_t*dW_1_t + 3*dW_2_t,
# 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.coeff="1", measure=list(intensity="1", df=list("dnorm(z, 0, 1)")),
measure.type="CP", solve.variable="x")
# Look at the model structure by
str(mod3)

# Ex 4. (Process with jumps (stable process))
# To describe
# dX_t = -theta*X_t*dt+sigma*dZ_t
\end{verbatim}
mod4 <- setModel(drift=c("-theta*x"), diffusion="sigma",
jump.coef=c("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 differential 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.coef=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*y*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:

\[ M_t = m_0 + \sum_{i=0}^{N_t} c \cdot Y_{\tau_i} \]

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**

\[ \text{setPoisson(intensity = 1, df = NULL, scale = 1, dimension=1, ...)} \]

**Arguments**

- **intensity**
  - either and expression or a numerical value representing the intensity function of the Poisson process \(N_t\).
- **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 `setModel`
Details
An object of \texttt{yuima.model-class} where the model slot is of class \texttt{yuima.poisson-class}.

Value
\begin{itemize}
  \item \texttt{model} \hspace{1em} an object of \texttt{yuima.model-class}.
\end{itemize}

Author(s)
The YUIMA Project Team

Examples
\begin{verbatim}
## Not run:
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=mu=0, sigma=2),sampling=samp)
plot(y5)
\end{verbatim}
```r
f <- function(t) beta*(1+sin(lambda*t))
curve(f, 0, Terminal, col="red")

## End(Not run)
```

---

**setPPR**

*Point Process*

**Description**

Constructor of a Point Process Regression Model

**Usage**

```r
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`: an object of `yuima.model-class` that describes the mathematical features of counting and covariates processes $Y[t]=(X[t],N[t])$.
- `counting.var`: a label denoting the name of the counting process.
- `gFun`: a vector string that is the mathematical expression of the vector function $g(t,Y[t-],theta)$ in the intensity process.
- `Kernel`: a matrix string that is the kernel $kappa(t-s,Y[s],theta)$ in the definition of the intensity process.
- `var.dx`: a string denoting the integration variable in the intensity process.
- `var.dt`: a string denoting the integration time variable in the intensity process.
- `lambda.var`: name of the intensity process.
- `lower.var`: Lower bound of the support for the integral in the definition of the intensity process.
- `upper.var`: Upper bound of the support for the integral in the definition of the intensity process.
- `nrow`: number of rows in the kernel.
- `ncol`: number of columns in the kernel.

**Value**

An object of `yuima.PPR`

**Note**

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

Set sampling information and create a ‘sampling’ object.

Description

setSampling is a constructor for yuima.sampling-class.

Usage

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

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

Details

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

Initial: initial time of the grid.
Terminal: terminal time of 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 leftmost observation.

The slot random is used as information in case a grid is already determined (e.g. n or delta, etc. ot the grid itself are given) or if some subsampling has occurred or if some particular method which causes a random grid is used in simulation (for example the space discretized Euler scheme). The slot random contains a list of two elements distr and scale, where distr is the distribution of independent random times and scale is either a scaling constant or a scaling function. If the grid of times is deterministic, then random is FALSE.
If not specified and random=FALSE, the slot grid is filled automatically by the function. It is eventually modified or created after the call to the function `simulate`.

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

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

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

Value

An object of type `yuima.sampling-class`.

Author(s)

The YUIMA Project Team

Examples

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

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

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

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

---

**setYuima**

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

**Description**

`setYuima` constructs an object of `yuima-class`.

**Usage**

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

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

Details

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

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

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

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

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

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

Value

- an object of `yuima-class`.

Author(s)

The YUIMA Project Team

Examples

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

yuima <- setYuima(model=ymodel, sampling=ysamp)
str(yuima)
```
**Description**

This function simulates increments of bivariate Brownian motions with multi-scale lead-lag relationships introduced in Hayashi and Koike (2018a) by the multi-dimensional circulant embedding method of Chan and Wood (1999).

**Usage**

`simBmllag(n, J, rho, theta, delta = 1/2^(J + 1), imaginary = FALSE)`  
`simBmllag.coef(n, J, rho, theta, delta = 1/2^(J + 1))`

**Arguments**

- `n`: the number of increments to be simulated.  
- `J`: a positive integer to determine the finest time resolution: $2^{-(J+1)}$ is regarded as the finest time resolution.  
- `rho`: a vector of scale-by-scale correlation coefficients. If `length(rho) < J`, zeros are appended to make the length equal to J.  
- `theta`: a vector of scale-by-scale lead-lag parameters. If `length(theta) < J`, zeros are appended to make the length equal to J.  
- `delta`: the step size of time increments. This must be smaller than or equal to $2^{-(J+1)}$.  
- `imaginary`: logical. See ‘Details’.

**Details**

Let $B(t)$ be a bivariate Gaussian process with stationary increments such that its marginal processes are standard Brownian motions and its cross-spectral density is given by Eq.(14) of Hayashi and Koike (2018a). The function `simBmllag` simulates the increments $B(i\delta) - B((i-1)\delta), i = 1, \ldots, n$. The parameters $R_j$ and $theta_j$ in Eq.(14) of Hayashi and Koike (2018a) are specified by rho and theta, while $\delta$ and $n$ are specified by delta and n, respectively.

Simulation is implemented by the multi-dimensional circulant embedding algorithm of Chan and Wood (1999). The last step of this algorithm returns a bivariate complex-valued sequence whose real and imaginary parts are independent and has the same law as $B(k\delta) - B((k-1)\delta), k = 1, \ldots, n$; see Step 3 of Chan and Wood (1999, Section 3.2). If `imaginary = TRUE`, the function `simBmllag` directly returns this bivariate complex-valued sequence, so we obtain two sets of simulated increments of $B(t)$ by taking its real and complex parts. If `imaginary = FALSE` (default), the function returns only the real part of this sequence, so we directly obtain simulated increments of $B(t)$.

The function `simBmllag.coef` is internally used to compute the sequence of coefficient matrices $R(k)\Lambda(k)^{1/2}$ in Step 2 of Chan and Wood (1999, Section 3.2). This procedure can be implemented before generating random numbers. Since this step typically takes the most computational cost,
this function is useful to reduce computational time when we conduct a Monte Carlo simulation for 
$$(B(k\delta) - B((k - 1)\delta))_{k=1}^n$$ with a fixed set of parameters. See ‘Examples’ for how to use this function to simulate 
$$(B(k\delta) - B((k - 1)\delta))_{k=1}^n$$.

