Package ‘quantspec’

March 26, 2015

Version 1.0-3
Encoding UTF-8
Date 2015-03-25
Title Quantile-Based Spectral Analysis of Time Series
Depends R (>= 3.0.0), stats4
Suggests testthat
Imports methods, graphics, quantreg, abind, zoo, snowfall, Rcpp (>= 0.11.0)
Description Methods to determine, smooth and plot quantile (i.e., Laplace or copula) periodograms for univariate time series.
License GPL (>= 2)
URL http://github.com/tobiaskley/quantspec
BugReports http://github.com/tobiaskley/quantspec/issues
LazyData TRUE
LinkingTo Rcpp
Collate 'Class-BootPos.R' 'generics.R' 'aux-functions.R'
 'Class-QSpecQuantity.R' 'Class-FreqRep.R' 'Class-ClippedFT.R'
 'Class-QuantileSD.R' 'Class-IntegrQuantileSD.R'
 'Class-Weight.R' 'Class-KernelWeight.R' 'Class-MovingBlocks.R'
 'Class-QRegEstimator.R' 'Class-QuantilePG.R'
 'Class-SmoothedPG.R' 'Class-SpecDistrWeight.R' 'RcppExports.R'
 'data.R' 'deprecated.R' 'kernels.R' 'models.R'
 'quantspec-package.R'

Author Tobias Kley [aut, cre]
Maintainer Tobias Kley <tobias.kley@ruhr-uni-bochum.de>
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-03-26 17:18:14
R topics documented:

- quantspec-package ........................................... 3
- BootPos-class .................................................. 7
- ClippedFT-class ................................................ 8
- ClippedFT-constructor ........................................ 8
- closest.pos .................................................... 9
- data-sp500 ..................................................... 10
- data-wheatprices ............................................. 11
- FreqRep-class .................................................. 11
- frequenciesValidator ......................................... 13
- generics-accessors ............................................ 14
- generics-associations ........................................ 15
- generics-functions ........................................... 16
- getB-FreqRep ................................................... 17
- getBootPos-FreqRep ........................................... 17
- getBw-KernelWeight ........................................... 18
- getDescr-Weight ............................................... 18
- getFreqRep-QuantilePG ....................................... 19
- getFrequencies-FreqRep ...................................... 19
- getFrequencies-QSpecQuantity ............................... 20
- getIsRankBased-FreqRep ...................................... 20
- getLevels-FreqRep ............................................. 21
- getLevels-QSpecQuantity ..................................... 21
- getMeanPG-QuantileSD ......................................... 22
- getN-QuantileSD ................................................ 22
- getParallel-QRegEstimator ................................... 23
- getPositions-MovingBlocks ................................... 23
- getQuantilePG-QuantileSD ................................... 24
- getQuantilePG-SmoothedPG ................................... 25
- getQuantileSD-IntegrQuantileSD ............................ 26
- getQuantileSD-IntegrQuantileSD ......................... 26
- getR-QuantileSD ............................................... 27
- getSdBoot-SmoothedPG ........................................ 27
- getSdNaive-SmoothedPG ....................................... 28
- getStdError-QuantileSD ....................................... 29
- getTs-QuantileSD .............................................. 30
- getType-QuantileSD .......................................... 30
- getValues-FreqRep ............................................. 31
- getValues-IntegrQuantileSD .................................. 32
- getValues-KernelWeight ...................................... 32
- getValues-QuantilePG ......................................... 33
- getValues-QuantileSD ........................................ 34
- getValues-SmoothedPG ........................................ 35
- getValues-SpecDistrWeight .................................... 36
- getW-KernelWeight ............................................ 36
- getWeight-SmoothedPG ........................................ 37
- getWnj-KernelWeight .......................................... 37
Description

Methods to determine, smooth and plot quantile (i.e., Laplace or copula) periodograms for univariate time series.
Contents

The `quantspec` package contains an hierarchy of S4 classes with corresponding methods and functions serving as constructors. The following class diagrams provide an overview on the structure of the package. In the first class diagram the classes implementing the estimators are implemented.

