Package 'BSS'

October 12, 2022

Type Package

Title Brownian Semistationary Processes

Version 0.1.0

Description Efficient simulation of Brownian semistationary (BSS) processes using the hybrid simulation scheme, as described in
Bennedsen, Lunde, Pakkannen (2017) <arXiv:1507.03004v4>, as well as functions to fit BSS processes
to data, and functions to estimate the stochastic volatility process of a BSS process.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Imports hypergeo, MASS, phangorn

Suggests testthat

NeedsCompilation no

Author Phillip Murray [aut, cre]

Maintainer Phillip Murray <phillip.murray18@imperial.ac.uk>

Repository CRAN

Date/Publication 2020-06-24 12:10:11 UTC

R topics documented:

---

bssAlphaFit — *Estimating the smoothness parameter of a Brownian semistationary process*

Description

`bssAlphaFit` uses the 'Change of Frequency' method to estimate the smoothness parameter, `alpha`, of a BSS process. The COF method needs only minimal assumptions on the parametric form of the kernel, therefore the estimate can be used in any kernel.

Usage

```
bssAlphaFit(Y, p = 2)
```

Arguments

`Y`	a vector of observations of a BSS process at any frequency.
`p`	the power to be used in the change of frequency method. The default value is p = 2.

Value

The function returns a single value - an estimate for the smoothness parameter alpha.

Examples

```
N <- 10000
n <- 100
T <- 1.0
theta <- 0.5
beta <- 0.125

kappa <- 3
alpha <- -0.2
lambda <- 1.0


vol <- exponentiatedOrnsteinUhlenbeck(N, n, T, theta, beta)
```

```
bss_simulation <- gammaKernelBSS(N, n, T, kappa, alpha, lambda, sigma = vol)
y <- bss_simulation$bss

bssAlphaFit(y, p = 2)
```

`estimateAccumulatedVolatility` *Estimate accumulated volatility processes*

Description

`estimateAccumulatedVolatility` estimates the pth power accumulated volatility process for a Brownian semistationary process, using either parametric methods of model fitting first, or using a non-parametric estimator for the scale factor.

Usage

```
estimateAccumulatedVolatility(
  Y,
  n,
  p = 2,
  method = "nonparametric",
  kernel = "gamma"
)
```

Arguments

`Y`	a vector of observations of a BSS process.
`n`	positive integer indicating the number of observations per unit of time.
`p`	the power to evaluate the accumulated power volatility process for. Defaults to 2, in order to estimate the accumulated squared volatility process.
`method`	text string representing the method used to estimate the accumulated volatility. Options are `'acf'`, `'cof'` or `nonparametric`. If `'acf'` is selected, model parameters are fit to the data `Y` using least squares on the autocorrelation function and these parameters are used to estimate the scale factor. If `'cof'` is selected, only the smoothness parameter `alpha` is estimated using the change of frequency method, and then put into an asymptotic expression for the scale factor in the calculation. If `'nonparametric'` is selected then the non-parametric estimator for the scale factor will be used in the calculation. Defaults to `'nonparametric'`.
`kernel`	text string representing the choice of kernel when fitting the model to estimate the scale factor parametrically. Options are `'gamma'` and `'power'`. Defaults to `'gamma'`.

Value

The function returns a vector of the same length as `Y` which is the estimate for the accumulated volatility process, observed from time 0 to T, at intervals of T/n. Note that the values have been divided by m_p in the output, so that the estimation is of the integral alone. If the non-parametric estimator for tau_n is used then the values will be scaled by the expectation of the squared volatility, as per the theory.

Examples

```
N <- 10000
n <- 100
T <- 1.0
theta <- 0.5
beta <- 0.125

kappa <- 3
alpha <- -0.2
lambda <- 1.0


vol <- exponentiatedOrnsteinUhlenbeck(N, n, T, theta, beta)
bss_simulation <- gammaKernelBSS(N, n, T, kappa, alpha, lambda, sigma = vol)
y <- bss_simulation$bss
estimateAccumulatedVolatility(y, n, p = 2, method = 'nonparametric', kernel = 'gamma')

#'
```

`estimateAccumulatedVolatilityCI`
Estimate confidence interval for the accumulated volatility processes

---

Description

`estimateAccumulatedVolatility` estimates a confidence interval for the pth power accumulated volatility process for a Brownian semistationary process, using either parametric methods of model fitting first, or using a non-parametric estimator for the scale factor.