Value

simBmllag returns a n x 2 matrix if imaginary = FALSE (default). Otherwise, simBmllag returns a complex-valued n x 2 matrix.

simBmllag.coef returns a complex-valued m x 2 x 2 array, where m is an integer determined by the rule described at the end of Chan and Wood (1999, Section 2.3).

Note

There are typos in the first and second displayed equations in page 1221 of Hayashi and Koike (2018a): The $j$-th summands on their right hand sides should be multiplied by $2^j$.

Author(s)

Yuta Koike with YUIMA project Team

References


See Also

wllag

Examples

```r
## Not run:
## Example 1
## Simulation setting of Hayashi and Koike (2018a, Section 4).

n <- 15000
J <- 13
rho <- c(0.3, 0.5, 0.7, 0.5, 0.5, 0.5, 0.5, 0.5)
theta <- c(-1, -1, -2, -2, -3, -5, -7, -10)/2^(J + 1)
set.seed(123)

dB <- simBmllag(n, J, rho, theta)
str(dB)
n/2^(J + 1) # about 0.9155
```
sum(dB[,1]^2) # should be close to n/2^(J + 1)
sum(dB[,2]^2) # should be close to n/2^(J + 1)

# Plot the sample path of the process
B <- apply(dB, 2, "diffinv") # construct the sample path
Time <- seq(0, by = 1/2^(J+1), length.out = n) # Time index
plot(zoo(B, Time), main = "Sample path of B(t)"

# Using simBmllag.coef to implement the same simulation
a <- simBmllag.coef(n, J, rho, theta)
m <- dim(a)[1]

set.seed(123)
z1 <- rnorm(m) + 1i * rnorm(m)
z2 <- rnorm(m) + 1i * rnorm(m)
y1 <- a[,1,1] * z1 + a[,1,2] * z2
y2 <- a[,2,1] * z1 + a[,2,2] * z2
dW <- mvfft(cbind(y1, y2))[1:n, ]/sqrt(m)
dB2 <- Re(dW)

plot(diff(dB - dB2)) # identically equal to zero

## Example 2
## Simulation Scenario 2 of Hayashi and Koike (2018b, Section 5).

# Simulation of Bm driving the log-price processes
n <- 30000
J <- 14
rho <- c(0.3,0.5,0.7,0.5,0.5,0.5,0.5,0.5)
theta <- c(-1,-1, -2, -2, -3, -5, -7, -10)/2^(J + 1)

dB <- simBmllag(n, J, rho, theta)

# Simulation of Bm driving the volatility processes
R <- -0.5 # leverage parameter
delta <- 1/2^(J+1) # step size of time increments
dW1 <- R * dB[,1] + sqrt(1 - R^2) * rnorm(n, sd = sqrt(delta))
dW2 <- R * dB[,2] + sqrt(1 - R^2) * rnorm(n, sd = sqrt(delta))

# Simulation of the model by the simulate function
dW <- rbind(dB[,1], dB[,2], dW1, dW2) # increments of the driving Bm

# defining the yuima object
drift <- c(0, 0, "kappa*(eta - x3)", "kappa*(eta - x4)"
diffusion <- diag(4)
diag(diffusion) <- c("sqrt(max(x3,0))", "sqrt(max(x4,0))",
"xi*sqrt(max(x3,0))", "xi*sqrt(max(x4,0))")
xinit <- c(0,0,"rgamma(1, 2* kappa* eta/xi^2,2*kappa/ xi^2)",
"rgamma(1, 2* kappa* eta/ xi^2, 2*kappa/ xi^2)"
mod <- setModel(drift = drift, diffusion = diffusion,
xinit = xinit, state.variable = c("x1","x2","x3","x4"))
samp <- setSampling(Terminal = n * delta, n = n)
yuima <- setYuima(model = mod, sampling = samp)

# simulation
result <- simulate(yuima, increment.W = dW,
                   true.parameter = list(kappa = 5, eta = 0.04, xi = 0.5))
plot(result)
## End(Not run)

simCIR Simulation of the Cox-Ingersoll-Ross diffusion

Description
This is a function to simulate a Cox-Ingersoll-Ross process given via the SDE
dX_t = (\alpha - \beta X_t)dt + \sqrt{\gamma X_t}dW_t

with a Brownian motion (W_t)_{t \geq 0} and parameters \alpha, \beta, \gamma > 0. We use an exact CIR simulator for
(X_{t_j})_{j=1,...,n} through the non-central chi-squares distribution.

Usage
simCIR(time.points, n, h, alpha, beta, gamma, equi.dist=FALSE )

Arguments
alpha, beta, gamma
  numbers given as in the SDE above.
equi.dist
  a logical value indicating whether the sampling points are equidistant (default equi.dist=FALSE).
n
  a number indicating the quantity of sampling points in the case equi.dist=TRUE.
h
  a number indicating the step size in the case equi.dist=TRUE.
time.points
  a numeric vector of sampling times (necessary if equi.dist=FALSE).

Value
A numeric matrix containing the realization of \((t_0, X_{t_0}),\ldots,(t_n, X_{t_n})\) with \(t_j\) denoting the \(j\)-th sampling times.

Author(s)
Nicole Hufnagel
Contacts: <nicole.hufnagel@math.tu-dortmund.de>
References


Examples

```r
## You always need the parameters alpha, beta and gamma
## Additionally e.g. time.points
data <- simCIR(alpha=3,beta=1,gamma=1,
               time.points = c(0,0.1,0.2,0.25,0.3))
## or n, number of observations, h, distance between observations,
## and equi.dist=TRUE
data <- simCIR(alpha=3,beta=1,gamma=1,n=1000,h=0.1,equi.dist=TRUE)
plot(data[,1],data[,2], type="l",col=4)

## If you input every value and equi.dist=TRUE, time.points are not
## used for the simulations.
data <- simCIR(alpha=3,beta=1,gamma=1,n=1000,h=0.1,
               time.points = c(0,0.1,0.2,0.25,0.3),
               equi.dist=TRUE)

## If you leave equi.dist=FALSE, the parameters n and h are not
## used for the simulation.
data <- simCIR(alpha=3,beta=1,gamma=1,n=1000,h=0.1,
               time.points = c(0,0.1,0.2,0.25,0.3))
```

---

**simFunctional**

*Calculate the value of functional*

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

Fe a real value

Note

we need to fix this routine.