In the second class diagram the classes implementing model quantities are displayed. A relation to the “empirical classes” is given via the fact that the quantile spectral densities are computed by simulation of quantile periodograms and a common abstract superclass `QSpecQuantity` which is used to provide a common interface to quantile spectral quantities.
Besides the object-oriented design a few auxiliary functions exist. They serve as parameters or are mostly for internal use. A more detailed description of the framework can be found in the paper on the package (Kley, 2015).

**Organization of the source code / files in the /R folder**

All of the source code related to the specification of a certain class is contained in a file named `Class-[Name_of_the_class].R`. This includes, in the following order,

1. all roxygen `@include` to insure the correctly generated collate for the DESCRIPTION file.
2. `@setClass` preceded by a meaningful roxygen documentation.
3. specification of an initialize method, where appropriate.
4. all accessor and mutator method (i.e., getter and setter); first the ones returning attributes of the object, then the ones returning associated objects.
5. constructors; use generics if there is more than one of them.
6. `show` and `plot` methods.

**Coding Conventions**

To improve readability of the software and documentation this package was written in the spirit of the “Coding conventions of the Java Programming Language” (Oracle, 2015). In particular, the naming conventions for classes and methods have been adopted, where “Class names should be nouns, in mixed case with the first letter of each internal word capitalized.” and “Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.”

**Naming Conventions for the Documentation**

To reflect the structure of the contents of the package in the documentation file, the following system for naming of the sections is adopted:

- Documentation of an S4 class is named as the name of the class followed by “-class”. [cf. `QuantilePG-class`]
• Documentation of a constructor for an S4-class is named as the name of the class followed by “-constructor”. [cf. Quan$\text{tilePG}$-constructor]

• Documentation of a method dispatching to an object of a certain S4 class is named by the name of the method, followed by “-”, followed by the name of the Class. [cf. getValues-Quan$\text{tilePG}$]

Author(s)

Tobias Kley

References


---

BootPos-class

Class for Generation of Bootstrapped Replications of a Time Series.

Description

BootPos is an S4 class that provides a common interface to different algorithms that can be used for implementation of a block bootstrap procedure in the time domain.

Details

After initialization the bootstrapping can be performed by applying getPositions to the object. Different block bootstraps are implemented by creating a subclass together with a getPositions method that contains the implementation of the block resampling procedure.

Currently the following implementations are available:

• MovingBlocks and getPositions-MovingBlocks.

Slots

1 the (expected) block length for the block bootstrap methods
N number of available observations to bootstrap from
References

**ClippedFT-class**  
*Class for Fourier transform of the clipped time series.*

**Description**
ClippedFT is an S4 class that implements the necessary calculations to determine the Fourier transform of the clipped time series. As a subclass to `freqrep` it inherits slots and methods defined there; it serves as a frequency representation of a time series as described in Kley et. al (2015+).

**Details**
For each frequency $\omega$ from frequencies and level $q$ from levels the statistic

$$
\sum_{t=0}^{n-1} I\{Y_t \leq q\} e^{-i\omega t}
$$

is determined and stored to the array `values`. Internally the methods `mvfft` and `fft` are used to achieve good performance.

Note that, all remarks made in the documentation of the super-class `FreqRep` apply.

**References**

**See Also**
For an example see `FreqRep`.

---

**ClippedFT-constructor**  
*Create an instance of the ClippedFT class.*

**Description**
The parameter `type.boot` can be set to choose a block bootstrapping procedure. If "none" is chosen, a moving blocks bootstrap with $l=length(Y)$ and $N=length(Y) would be done. Note that in that case one would also chose $B=0$ which means that `getPositions` would never be called. If $B>0$ then each bootstrap replication would be the undisturbed time series.
Usage

closest.pos(x, y)

Arguments

x Vector of elements among which to find the closest one for each element in y.

y Vector of elements for which to find the closest element in x.
Value

Returns a vector of same length as \( X \), with indices indicating which element in \( Y \) is closest.

Examples

\[
X_1 \leftarrow c(1,2,3)
\]
\[
\text{closest.pos}(X_1, 1.7)
\]
\[
\text{closest.pos}(X_1, c(1.3,2.2))
\]

\[
X_2 \leftarrow c(2,1,3)
\]
\[
\text{closest.pos}(X_2, 1.5)
\]

---

**data-sp500**  

*S&P 500: Standard and Poor’s 500 stock index, 2007–2010*

---

Description

Contains the returns of the S&P 500 stock index for the years 2007–2010. The returns were computed as \((\text{Adjusted.Close}/\text{Open})/\text{Open}\).