Usage

```
estimateAccumulatedVolatilityCI(
  Y,
  n,
  p,
  method = "nonparametric",
  kernel = "gamma",
  confidence_level
)
```

Arguments

`Y`	a vector of observations of a BSS process.
`n`	positive integer indicating the number of observations per unit of time.
`p`	the power to evaluate the accumulated power volatility process for. Defaults to 2, in order to estimate the accumulated squared volatility process.
`method`	text string representing the method used to estimate the accumulated volatility. Options are `'acf'`, `'cof'` or `nonparametric`. If `'acf'` is selected, model parameters are fit to the data `Y` using least squares on the autocorrelation function and these parameters are used to estimate the scale factor. If `'cof'` is selected, only the smoothness parameter `alpha` is estimated using the change of frequency method, and then put into an asymptotic expression for the scale factor in the calculation. If `'nonparametric'` is selected then the non-parametric estimator for the scale factor will be used in the calculation. Defaults to `'nonparametric'`.
`kernel`	text string representing the choice of kernel when fitting the model to estimate the scale factor parametrically. Options are `'gamma'` and `'power'`. Defaults to `'gamma'`.
`confidence_level`	the required level for the confidence interval, as a probability between 0 and 1.

Value

The function returns a list of two vectors of the same length as `Y` which are the estimates for the lower and upper values for the confidence interval. Note that the values have been divided by m_p in the output, so that the estimation is of the integral alone. If the non-parametric estimator for tau_n is used then the values will be scaled by the expectation of the squared volatility, as per the theory.

Examples

```
N <- 10000
n <- 100
T <- 1.0
theta <- 0.5
beta <- 0.125

kappa <- 3
alpha <- -0.2
lambda <- 1.0


vol <- exponentiatedOrnsteinUhlenbeck(N, n, T, theta, beta)
bss_simulation <- gammaKernelBSS(N, n, T, kappa, alpha, lambda, sigma = vol)
y <- bss_simulation$bss
estimateAccumulatedVolatility(y, n, p = 2, method = 'nonparametric', kernel = 'gamma')

#'
```

exponentiatedOrnsteinUhlenbeck *Simulate an exponentiated OU volatility process*

Description

`exponentiatedOrnsteinUhlenbeck` simulates an exponentiated Ornstein-Uhlenbeckprocess of the correct length to be used as the volatility process within the hybrid scheme.

Usage

```
exponentiatedOrnsteinUhlenbeck(N, n, T, theta, beta)
```

Arguments

`N`	positive integer determining the number of terms in the Riemman sum element of the hybrid scheme calculation. Should be of order at least `n`.
`n`	positive integer indicating the number of observations per unit of time. It represents the fineness or frequency of observations.
`T`	the time interval to simulate the BSS process over.
`theta`	positive number giving the mean reversion rate of the OU process.
`beta`	the factor in the exponential.

Value

The function returns a vector of length `N + n*T + 1`

Examples

```
N <- 10000
n <- 100
T <- 1.0
theta <- 0.5
beta <- 0.125

vol <- exponentiatedOrnsteinUhlenbeck(N, n, T, theta, beta)
```

`gammaKernelBSS` *Simulation of gamma kernel Brownian semistationary processes*

Description

`gammaKernelBSS` uses the Hybrid scheme to simulate a Brownian semistationary process from the gamma kernel. It simulates a path where the volatility process is independent of the driving Brownian motion of the BSS process.

Usage

```
gammaKernelBSS(
  N,
  n,
  T,
  kappa = 3,
  alpha,
  lambda,
  sigma = rep(1, N + n * T + 1)
)
```

Arguments

`N`	positive integer determining the number of terms in the Riemman sum element of the hybrid scheme calculation. Should be of order at least `n`.
`n`	positive integer indicating the number of observations per unit of time. It represents the fineness or frequency of observations.
`T`	the time interval to simulate the BSS process over.
`kappa`	positive integer giving the number of terms to use in the 'lower' sum of the hybrid scheme. Default set to 3.
`alpha`	the smoothness parameter of the BSS process to simulate.
`lambda`	the exponent parameter of the BSS process to simulate.
`sigma`	the volatility process used in the BSS simulation. This should be a vector of length `N + n*T + 1` representing the sample path of sigma from -N to nT. By default this is set to by a vector of 1s so that the Gaussian core is simulated.

Value

The function returns a list of three objects, `core` gives the Gaussian core of the process between 0 and T, at intervals of 1/n. `bss` gives the BSS sample path on the between 0 and T, at intervals of 1/n, and `vol` gives the volatilty process over the same time period.