Author(s)

YUIMA Project Team

Examples

set.seed(123)
# to the Black-Scholes economy:
# dXt^e = Xt^e * dt + e * Xt^e * dWt
diff.matrix <- matrix( c("x*e"), 1,1)
model <- setModel(drift = c("x"), diffusion = diff.matrix)
# call option is evaluated by 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
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, ...)


simulate

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>an yuima-class, yuima.model-class or yuima.carma-class object.</td>
</tr>
<tr>
<td>xinit</td>
<td>initial value vector of state variables.</td>
</tr>
<tr>
<td>true.parameter</td>
<td>named list of parameters.</td>
</tr>
<tr>
<td>space.discretized</td>
<td>flag to switch to space-discretized Euler Maruyama method.</td>
</tr>
<tr>
<td>increment.W</td>
<td>to specify Wiener increment for each time tics in advance.</td>
</tr>
<tr>
<td>increment.L</td>
<td>to specify Levy increment for each time tics in advance.</td>
</tr>
<tr>
<td>method</td>
<td>string Variable for simulation scheme. The default value method=euler uses</td>
</tr>
<tr>
<td></td>
<td>the euler discretization for the simulation of a sample path.</td>
</tr>
<tr>
<td>nsim</td>
<td>Not used yet. Included only to match the standard generic in package stats.</td>
</tr>
<tr>
<td>seed</td>
<td>Not used yet. Included only to match the standard generic in package stats.</td>
</tr>
<tr>
<td>hurst</td>
<td>value of Hurst parameter for simulation of the fGn. Overrides the specified</td>
</tr>
<tr>
<td></td>
<td>hurst slot.</td>
</tr>
<tr>
<td>methodfGn</td>
<td>simulation methods for fractional Gaussian noise.</td>
</tr>
<tr>
<td>...</td>
<td>passed to setSampling to create a sampling</td>
</tr>
<tr>
<td>sampling</td>
<td>a yuima.sampling-class object.</td>
</tr>
<tr>
<td>subsampling</td>
<td>a yuima.sampling-class object.</td>
</tr>
</tbody>
</table>

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 with/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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yuima</td>
<td>a yuima-class object.</td>
</tr>
</tbody>
</table>

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).
Examples

set.seed(123)

# Path-simulation for 1-dim diffusion process.
# \( \text{d}X_t = -0.3 \times X_t \times \text{d}t + \text{dW}_t \)
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.
# \( \text{d}X_t = \theta \times X_t \times \text{d}t + \text{dW}_t \)
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:
# \( X=(X_1,X_2,X_3); \text{d}X_t = U(t,X_t) \times \text{d}t + V(t) \times \text{dW}_t \)
# 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-x)*dt + b*sqrt(x)*dWt
sq <- function(x){y = 0;if(x>0){y = sqrt(x);};return(y);}
model<- setModel(drift="0.8*(0.2-x)",
diffusion="0.5*sq(x)",solve.variable=c("x"))
T<-10
n<-1000
sampling <- setSampling(Terminal=T,n=n)
yuima<-setYuima(model=model, sampling=sampling)
cir<-simulate(yuima,xinit=0.1)
plot(cir)

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

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

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

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

## Solve SDEs using Euler-Maruyama method.
yuima.mod <- simulate(yuima.mod,
  xinit=1,
  space.discretized=FALSE,
  increment.W=my.dW)
if( !is.null(yuima.mod) ){
  dev.new()
  # x11()
  plot(yuima.mod)
}

## A multi-dimensional (correlated) diffusion process.
## To describe the following model:
## X=(X1,X2,X3); dXt = U(t,Xt)dt + V(t)dWt
## For drift 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)"),
simulate

"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()
  # x1()
  plot(yuima.obj.path)
}

##:: sample for Levy process ("CP" type)
## specify the jump term as c(x,t)dz
obj.model <- setModel(drift="-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()}
## 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))

## 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)

## sample for multidimensional Levy process ("code" type)
## dX = \( \theta - A X \) dt + dZ,
## \( \theta = (\theta_1, \theta_2) = (1,0.5) \)
## \( A = \begin{bmatrix} a_{ij}\end{bmatrix} \), \( a_{11} = 2, a_{12} = 1, a_{21} = 1, a_{22} = 2 \)
require(yuima)
x0 <- c(1,1)
theta <- c(1,0.5)
A <- matrix(c(2,-1,-1,2),2,2)

result <- simulate(obj.yuima, xinit=x0, true.par=list(alpha=alpha,
beta=beta, delta0=delta0, mu=mu, Lambda=Lambda))

# 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)

---

**snr**

*Calculating self-normalized residuals for SDEs.*

**Description**

Calculate self-normalized residuals based on the Gaussian quasi-likelihood estimator.
Usage

\texttt{snr(yuima, start, lower, upper, withdrift)}

Arguments

\begin{itemize}
  \item \texttt{yuima} a yuima object.
  \item \texttt{lower} a named list for specifying lower bounds of parameters.
  \item \texttt{upper} a named list for specifying upper bounds of parameters.
  \item \texttt{start} initial values to be passed to the optimizer.
  \item \texttt{withdrift} use drift information for constructing self-normalized residuals. by default, \texttt{withdrift = FALSE}
\end{itemize}

Details

This function calculates the Gaussian quasi maximum likelihood estimator and associated self-normalized residuals.

Value

\begin{itemize}
  \item \texttt{estimator} Gaussian quasi maximum likelihood estimator
  \item \texttt{snr} self-normalized residuals based on the Gaussian quasi maximum likelihood estimator
\end{itemize}

Author(s)

The YUIMA Project Team

Contacts: Yuma Uehara <y-uehara@ism.ac.jp>

References


Examples

```r
## Not run:
# Test code (1. diffusion case)
yuima.mod <- setModel(drift="-theta*x", diffusion="theta1/sqrt(1+x^2)")
n <- 10000
ysamp <- setSampling(Terminal=n^(1/3), n=n)
yuima <- setYuima(model=yuima.mod, sampling=ysamp)
set.seed(123)
yuima <- simulate(yuima, xinit=0, true.parameter = list(theta=2,theta1=3))
start=list(theta=3,theta1=0.5)
lower=list(theta=1,theta1=0.3)
upper=list(theta=5,theta1=3)
res <- snr(yuima,start,lower,upper)
str(res)
```
# Test code (2. jump diffusion case)
a<-3
b<-5
mod <- setModel(drift="10-theta*x", #drift="10-3*x/(1+x^2)",
diffusion="theta1*(2+x^2)/(1+x^2)",
jump.coeff="1",
# measure=list(intensity="10",df=list("dgamma(z, a, b)")),
measure=list(intensity="10",df=list("dunif(z, a, b)")),
measure.type="CP")