Format

A univariate time series with 1008 observations; a zoo object

Details

The data was downloaded from the Yahoo! Finance Website.

References


Examples

plot(sp500)
Beveridge’s Wheat Price Index (detrended and demeaned), 1500–1869

Description

Contains a detrended and demeaned version of the well-known Beveridge Wheat Price Index which
gives annual price data from 1500 to 1869, averaged over many locations in western and central
Europe [cf. Beveridge (1921)]. The index series \( x_t \) was detrended as proposed by Granger (1964),
p. 21, by letting

\[
y_t := \frac{x_t}{\sum_{j=-15}^{15} x_{t+j}},
\]

where \( x_t := x_{1}, t < 1 \) and \( x_t := x_{n}, t > n \). The time series in the data set is also demeaned by
letting

\[
\hat{y}_t := y_t - \frac{1}{n} \sum_{t=1}^{n} y_t.
\]

Format

A univariate time series \((\hat{y}_t)\) with 370 observations; a ts object.

Details

The index data cited in Beveridge’s paper was taken from \( \text{bev} \) in the \text{tseries} package.

References

Princeton, NJ.

Examples

\[
\text{plot(} \text{wheatprices) \text{)}
\]

FreqRep-class

\textit{Class for Frequency Representation.}

Description

FreqRep is an S4 class that encapsulates, for a time series \((Y_t)_{t=0, \ldots, n-1}\), the data structures for the
storage of a frequency representation. Examples of such frequency representations include

- the Fourier transformation of the clipped time series \((I\{Y_t \leq q\})\), or
- the weighted \(L_1\)-projection of \((Y_t)\) onto an harmonic basis.

Examples are realized by implementing a sub-class to FreqRep. Currently, implementations for the
two examples mentioned above are available: \textit{ClippedFT} and \textit{QRegEstimator}. 
Details

It is always an option to base the calculations on the pseudo data \( R_{t,n}/n \) where \( R_{t,n} \) denotes the rank of \( Y_t \) among \((Y_t)_{t=0,\ldots,n-1}\).

To allow for a block bootstrapping procedure a number of \( B \) estimates determined from bootstrap replications of the time series which are yield by use of a \texttt{BootPos}-object can be stored on initialization.

The data in the frequency domain is stored in the array \texttt{values}, which has dimensions \((J,K,B+1)\), where \( J \) is the number of frequencies, \( K \) is the number of levels and \( B \) is the number of bootstrap replications requested on initialzation. In particular, \texttt{values}[,\( j,k,1 \)] corresponds to the time series’ frequency representation with \texttt{frequencies}[,\( j \)] and \texttt{levels}[,\( k \)], while \texttt{values}[,\( j,k,b+1 \)] is the for the same, but determined from the \( b \)th block bootstrapped replicate of the time series.

Slots

- \( Y \) The time series of which the frequency representation is to be determined.
- \texttt{frequencies} The frequencies for which the frequency representation will be determined. On initialization \texttt{frequenciesValidator} is called, so that it will always be a vector of reals from \([0,\pi]\). Also, only Fourier frequencies of the form \( 2\pi j/n \) with integers \( j \) and \( n \) the \texttt{length(Y)} are allowed.
- \texttt{levels} The levels for which the frequency representation will be determined. If the flag \texttt{isRankBased} is set to \texttt{FALSE}, then it can be any vector of reals. If \texttt{isRankBased} is set to \texttt{TRUE}, then it has to be from \([0,1]\).
- \texttt{values} The array holding the determined frequency representation. Use a \texttt{getValues} method of the relevant subclass to access it.
- \texttt{isRankBased} A flag that is \texttt{FALSE} if the determined \texttt{values} are based on the original time series and \texttt{TRUE} if it is based on the pseudo data as described in the Details section of this topic.
- \texttt{positions.boot} An object of type \texttt{BootPos}, that is used to determine the block bootstrapped replicates of the time series.
- \( B \) Number of bootstrap replications to perform.