N <- 10000 n <- 100 T <- 1.0 theta <- 0.5 beta <- 0.125

kappa <- 3 alpha <- -0.2 lambda <- 1.0

vol <- exponentiatedOrnsteinUhlenbeck(N, n, T, theta, beta) bss_simulation <- gammaKernelBSS(N, n, T, kappa, alpha, lambda, sigma = vol)

gammaKernelBSSFit *Fitting gamma kernel Brownian semistationary processes*

Description

gammaKernelBSSFit uses a method of moments to fit the parameters of a gamma kernel Brownian semistationary process to a vector of observations. A least squares estimate of the parameters is obtained by minimising the mean square error between the true gamma kernel autocorrelation function and the empirical ACF of the data, using lags 0,...,H. The number of lags num_lags used can be adjusted. The volatility process does not need to be specified.

Usage

```
gammaKernelBSSFit(Y, n, num_lags = 10)
```

Arguments

Y	a vector of observations of a BSS process at frequency n.
n	positive integer indicating the number of observations per unit of time.
num_lags	the number of lags to be used in the regression. The default is to use the first 10 lags.

Value

The function returns a list containing the parameters alpha and lambda, and also the mean square error mse of the least squares fit. This can be used to compare model fit when trying different kernels.

Examples

```
N <- 10000
n <- 100
T <- 1.0
theta <- 0.5
beta <- 0.125

kappa <- 3
alpha <- -0.2
lambda <- 1.0


vol <- exponentiatedOrnsteinUhlenbeck(N, n, T, theta, beta)
bss_simulation <- gammaKernelBSS(N, n, T, kappa, alpha, lambda, sigma = vol)
y <- bss_simulation$bss

gammaKernelBSSFit(y, n, num_lags = 10)
```

gammaKernelCorrelation *Autocorrelation function for the gamma kernel*

Description

`gammaKernelCorrelation` calculates the value of the gamma kernel autocorrelation function directly using the analytic expression.

Usage

```
gammaKernelCorrelation(alpha, lambda, h)
```

Arguments

`alpha`	the smoothness parameter, alpha, for the gamma kernel.
`lambda`	the exponent parameter, lambda, for the gamma kernel.
`h`	the lag to calculate the autocorrelation at.

Value

The function returns the autocorrelation for the gamma kernel with parameters `alpha` and `lambda` at lag `h`.

gammaKernelTau *Scale factor for the gamma kernel*

Description

`gammaKernelTau` evaluates the scale factor tau_n for the gamma kernel using the exact expression derived from the covariance function.

Usage

```
gammaKernelTau(n, alpha, lambda)
```

Arguments

`n`	a positive integer indicating the number of observations per unit of time.
`alpha`	the smoothness parameter, alpha, for the gamma kernel.
`lambda`	the exponent parameter, lambda, for the gamma kernel.

Value

The function returns the scale factor (tau_n) for the gamma kernel with parameters `alpha` and `lambda`, observed at frequency `n` per unit of time.

`gammaKernelTauAsymptotic` *Asymptotic scale factor for the gamma kernel*

Description

Asymptotic scale factor for the gamma kernel

Usage

```
gammaKernelTauAsymptotic(n, alpha)
```

Arguments

`n`	a positive integer indicating the number of observations per unit of time.
`alpha`	the smoothness parameter, alpha, for the gamma kernel.

Value

The function returns an approximation for the scale factor (tau_n) for the gamma kernel with smoothness parameter `alpha`, observed at frequency `n` per unit of time, using the asymptotic expression for the scale factor.

`hybridSchemeCovarianceMatrix` *Hybrid scheme covariance matrix*

Description

Generates the covariance matrix used in simulating Brownian semistationary processes by the hybrid scheme.

Usage

```
hybridSchemeCovarianceMatrix(kappa, n, alpha)
```

Arguments

`kappa`	number of terms needed for the lower sum in the hybrid scheme.
`n`	number of observations per unit of time, n = 1/delta.
`alpha`	smoothness parameter used in the BSS simulation.

Value

Returns the covariance matrix for the lower sum in the hybrid scheme calculations. The dimensions of the covariance matrix will be (kappa + 1) by (kappa + 1).

Examples

```
kappa <- 3
n <- 100
alpha <- -0.2

hybridSchemeCovarianceMatrix(kappa, n, alpha)
```

`powerKernelBSS` *Simulation of power law kernel Brownian semistationary processes*

Description

`powerKernelBSS` uses the Hybrid scheme to simulate a Brownian semistationary process from the power law kernel. It simulates a path where the volatility process is independent of the driving Brownian motion of the BSS process.