T <- 100 ## Terminal
n <- 10000 ## generation size
samp <- setSampling(Terminal=T, n=n) ## define sampling scheme
yuima <- setYuima(model = mod, sampling = samp)

yuima <- simulate(yuima, xinit=1,
true.parameter=list(theta=2,theta1=sqrt(2),a=a,b=b),
sampling = samp)
start=list(theta=3,theta1=0.5)
lower=list(theta=1,theta1=0.3)
upper=list(theta=5,theta1=3)
res <- snr(yuima,start,lower,upper)
str(res)

## 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 cumulative 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 yuima-class or yuima.data-class.
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 \text{dim}(x) 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-th component of noise.var must indicates the variance of the noise for the 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 \text{as.vector}(covmat)
Sigma.p the pilot estimates of the spot covariance matrix
Author(s)
Yuta Koike with YUIMA Project Team

References


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

**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

```r
## 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,
solvevariable=c("x1","x2"), xinit=c(3,2))
set.seed(111)

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

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

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

---

**Description**

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

**Usage**

```r
## S3 method for class 'yuima'
\texttt{toLatex}(object,\ldots)

## S3 method for class 'yuima.model'
\texttt{toLatex}(object,\ldots)

## S3 method for class 'yuima.carma'
\texttt{toLatex}(object,\ldots)

## S3 method for class 'yuima.cogarch'
\texttt{toLatex}(object,\ldots)
```

**Arguments**

- `object` object of a class yuima, yuima.model or yuima.carma.
- `\ldots` 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**

```r
# dXt = \theta * X_t * dt + dW_t
mod1 <- setModel(drift="\theta * y", diffusion=1, solve.variable=c("y"))
str(mod1)
\texttt{toLatex}(mod1)

# A multi-dimensional (correlated) diffusion process.
# To describe the following model:
```
variable.Integral: Class for the mathematical description of integral of a stochastic process

Description

Auxiliar class for definition of an object of class yuima.Integral. see the documentation of yuima.Integral for more details.
Description

This function estimates lead-lag parameters on a scale-by-scale basis from non-synchronously observed bivariate processes, using the estimators proposed in Hayashi and Koike (2018b).

Usage

```r
wllag(x, y, J = 8, N = 10, tau = 1e-3, from = -to, to = 100,
       verbose = FALSE, in.tau = FALSE, tol = 1e-6)
```

Arguments

- `x`: a `zoo` object for observation data of the first process.
- `y`: a `zoo` object for observation data of the second process.
- `J`: a positive integer. Scale-by-scale lead-lag parameters are estimated up to the level J.
- `N`: The number of vanishing moments of Daubechies’ compactly supported wavelets. This should be an integer between 1 and 10.
- `tau`: the step size of a finite grid on which objective functions are evaluated. Note that this value is identified with the finest time resolution of the underlying model. The default value `1e-3` corresponds to 1 mili-second if the unit time corresponds to 1 second.
- `from`: a negative integer. `from*tau` gives the lower end of a finite grid on which objective functions are evaluated.
- `to`: a positive integer. `to*tau` gives the upper end of a finite grid on which objective functions are evaluated.
- `verbose`: a logical. If `FALSE` (default), the function returns only the estimated scale-by-scale lead-lag parameters. Otherwise, the function also returns some other statistics such as values of the signed objective functions. See ‘Value’.
- `in.tau`: a logical. If `TRUE`, the estimated lead-lag parameters are returned in increments of `tau`. That is, the estimated lead-lag parameters are divided by `tau`.
- `tol`: tolerance parameter to avoid numerical errors in comparison of time stamps. All time stamps are divided by `tol` and rounded to integers. A reasonable choice of `tol` is the minimum unit of time stamps. The default value `1e-6` supposes that the minimum unit of time stamps is greater or equal to 1 micro-second.

Details

Hayashi and Koike (2018a) introduced a bivariate continuous-time model having different lead-lag relationships at different time scales. The wavelet cross-covariance functions of this model, computed based on the Littlewood-Paley wavelets, have unique maximizers in absolute values at
each time scale. These maximizers can be considered as lead-lag parameters at each time scale. To estimate these parameters from discrete observation data, Hayashi and Koike (2018b) constructed objective functions mimicking behavior of the wavelet cross-covariance functions of the underlying model. Then, estimates of the scale-by-scale lead-lag parameters can be obtained by maximizing these objective functions in absolute values.

Value

If verbose is FALSE, a numeric vector with length \( J \), corresponding to the estimated scale-by-scale lead-lag parameters, is returned. Note that their positive values indicate that the first process leads the second process.

Otherwise, an object of class "yuima.wllag", which is a list with the following components, is returned:

- **lagtheta**: the estimated scale-by-scale lead-lag parameters. The \( j \) th component corresponds to the estimate at the level \( j \). A positive value indicates that the first process leads the second process.
- **obj.values**: the values of the objective functions evaluated at the estimated lead-lag parameters.
- **obj.fun**: a list of values of the objective functions. The \( j \) th component of the list corresponds to a zoo object for values of the signed objective function at the level \( j \) indexed by the search grid.
- **theta.hry**: the lead-lag parameter estimate in the sense of Hoffmann, Rosenbaum and Yoshida (2013).
- **cor.hry**: the correlation coefficient in the sense of Hoffmann, Rosenbaum and Yoshida (2013), evaluated at the estimated lead-lag parameter.
- **ccor.hry**: a zoo object for values of the cross correlation function in the sense of Hoffmann, Rosenbaum and Yoshida (2013) indexed by the search grid.

Note

Smaller levels correspond to finer time scales. In particular, the first level corresponds to the finest time resolution, which is defined by the argument \( \text{tau} \).

If there are multiple maximizers in an objective function, \texttt{wllag} takes a maximizer farthest from zero (if there are two such values, the function takes the negative one). This behavior is different from \texttt{llag}.

The objective functions themselves do NOT consistently estimate the corresponding wavelet covariance functions. This means that values in \texttt{obj.values} and \texttt{obj.fun} cannot be interpreted as covariance estimates (their scales depend on the degree of non-synchronicity of observation times).

Author(s)

Yuta Koike with YUIMA Project Team
References


See Also

`simBmllag`, `llag`

Examples

```r
## Not run:
## An example from a simulation setting of Hayashi and Koike (2018b)
set.seed(123)

# Simulation of Bm driving the log-price processes
n <- 15000
J <- 13
tau <- 1/2^(J+1)

rho <- c(0.3, 0.5, 0.7, 0.5, 0.5, 0.5, 0.5, 0.5)
theta <- c(-1, -1, -2, -2, -3, -5, -7, -10) * tau

dB <- simBmllag(n, J, rho, theta)