Examples

```r
Y <- rnorm(32)
freq <- 2*pi*c(0:31)/32
levels <- c(0.25,0.5,0.75)
cFT <- clippedFT(Y, freq, levels)
plot(cFT)

# Get values for all Fourier frequencies and all levels available.
V.all <- getValues(cFT)

# Get values for every second frequency available
V.coarse <- getValues(cFT, frequencies = 2*pi*c(0:15)/16, levels = levels)

# Trying to get values on a finer grid of frequencies than available will
# yield a warning and then all values with frequencies closest to that finer
frequenciesValidator

Validates if frequencies are Fourier frequencies from $[0, \pi]$.

Description

Validation of the parameter `freq` is performed in six steps:

1. Throw an error if parameter is not a vector or not numeric.
2. Transform each element $\omega$ of the vector to $[0, 2\pi)$, by replacing it with $\omega \mod 2\pi$.
3. Check whether all elements $\omega$ of the vector are Fourier frequency $2\pi j/T$, $j \in \mathbb{Z}$. If this is not the case issue a warning and round each frequency to the next Fourier frequency of the mentioned type; the smaller one, if there are two.
4. Transform each element $\omega$ with $\pi < \omega < 2\pi$ of the vector to $[0, \pi]$, by replacing it with $2\pi - \omega$.
5. Check for doubles and remove all but the first appearance.
6. Sort in ascending order.

Any subset of the six steps can be chosen, but 1 should almost always be among the steps to be performed.

Usage

`frequenciesValidator(freq, N, steps = 1:6)`

Arguments

- `freq` the vector of frequencies to be validated.
- `N` the base of the Fourier frequencies against which the values in `freq` will be compared.
- `steps` a vector containing a subset of 1,2,3,4,5,6, indicating which of the steps are to be performed.

Value

Returns a vector of Fourier frequencies that is yield by the transformations described above.
Examples

freq <- 2*pi*c(3,2,5,8,9)/10
res <- frequenciesValidator(freq, N=10, steps=1:3)
res * 10 / (2*pi) # Returns: [1] 3 2 5 8 9

res <- frequenciesValidator(freq, N=10, steps=1:4)
res * 10 / (2*pi) # Returns: [1] 3 2 5 2 1

res <- frequenciesValidator(freq, N=10, steps=1:5)
res * 10 / (2*pi) # Returns: [1] 3 2 5 1

res <- frequenciesValidator(freq, N=10, steps=1:6)
res * 10 / (2*pi) # Returns: [1] 1 2 3 5

Description

These generic functions are needed to access the objects’ attributes. Note that the naming convention getAttribute was applied, where attribute is the name of the attribute/slot of the class of the object.

Usage

getY(object, ...)
getValues(object, ...)
getIsRankBased(object, ...)
getB(object, ...)
getParallel(object, ...)
getFrequencies(object, ...)
getLevels(object, ...)
getMeanPG(object, ...)
getError(object, ...)
getN(object, ...)
getR(object, ...)
getType(object, ...)  
getTs(object, ...)  
getSdNaive(object, ...)  
getSdBoot(object, ...)  
getPointwiseCIs(object, ...)  
getDescr(object, ...)  
getW(object, ...)  
getBw(object, ...)  
getWnj(object, ...)  

Arguments

object  object from which to get the value
...  optional parameters; for documentation see the documentation of the methods
to each of the generic.

See Also

For an overview on the classes of the framework, and all of their attributes, see the class diagrams
in the package description [cf. quantspec-package].
getQuantileSD(object, ...)

getWeight(object, ...)

**Arguments**

object  
object from which to get the associated object

...  
optional parameters; for documentation see the documentation of the methods to each of the generic.

**See Also**

For an overview on the classes of the framework, and all associations, see the class diagrams in the package description [cf. quantspec-package].

generics-functions  
Generic functions for implementation of methods of a class

description

These generic functions need to be defined to allow for the automatic dispatching mechanism.

**Usage**

increasePrecision(object, ...)

getPositions(object, ...)

**Arguments**

object  
specifies the object from which the method is to be applied.

...  
optional parameters; for documentation see the documentation of the methods to the generic.

**See Also**

For an overview on the classes of the framework, and all of their methods, see the class diagrams in the package description [cf. quantspec-package].
**getB-FreqRep**

Get B from a *FreqRep* object.

**Description**

Get B from a *FreqRep* object.

**Usage**