Usage

```
powerKernelBSS(N, n, T, kappa, alpha, beta, sigma = rep(1, N + n * T + 1))
```

Arguments

`N`	positive integer determining the number of terms in the Riemman sum element of the hybrid scheme calculation. Should be of order at least `n`.
`n`	positive integer indicating the number of observations per unit of time. It represents the fineness or frequency of observations.
`T`	the time interval to simulate the BSS process over.
`kappa`	positive integer giving the number of terms to use in the 'lower' sum of the hybrid scheme. Default set to 3.
`alpha`	the smoothness parameter of the BSS process to simulate.
`beta`	the exponent parameter of the BSS process to simulate.
`sigma`	the volatility process used in the BSS simulation. This should be a vector of length `N + n*T + 1` representing the sample path of sigma from -N to nT. By default this is set to by a vector of 1s so that the Gaussian core is simulated.

Value

The function returns a list of three objects, `core` gives the Gaussian core of the process between 0 and T, at intervals of 1/n. `bss` gives the BSS sample path on the between 0 and T, at intervals of 1/n, and `vol` gives the volatilty process over the same time period.

Examples

```
N <- 10000
n <- 100
T <- 1.0
theta <- 0.5
beta_vol <- 0.125

kappa <- 3
alpha <- -0.2
beta_pwr <- -1.0


vol <- exponentiatedOrnsteinUhlenbeck(N, n, T, theta, beta_vol)
bss_simulation <- powerKernelBSS(N, n, T, kappa, alpha, beta_pwr, sigma = vol)
```

`powerKernelBSSFit` *Fitting power law kernel Brownian semistationary processes*

Description

`powerKernelBSSFit` uses a method of moments to fit the parameters of a power law kernel Brownian semistationary process to a vector of observations. A least squares estimate of the parameters is obtained by minimising the mean square error between the true power law kernel autocorrelation function (found by numerical intergration) and the empirical ACF of the data, using lags 0,...,H. The number of lags `num_lags` used can be adjusted. The volatility process does not need to be specified.

Usage

```
powerKernelBSSFit(Y, n, num_lags = 10)
```

Arguments

`Y`	a vector of observations of a BSS process at frequency `n`.
`n`	positive integer indicating the number of observations per unit of time.
`num_lags`	the number of lags to be used in the regression. The default is to use the first 10 lags.

Value

The function returns a list containing the parameters `alpha` and `beta`, and also the mean square error `mse` of the least squares fit. This can be used to compare model fit when trying different kernels.

Examples

```
N <- 10000
n <- 100
T <- 1.0
theta <- 0.5
beta_vol <- 0.125

kappa <- 3
alpha <- -0.2
beta_pwr <- -1.0


vol <- exponentiatedOrnsteinUhlenbeck(N, n, T, theta, beta_vol)
bss_simulation <- gammaKernelBSS(N, n, T, kappa, alpha, beta_pwr, sigma = vol)
y <- bss_simulation$bss

powerKernelBSSFit(y, n, num_lags = 10)
```

---

powerKernelCorrelation
Autocorrelation function for the power law kernel

Description

powerKernelCorrelation calculates the value of the power law kernel autocorrelation function directly using numerical integration for the numerator (the covariance term) and the analytic expression for the denominator (variance term).

Usage

```
powerKernelCorrelation(alpha, beta, h)
```

Arguments

alpha	the smoothness parameter, alpha, for the power law kernel.
beta	the exponent parameter, beta, for the power law kernel.
h	the lag to calculate the autocorrelation at.

Value

The function returns the autocorrelation for the power law kernel with parameters alpha and beta at lag h.

powerKernelTau *Scale factor for the power law kernel*

Description

`powerKernelTau` evaluates the scale factor tau_n for the power law kernel using numerical integration for the covariance term, and exact evaluation for the variance term.

Usage

```
powerKernelTau(n, alpha, beta)
```

Arguments

`n`	a positive integer indicating the number of observations per unit of time.
`alpha`	the smoothness parameter, alpha, for the power law kernel.
`beta`	the exponent parameter, beta, for the power law kernel.

Value

The function returns the scale factor (tau_n) for the power law kernel with parameters `alpha` and `beta`, observed at frequency `n` per unit of time.

realisedPowerVariation *Realised power variation*

Description

`realisedPowerVariation` calculates the realised power variation for a BSS process, which is then used as a component of estimating the accumulated volatility process.

Usage

```
realisedPowerVariation(Y, p)
```

Arguments

`Y`	a vector of observations of a BSS process.
`p`	the power variation to be calculated.

Value

The function returns the sum of the pth powers of the absolute first differences of the BSS process.

`tauNonParametricEstimate` *Non-parametric estimate of the scale factor*

Description

Non-parametric estimate of the scale factor

Usage

```
tauNonParametricEstimate(Y)
```

Arguments

`Y`	a vector of observations of a BSS process.

Value

The function returns the non-parametric estimate for the scale factor. Note that this will be scaled by the expectation of the square of the volatitity.

Index