Time <- seq(0, by = tau, length.out = n) # Time index
x <- zoo(diffinv(dB[,1]), Time) # simulated path of the first process
y <- zoo(diffinv(dB[,2]), Time) # simulated path of the second process

# Generate non-synchronously observed data
x <- x[as.logical(rbinom(n + 1, size = 1, prob = 0.5))]
y <- y[as.logical(rbinom(n + 1, size = 1, prob = 0.5))]

# Estimation of scale-by-scale lead-lag parameters (compare with theta/tau)
wllag(x, y, J = 8, tau = tau, tol = tau, in.tau = TRUE)

# Estimation with other information
out <- wllag(x, y, tau = tau, tol = tau, in.tau = TRUE, verbose = TRUE)
out

# Plot of the HRY cross-correlation function
plot(out$hry, xlab = expression(theta), ylab = expression(U(theta)))

# Plot of the objective functions
op <- par(mfrow = c(4,2))
plot(out)
par(op)
```
### ybook

**Description**

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

**Usage**

```r
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**

```r
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", characteristic = "yuima.characteristic": 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", characteristic = "yuima.characteristic": 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 signature(x = "yuima"): bind yuima.data object.

Author(s)

The YUIMA Project Team

---

**yuima.ae-class**

Class for the asymptotic expansion of diffusion processes

Description

The yuima.ae class is used to describe the output of the functions ae and aeMarginal.
yuima.carma-class

Slots

order integer. The order of the expansion.
var character. The state variables.
u.var character. The variables of the characteristic function.
eps.var character. The perturbation variable.
characteristic expression. The characteristic function.
density expression. The probability density function.
Z0 numeric. The solution to the deterministic process obtained by setting the perturbation to zero.
Mu numeric. The drift vector for the representation of Z1.
Sigma matrix. The diffusion matrix for the representation of Z1.
c.gamma list. The coefficients of the Hermite polynomials.
h.gamma list. Hermite polynomials.

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: whether 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 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.carmaHawkes-class

*Class for the mathematical description of a Hawkes process with a CARMA(p,q) intensity*

---

Description

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

Slots

info: is an carmaHawkes.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: whether jump.coeff include jump.variable.
yuima.characteristic-class

Class 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 a 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.
- parameter: is an object of class model.parameter-class.
yuima.CP.qmle-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.coeff 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 La-
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**

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

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

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

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

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

Objects in this class contain two slots described here.

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

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

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

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

If data are obtained by simulation, the original.data slot is usually empty.
yuima.functional-class

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.

cc 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**

The `yuima.Hawkes` class is a class of the `yuima` package that extends the `yuima.PPR-class`. The object of this class contains all the information about the Hawkes process with exponential kernel. An object of this class can be created by calls of the function `setHawkes`.

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

\[ z_t = \int_0^t h(\theta, X_s, s) \, dX_s \]

**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: A mathematical description for the noise.

Description

A yuima class that contains all information on the noise. This class is a bridge between a yuima.model-class and a noise constructed by users.

Slots

- **rng**  A user defined function that generates the noise sample.
- **density**  A user defined function that is the density of the noise at time t.
- **cdf**  A user defined function that is the cumulative distribution function of the noise at time t.
- **quantile**  A user defined function that is the quantile of the noise at time t.
- **characteristic**  A user defined function that is the characteristic function of the noise at time t.
- **param.measure**  A character object that contains the parameters of the noise.
- **time.var**  the label of the time variable.
- **dim**  Dimension of the noise

Methods

- **rand**  signature(object = "yuima.law", n = "numeric", param = "list", ...): This method returns a sample of the noise, n is the sample size.
- **dens**  signature(object = "yuima.law", x = "numeric", param = "list", log = FALSE, ...): This method returns the density of the noise, x is the vector of the support.
- **cdf**  signature(object = "yuima.law", q = "numeric", param = "list", ...): This method returns the cdf of the noise, q is the vector of the support.
- **quant**  signature(object = "yuima.law", p = "numeric", param = "list", ...): This method returns the quantile of the noise, p is the vector of the support.
- **char**  signature(object = "yuima.law", u = "numeric", param = "list", ...): This method returns the characteristic function of the noise, u is the vector of the support.

Author(s)

The YUIMA Project Team

Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>
yuima.LevyRM-class
yuima.LevyRM: A class for the mathematical description of the t-Student regression model.

Description

A yuima class that contains all information on the regression model with t-student Levy process noise. This class extends yuima-class and contains information on the regressors used in the definition of the model. The regressors are represented by an object of yuima.model-class. An object of this class can be created by calls of the function setLRM.

Methods

initialize Initialize method. It makes a copy of each argument.
simulate simulation method. For more information see simulate.

Author(s)

The YUIMA Project Team
Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>

yuima.Map-class
Class for the mathematical description of function of a stochastic process

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
\[ zt = g(\theta, X_t, t) : \mathbb{R}^{q \times d \times 1} \rightarrow \mathbb{R}^{l1 \times l2 \times \ldots} \]
or an operator between two independent stochastic process:
\[ zt = h(\theta, X_t, Y_t, t) \]
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 \(g(\theta, X_t, t)\) or the operator \(h(\theta, X_t, Y_t, t)\)
donimension: 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.
  all.param: labels of all parameters (model and map/operators).
  all.paramMap: 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

Classes for the mathematical description of stochastic differential equations

Description

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

Slots

drift: is an \( R \) expression which specifies the drift coefficient (a vector).
diffusion: is an \( R \) expression which specifies the diffusion coefficient (a matrix).
hurst: the Hurst parameter of the gaussian noise. If \( h=0.5 \), the process is Wiener otherwise it is fractional Brownian motion with that precise value of the Hurst index. Can be set to \( NA \) for further specification.
jump.coeff: a matrix of expressions for the jump component.
measure: Levy measure for jump variables.
measure.type: Type specification for Levy measures.
state.variable: a vector of names identifying the names used to denote the state variable in the drift and diffusion specifications.
parameter: which is a short name for “parameters”, is an object of class model.parameter-class.
For more details see model.parameter-class documentation page.
state.variable: identifies the state variables in the \( R \) expression.
jump.variable: identifies the variable for the jump coefficient.
time.variable: the time variable.
noise.number: denotes the number of sources of noise. Currently only for the Gaussian part.
equation.number: denotes the dimension of the stochastic differential equation.
dimension: the dimensions of the parameter given in the parameter slot.
solve.variable: identifies the variable with respect to which the stochastic differential equation
has to be solved.
xinit: contains the initial value of the stochastic differential equation.
J.flag: wheather jump.coeff include jump.variable.

Author(s)
The YUIMA Project Team

---

yuima.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

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

## We define the random number generator

```r
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

```r
mod <- setModel(drift = matrix(c("0","0","0"),3,1), diffusion = NULL,
  jump.coef = matrix(c("1","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

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

## Simulation

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

## Plot

```r
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 definition

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","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 Compound 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))
}
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(\(\emptyset\)).
hurst: always \(h=0.5\), but ignored for this model.
jump.coeff: set according to scale in \texttt{setPoisson}.
measure: a list containing the intensity measure and the jump distribution.
measure.type: always "CP".
\textbf{state.variable} a vector of names identifying the names used to denote the state variable in the drift and diffusion specifications.