```r
## S4 method for signature 'FreqRep'
getB(object)
```

**Arguments**

- `object` : *FreqRep* of which to get the B

**Value**

Returns the attribute B that’s a slot of object.

---

**getBootPos-FreqRep**

Get associated *BootPos* from a *FreqRep*.

**Description**

Get associated *BootPos* from a *FreqRep*.

**Usage**

```r
## S4 method for signature 'FreqRep'
getBootPos(object)
```

**Arguments**

- `object` : *FreqRep* from which to get the *BootPos*.

**Value**

Returns the *BootPos* object associated.
**getBw-KernelWeight**  
*Get attribute bw (bandwidth / scaling parameter used for smoothing) from a KernelWeight.*

---

**Description**  
Get attribute bw (bandwidth / scaling parameter used for smoothing) from a KernelWeight.

**Usage**  
```r
## S4 method for signature 'KernelWeight'
getBw(object)
```

**Arguments**  
- `object` : KernelWeight from which to get the bandwidth bw.

**Value**  
Returns the bw attribute.

---

**getDescr-Weight**  
*Get attribute descr from a Weight.*

---

**Description**  
Get attribute descr from a Weight.

**Usage**  
```r
## S4 method for signature 'Weight'
getDescr(object)
```

**Arguments**  
- `object` : Weight from which to get the descr.

**Value**  
Returns the descr attribute.
getFreqRep-QuantilePG

Get associated `FreqRep` from a `QuantilePG`.

Description

Get associated `FreqRep` from a `QuantilePG`.

Usage

```r
## S4 method for signature 'QuantilePG'
getFreqRep(object)
```

Arguments

- `object`: QuantilePG from which to get the `FreqRep`.

Value

Returns the `FreqRep` object associated.

getFrequencies-FreqRep

Get attribute frequencies from a `FreqRep`.

Description

Get attribute frequencies from a `FreqRep`.

Usage

```r
## S4 method for signature 'FreqRep'
getFrequencies(object)
```

Arguments

- `object`: `FreqRep` from which to get the frequencies.

Value

Returns the `frequencies` attribute, as a vector of real numbers.
getFrequencies-QSpecQuantity

*Get attribute* frequencies from a `QSpecQuantity`.

Description

Get attribute frequencies from a `QSpecQuantity`.

Usage

```r
## S4 method for signature 'QSpecQuantity'
getFrequencies(object)
```

Arguments

- `object`: `QSpecQuantity` from which to get the frequencies.

Value

Returns the frequencies attribute, as a vector of real numbers.

Examples

```r
qPG <- quantilePG(rnorm(10), levels=1c(0.25, 0.5))
freq <- getFrequencies(qPG)
```

getIsRankBased-FreqRep

*Get isRankBased from a `FreqRep` object*

Description

Get isRankBased from a `FreqRep` object.

Usage

```r
## S4 method for signature 'FreqRep'
getIsRankBased(object)
```

Arguments

- `object`: `FreqRep` of which to get the `isRankBased`

Value

Returns the attribute `isRankBased` that’s a slot of `object`.  

```r
```
**getLevels-FreqRep**  
*Get attribute levels from a FreqRep.*

---

**Description**

Get attribute levels from a FreqRep.

**Usage**

```r
## S4 method for signature 'FreqRep'
getLevels(object)
```

**Arguments**

- `object` FreqRep from which to get the levels.

**Value**

Returns the levels attribute, as a vector of real numbers.

---

**getLevels-QSpecQuantity**  
*Get attribute levels from a QSpecQuantity.*

---

**Description**

If the optional parameter `j` is supplied, then the `j`th vector of levels will be returned, a list with all vectors otherwise.

**Usage**

```r
## S4 method for signature 'QSpecQuantity'
getLevels(object, j)
```

**Arguments**

- `object` QSpecQuantity from which to get the levels.
- `j` Index pointing to a set of levels in the list; optional.

**Value**

Returns levels attribute, as a vector of real numbers.

**Examples**