\textbf{parameter}: which is a short name for “parameters”, is an object of class \texttt{model.parameter-class}.

For more details see \texttt{model.parameter-class} documentation page.

\textbf{state.variable}: identifies the state variables in the R expression.
\textbf{jump.variable}: identifies the variable for the jump coefficient.
\textbf{time.variable}: the time variable.
\textbf{noise.number}: denotes the number of sources of noise.
\textbf{equation.number}: denotes the dimension of the stochastic differential equation.
\textbf{dimension}: the dimensions of the parameter given in the parameter slot.
\textbf{solve.variable}: identifies the variable with respect to which the stochastic differential equation has to be solved.
\textbf{xinit}: contains the initial value of the stochastic differential equation.
\textbf{J.flag}: whether jump.coeff include jump.variable.

\textbf{Methods}

\texttt{simulate} simulation method. For more information see \texttt{simulate}.
\texttt{qmle} Quasi maximum likelihood estimation procedure. For more information see \texttt{qmle}.

\textbf{Author(s)}

The YUIMA Project Team

---

\textbf{yuima.PPR} \hspace{1cm} \textit{Class for a mathematical description of a Point Process}

---

\textbf{Description}

The \texttt{yuima.PPR} class is a class of the \texttt{yuima} package that extends the \texttt{yuima-class}. The object of this class contains all the information about the Point Process Regression Model.

\textbf{Objects from the Class}

Objects can be created by calls of the function \texttt{setPPR}.
yuima.qmleLevy.incr

Slots

PPR: is an object of class info.PPR.
gFun: is an object of class info.Map.
Kernel: is an object of class Integral.sde.
data: is an object of class yuima.data-class. The slot contain either true data or simulated data
model: is an object of class yuima.model-class. The slot contains all the information about the
covariates
sampling: is an object of class yuima.sampling-class.
characteristic: is an object of class yuima.characteristic-class.
model: is an object of class yuima.functional-class.

Author(s)

The YUIMA Project Team

yuima.qmleLevy.incr Class for Quasi Maximum Likelihood Estimation of Levy SDE model

Description

The yuima.qmleLevy.incr-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 yuima.data-class that contains the estimated increments of the
noise.
logL.Incr: an numeric object that represents the value of the loglikelihood for the estimated Levy
increments.
minusloglLevy: an R function that evaluates the loglikelihood of the estimated Levy increments.
The function is used internally in qmleLevy for the estimation of the Levy measure parameters.
Levydetails: a list containing additional information about the optimization procedure in the
estimation of the Levy measure parameters. See optim help for the meaning of the compo-
nents of this list.
Data: is an object of yuima.data-class containing observation data.
model: is an object of of yuima.carma-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.
nobs: an object of class numeric.
method: is an object of class character.
Methods

Methods mle  All methods for mle-class are available.

Author(s)

The YUIMA Project Team

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

Description

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

Details

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

Slots

Initial: initial time of the grid.
Terminal: terminal time of 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
yuima.snr-class

Class "yuima.snr" for self-normalized residuals of SDE "yuima" class object

Description

The yuima.snr-class is a class of the yuima package used to store the calculated self-normalized residuals of an SDEs.

Slots

call: The original call.
coef: A numeric vector.
snr: A numeric vector of residuals.
model: A yuima.model.

Methods

show print method

Author(s)
The YUIMA Project Team

yuima.th-class

yuima.th-class: A mathematical description for the t-Levy process.

Description

A yuima class that contains all information on the noise for t-Levy process. This class extends yuima.law-class and contains info on the numerical method used for the inversion of the characteristic function. Three inversion methods are available: cos, Laguerre and FFT. An object of this class can be created by calls of the function setLaw_th.

Methods

rand signature(object = "yuima.th", n = "numeric", param = "list", ...): This method returns a sample of the noise, n is the sample size.
dens signature(object = "yuima.th", x = "numeric", param = "list", log = FALSE, ...): This method returns the density of the noise, x is the vector of the support.
cdf signature(object = "yuima.th", q = "numeric", param = "list", ...): This method returns the cdf of the noise, q is the vector of the support.
quant signature(object = "yuima.th", p = "numeric", param = "list", ...): This method returns the quantile of the noise, p is the vector of the support.
char signature(object = "yuima.th", u = "numeric", param = "list", ...): This method returns the characteristic function of the noise, u is the vector of the support.
Author(s)

The YUIMA Project Team
Contacts: Lorenzo Mercuri <lorenzo.mercuri@unimi.it>
Index

* CIR diffusion
  simCIR, 155
* COGARCH
  setCogarch, 130
* Carma
  Diagnostic.Carma, 50
* Estimation COGARCH
  gmm, 60
* Estimation
  qmleLevy, 116
* Information criteria
  IC, 65
* Method of Moments
  gmm, 60
* Noise
  Diagnostic.Carma, 50
* PPR Constructor
  setPPR, 147
* PPR model
  setPPR, 147
* classes
  carma.info-class, 21
carmaHawkes.info-class, 22
Class for Quasi Maximum Likelihood
  Estimation of Point Process
  Regression Models, 40
cogarch.est.-class, 41
cogarch.est.incr-class, 41
cogarch.info-class, 42
model.parameter-class, 93
yuima-class, 177
yuima.carma-class, 179
yuima.carma.qmle-class, 180
yuima.characteristic-class, 182
yuima.cogarch-class, 182
yuima.CP.qmle-class, 183
yuima.data-class, 184
yuima.functional-class, 185
yuima.law-class, 187
yuima.LevyRM-class, 188
yuima.Map-class, 188
yuima.model-class, 189
yuima.multimodel-class, 190
yuima.poisson-class, 193
yuima.PPR, 194
yuima.qmleLevy.incr, 195
yuima.sampling-class, 196
yuima.snr-class, 197
yuima.th-class, 197
* datasets
  LogSPX, 85
  MWK151, 96
* data
  fitCIR, 55
* misc
  toLatex, 172
  ybook, 177
* qmle
  qmleLevy, 116
* ts
  adaBayes, 4
  asymptotic_term, 17
  bns.test, 19
  CarmaNoise, 23
  cce, 25
cce.factor, 34
cogarchNoise, 43
CPPoint, 44
hyavar, 62
lasso, 73
limiting.gamma, 75
llag, 77
llag.test, 81
lm.jumptest, 84
lseBayes, 86
mmfrac, 92
mpv, 94
INDEX

noisy.sampling, 97
ntv, 99
phi.test, 102
poisson.random.sampling, 103
pz.test, 104
qgv, 105
qmle, 108
rconst, 119
rng, 120
setCarma, 125
setCharacteristic, 129
setData, 132
setFunctional, 133
setModel, 142
setPoisson, 145
setSampling, 148
setYuima, 150
simBmllag, 152
simFunctional, 156
simulate, 157
spectralcov, 166
subsampling, 170
wllag, 174