```r
qPG <- quantilePG(rnorm(10), levels.1=c(0.25,0.5))
levels.list <- getLevels(qPG)
levels.1 <- getLevels(qPG,1)
```
### getMeanPG-QuantileSD

*Get meanPG from a quantile spectral density kernel*

**Description**

The selection mechanism for frequencies and levels operates in the same way as described in `getValues-QuantileSD`. The format of the output is also described there.

**Usage**

```r
## S4 method for signature 'QuantileSD'
getMeanPG(object, frequencies = 2 * pi *
    (0:(getN(object) - 1))/getN(object), levels.1 = getLevels(object, 1),
    levels.2 = getLevels(object, 2))
```

**Arguments**

- `object`: QuantileSD of which to get the meanPG
- `frequencies`: a vector of frequencies for which to get the meanPG
- `levels.1`: the first vector of levels for which to get the meanPG
- `levels.2`: the second vector of levels for which to get the meanPG

**Value**

Returns the array meanPG that's a slot of object.

### getN-QuantileSD

*Get N from a quantile spectral density kernel*

**Description**

Get N from a quantile spectral density kernel

**Usage**

```r
## S4 method for signature 'QuantileSD'
getN(object)
```

**Arguments**

- `object`: QuantileSD of which to get the N

**Value**

Returns the attribute N that's a slot of object.
getParallel-QRegEstimator

Get getParallel from a QRegEstimator object

Description

Get getParallel from a QRegEstimator object

Usage

```r
## S4 method for signature 'QRegEstimator'
getParallel(object)
```

Arguments

- `object`: QRegEstimator of which to get the parallel

Value

Returns the attribute parallel that's a slot of object.

getPointwiseCIs-SmoothedPG

Get pointwise confidence intervals for the quantile spectral density kernel

Description

Returns a list of two arrays lowerCIs and upperCIs that contain the upper and lower limits for a level 1-\(\alpha\) confidence interval of the copula spectral density kernel. Each array is of dimension \([J,K1,K2]\), where \(J=\text{length(frequencies)}\), \(K1=\text{length(levels.1)}\), and \(K2=\text{length(levels.2)}\). At position \((j,k1,k2)\) the real (imaginary) part of the returned values are the bounds of the confidence interval for the real (imaginary) part of the quantile spectrum, which corresponds to frequencies\([j]\), levels.1\([k1]\) and levels.2\([k2]\) closest to the Fourier frequencies, levels.1 and levels.2 available in object; \code{closest.pos} is used to determine what closest to means.

Usage

```r
## S4 method for signature 'SmoothedPG'
getPointwiseCIs(object, frequencies = 2 * pi *
    (0:(length(object@qPG@freqRep@Y) - 1))/length(object@qPG@freqRep@Y),
    levels.1 = getLevels(object, 1), levels.2 = getLevels(object, 2),
    alpha = 0.1, type = c("naive.sd", "boot.sd", "boot.full"))
```
Arguments

- **object**: SmoothedPG of which to get the confidence intervals
- **frequencies**: a vector of frequencies for which to get the result
- **levels.1**: the first vector of levels for which to get the result
- **levels.2**: the second vector of levels for which to get the result
- **alpha**: the level of the confidence interval; must be from \((0, 1)\)
- **type**: a flag indicating which type of confidence interval should be returned; can take one of the three values discussed above.

Details

Currently, three different types of confidence intervals are available:

- "naive.sd": confidence intervals based on the asymptotic normality of the smoothed quantile periodogram; standard deviations are estimated using `getSdNaive`.
- "boot.sd": confidence intervals based on the asymptotic normality of the smoothed quantile periodogram; standard deviations are estimated using `getSdBoot`.
- "boot.full": confidence intervals determined by estimating the quantiles of the distribution of the smoothed quantile periodogram, by the empirical quantiles of the sample of bootstrapped replications.

Value

Returns a named list of two arrays `lowerCIS` and `upperCIS` containing the lower and upper bounds for the confidence intervals.

Examples