* yuima.cogarch
  setCogarch, 130

adaBayes, 4
adaBayes,yuima-method (adaBayes), 4
ae, 6, 178
aeCharacteristic, 8
aeDensity, 9
aeExpectation, 10
aeKurtosis, 11
aeMarginal, 12, 178
aeMean, 13
aeMoment, 14
aeSd, 15
aeSkewness, 16
arima0, 29
asymptotic_term, 17
asymptotic_term,yuima-method
  (asymptotic_term), 17
bns.test, 19, 85, 100, 105
bns.test,list-method (bns.test), 19
bns.test,yuima-method (bns.test), 19
bns.test,yuima.data-method (bns.test), 19
boot, 82

CARMA (setCarma), 125
Carma (setCarma), 125
carma.info-class, 21
carma.qmle (yuima.carma.qmle-class), 180
Carma.Recovering (CarmaNoise), 23
CarmaHawkes (setCarmaHawkes), 127
carHawkes.info-class, 22
CarmaNoise, 23, 109, 180
CarmaRecoverNoise (CarmaNoise), 23
cbind,yuima,ANY-method (yuima-class), 177
cbind.yuima (setData), 132
cbind.yuima,yuima.data-method
  (yuima.data-class), 184
cce, 25, 35, 38, 62, 63, 79, 82, 95, 98, 100, 104, 168, 178, 185
cce,yuima-method (yuima-class), 177
cce,yuima.data-method
  (yuima.data-class), 184
cce.factor, 31, 34
cdf (LawMethods), 74
cdf,yuima.law-method (yuima.law-class), 187
cdf,yuima.th-method (yuima.th-class), 197
class (LawMethods), 74
class,yuima.law-method
  (yuima.law-class), 187
class,yuima.th-method (yuima.th-class), 197

Class for Quasi Maximum Likelihood
  Estimation of Point Process
  Regression Models, 40

COGARCH (setCogarch), 130
CoGarch (setCogarch), 130
Cogarch (setCogarch), 130
cogarch (setCogarch), 130
cogarch.est-class (cogarch.est.-class), 41
cogarch.est.-class, 41
cogarch.est.incr-class, 41
cogarch.info-class, 42
cogarch.Recovering (cogarchNoise), 43
cogarchNoise, 42, 43
CogarchRecoverNoise (cogarchNoise), 43
CP.qmle (yuima.CP.qmle-class), 183
CPoint, 44
cubintegrate, 11
Data (LogSPX), 85
DataPPR, 48
dbgamma (rng), 120
dconst (rcost), 119
dens (LawMethods), 74
dens, yuima.law-method
  (yuima.law-class), 187
dens, yuima.th-method (yuima.th-class), 197
dGH (rng), 120
dGIG (rng), 120
Diagnostic.Carma, 50
Diagnostic.Cogarch, 51
dIG (rng), 120
dim, 132, 185
dim (setData), 132
dim, yuima-method (yuima-class), 177
dim, yuima.data-method
  (yuima.data-class), 184
dNIG (rng), 120
dvgamma (rng), 120
est.cogarch.incr-class
  (cogarch.est.incr-class), 41
Estimation of t-Levy Regression Model
  (estimation_LRM), 52
Estimation. LevyIncr (qmleLevy), 116
estimation_LRM, 52
estimation_RLM, yuima.LevyRM-function
  (yuima.LevyRM-class), 188
EstimCarmaHawkes, 53
F0 (simFunctional), 156
F0, yuima-method (simFunctional), 156
fitCIR, 55
Fnorm (simFunctional), 156
Fnorm, yuima-method (simFunctional), 156
FromCF2yuima_law, 56
get.counting.data, 57
get.zoo.data (setData), 132
get.zoo.data, yuima-method
  (yuima-class), 177
get.zoo.data, yuima.data-method
  (yuima.data-class), 184
gete (setFunctional), 133
gete, yuima.functional-method
  (yuima.functional-class), 185
getF (setFunctional), 133
getF, yuima.functional-method
  (yuima.functional-class), 185
getG (setFunctional), 133
getG, yuima.functional-method
  (yuima.functional-class), 185
glassoFast, 38
gmm, 41, 42, 60
Hawkes.Carma.Intensity
  (setCarmaHawkes), 127
hyavar, 30, 31, 62, 77–79, 82, 90
IC, 65
incr.qmleLevy (yuima.qmleLevy.incr), 195
info.Map, 195
info.Map (info.Map-class), 69
info.Map-class, 69
info.PPR, 69, 195
info.PPR-class (info.PPR), 69
initialize, carma.info-method
  (yuima.carma-class), 179
initialize, carmaHawkes.info-method
  (yuima.carmaHawkes-class), 181
initialize, cogarch.info-method
  (yuima.cogarch-class), 182
initialize, info.Map-method
  (info.Map-class), 69
initialize, info.PPR-method (info.PPR), 69
initialize,Integral.sde-method
  (Integral.sde), 69
initialize, Integrand-method
  (Integrand), 70
initialize, model.parameter-method
  (yuima.model-class), 189
initialize, param.Integral-method
  (param.Integral), 101
initialize, param.Map-method
  (param.Map-class), 102
initialize, variable.Integral-method
  (variable.Integral), 173
initialize, yuima-method (yuima-class), 177
initialize, yuima.ae-method
  (yuima.ae-class), 178
initialize,yuima.carma-method (yuima.carma-class), 179
initialize,yuima.carmaHawkes-method (yuima.carmaHawkes-class), 181
initialize,yuima.characteristic-method (yuima.characteristic-class), 182
initialize,yuima.cogarch-method (yuima.cogarch-class), 182
initialize,yuima.data-method (yuima.data-class), 184
initialize,yuima.functional-method (yuima.functional-class), 185
initialize,yuima.Hawkes-method (yuima.Hawkes), 186
initialize,yuima.Integral-method (yuima.Integral-class), 186
initialize,yuima.law-method (yuima.law-class), 187
initialize,yuima.LevyRM-method (yuima.LevyRM-class), 188
initialize,yuima.Map-method (yuima.Map-class), 188
initialize,yuima.model-method (yuima.model-class), 189
initialize,yuima.multimodel-method (yuima.multimodel-class), 190
initialize,yuima.poisson-method (yuima.poisson-class), 193
initialize,yuima.PPR-method (yuima.PPR), 194
initialize,yuima.qmleLevy.incr-method (yuima.qmleLevy.incr), 195
initialize,yuima.sampling-method (yuima.sampling-class), 196
initialize,yuima.th-method (yuima.th-class), 197
Integral.sde, 69, 195
Integral.sde-class (Integral.sde), 69
Integrand, 70
Integrand-class (Integrand), 70
Intensity.PPR, 70
JBtest, 71
lambdaFromData, 72
lasso, 73
LawMethods, 74
length, 132, 185
length (setData), 132
length,yuima-method (yuima-class), 177
length,yuima.data-method (yuima.data-class), 184
Levy.Carma (CarmaNoise), 23
Levy.cogarch (cogarchNoise), 43
LevySDE (qmleLevy), 116
limiting.gamma, 75
limiting.gamma,yuima-method (yuima-class), 177
limiting.gamma,yuima.carma-method (yuima.carma-class), 179
limiting.gamma,yuima.cogarch-method (yuima.cogarch-class), 182
limiting.gamma,yuima.model-method (yuima.model-class), 189
llag, 77, 82, 89, 90, 175, 176, 178, 185
llag, list-method (llag), 77
llag,yuima-method (yuima-class), 177
llag,yuima.data-method (yuima.data-class), 184
llag.test, 79, 81, 90
lm.jumptest, 21, 84, 100, 105
lmm, 31, 38, 98
lmm (spectralcov), 166
LogSPX, 85
lse (qmle), 108
LSE,yuima-method (yuima-class), 177
lseBayes, 86
lseBayes,yuima-method (lseBayes), 86
Map of SDE (setMap), 141
Map of yuima (setMap), 141
medrv, 95
medrv (ntv), 99
medrv.test, 21, 85, 105
Method of Moment COGARCH (gmm), 60
MethodOfMoments.CarmaHawkes (EstimCarmaHawkes), 53
minrv, 95
minrv (ntv), 99
minrv.test, 21, 85, 105
ml ql,yuima-method (yuima-class), 177
mllag, 79, 82, 88
mmfrac, 92, 107
model.parameter-class, 93
mpv, 21, 94, 100
mpv,yuima-method (mpv), 94
mpv,yuima.data-method (mpv), 94
INDEX

setIntegral, 136
setLaw, 138, 139
setLaw_th, 139, 197
setLRM, 140, 188
setMap, 141
setModel, 31, 66, 93, 142, 145
setPoisson, 145, 190, 194
setPPR, 147, 194
setSampling, 148, 158, 196
setYuima, 140, 150
show.yuima.snr-method
  (yuima.snr-class), 197
simBllag, 152, 176
simCIR, 155
simFunctional, 156
simFunctional.yuima-method
  (simFunctional), 156
simulate, 42, 150, 157, 178, 180, 183, 186,
  188, 189, 191, 194, 196
simulate,cogarch.est.incr-method
  (cogarch.est.incr-class), 41
simulate,yuima-method (yuima-class), 177
simulate,yuima.carma-method
  (yuima.carma-class), 179
simulate,yuima.cogarch-method
  (yuima.cogarch-class), 182
simulate,yuima.Hawkes-method
  (yuima.Hawkes), 186
simulate,yuima.Integral-method
  (yuima.Integral-class), 186
simulate,yuima.LevyRM-method
  (yuima.LevyRM-class), 188
simulate,yuima.Map-method
  (yuima.Map-class), 188
simulate,yuima.model-method
  (yuima.model-class), 189
simulate,yuima.multimodel-method
  (yuima.multimodel-class), 190
simulate,yuima.PPR-method (yuima.PPR), 194
snr, 164
spectralcov, 166
subsampling, 170, 196
subsampling,yuima-method (yuima-class), 177
subsampling,yuima.data-method
  (yuima.data-class), 184
t-Levy process (yuima.th-class), 197
toLatex, 172
variable.Integral, 173
variable.Integral-class
  (variable.Integral), 173
wllag, 153, 174
ybook, 177
yuima-class, 177
yuima.ae-class, 178
yuima.carma-class, 179
yuima.carma.qmle-class, 180
yuima.carmaHawkes-class, 181
yuima.characteristic-class, 182
yuima.cogarch-class, 182
yuima.CP.qmle-class, 183
yuima.data-class, 184
yuima.functional-class, 185
yuima.Hawkes, 69, 135, 186
yuima.Hawkes-class (yuima.Hawkes), 186
yuima.Integral, 69, 70, 101, 136, 137, 173
yuima.Integral (yuima.Integral-class), 186
yuima.Integral-class, 186
yuima.law (yuima.law-class), 187
yuima.law-class, 187
yuima.levyRM (yuima.levyRM-class), 188
yuima.levyRM-class, 188
yuima.Map, 69, 102, 141
yuima.Map (yuima.Map-class), 188
yuima.Map-class, 188
yuima.model, 136, 141, 183
yuima.model (yuima.model-class), 189
yuima.model-class, 189
yuima.multimodel
  (yuima.multimodel-class), 190
yuima.multimodel-class, 190
yuima.poisson-class, 193
yuima.PPR, 48, 57, 69, 147, 194
yuima.PPR-class (yuima.PPR), 194
yuima.PPR.qmle, ANY-method (Class for
  Quasi Maximum Likelihood
  Estimation of Point Process
  Regression Models), 40
yuima.PPR.qmle-class (Class for Quasi
  Maximum Likelihood Estimation
  of Point Process Regression
  Models), 40
yuima.qmle-class (yuima.CP.qmle-class), 183
yuima.qmleLevy.incr, 195
yuima.qmleLevy.incr, ANY-method
(yuima.qmleLevy.incr), 195
yuima.qmleLevy.incr-class
(yuima.qmleLevy.incr), 195
yuima.sampling-class, 196
yuima.snr-class, 197
yuima.th (yuima.th-class), 197
yuima.th-class, 197
zoo, 42, 78, 132, 174, 175, 178, 180, 184, 185