```r
sPG <- smoothedPG(rnorm(2^10), levels.1=0.5)
CI.upper <- Re(getPointwiseCIs(sPG)$upperCIS[,1,1])
CI.lower <- Re(getPointwiseCIs(sPG)$lowerCIS[,1,1])
freq = 2*pi*(0:1023)/1024
plot(x = freq, y = rep(0.25/(2*pi),1024),
    ylim=c(min(CI.lower), max(CI.upper)),
    type="l", col="red") # true spectrum
lines(x = freq, y = CI.upper)
lines(x = freq, y = CI.lower)
```

getPositions-MovingBlocks

Get Positions for the Moving Blocks Bootstrap.

Description

Get Positions for the Moving Blocks Bootstrap.
Usage

```r
## S4 method for signature 'MovingBlocks'
getPositions(object, B = 1)
```

Arguments

- `object`: a MovingBlocks object; used to specify the parameters `N, l` and the type of the bootstrap.
- `B`: Number of independent repetitions to bootstrap.

Value

A matrix of dimension `[N, B]` where each column gives the positions in which to reorder the observations to yield one bootstrap replication.

---

### getQuantilePG-QuantileSD

Get associated QuantilePG from a QuantileSD.

---

Description

Get associated `QuantilePG` from a `QuantileSD`.

Usage

```r
## S4 method for signature 'QuantileSD'
getQuantilePG(object)
```

Arguments

- `object`: QuantileSD from which to get the `QuantilePG`.

Value

Returns the `QuantilePG` object associated.
**getQuantilePG-SmoothedPG**

*Get associated QuantilePG from a SmoothedPG.*

**Description**

Get associated QuantilePG from a SmoothedPG.

**Usage**

```r
## S4 method for signature 'SmoothedPG'
getQuantilePG(object)
```

**Arguments**

- `object` SmoothedPG from which to get the QuantilePG.

**Value**

Returns the QuantilePG object associated.

---

**getQuantileSD-IntegrQuantileSD**

*Get associated getQuantileSD from an IntegrQuantileSD.*

**Description**

Get associated getQuantileSD from an IntegrQuantileSD.

**Usage**

```r
## S4 method for signature 'IntegrQuantileSD'
getQuantileSD(object)
```

**Arguments**

- `object` IntegrQuantileSD from which to get the getQuantileSD.

**Value**

Returns the getQuantileSD object associated.
getR-QuantileSD

---

**getR-QuantileSD**

*Get R from a quantile spectral density kernel*

**Description**

Get R from a quantile spectral density kernel

**Usage**

```r
## S4 method for signature 'QuantileSD'
getR(object)
```

**Arguments**

- `object` QuantileSD of which to get the R

**Value**

Returns the attribute R that’s a slot of object.

---

getSdBoot-SmoothedPG

---

**getSdBoot-SmoothedPG**

*Get bootstrap estimates for the standard deviation of the smoothed quantile periodogram.*

**Description**

Determines and returns an array of dimension \([J,K1,K2]\), where \(J=\text{length(frequencies)}\), \(K1=\text{length(levels.1)}\), and \(K2=\text{length(levels.2)}\). At position \((j,k1,k2)\) the real part of the returned value is the standard deviation estimated from the real parts of the bootstrap replications and the imaginary part of the returned value is the standard deviation estimated from the imaginary part of the bootstrap replications. The estimate is determined from those bootstrap replicates of the estimator that have frequencies\([j]\), levels.1\([k1]\) and levels.2\([k2]\) closest to the frequencies, levels.1 and levels.2 available in object; `closest.pos` is used to determine what closest to means.

**Usage**

```r
## S4 method for signature 'SmoothedPG'
getSdBoot(object, frequencies = 2 * pi *
(0:(length(object$qPG@freqRep@Y) - 1))/length(object$qPG@freqRep@Y),
levels.1 = getLevels(object, 1), levels.2 = getLevels(object, 2))
```

**Arguments**

- `object` `SmoothedPG` of which to get the bootstrap estimates for the standard deviation.
- `frequencies` a vector of frequencies for which to get the result
- `levels.1` the first vector of levels for which to get the result
- `levels.2` the second vector of levels for which to get the result
Details

Requires that the SmoothedPG is available at all Fourier frequencies from \((0, \pi]\). If this is not the case the missing values are imputed by taking one that is available and has a frequency that is closest to the missing Fourier frequency; closest.\pos is used to determine which one this is.

If there are no bootstrap replicates available (i.e., \(B = 0\)) an error is returned.

Note the “standard deviation” estimated here is not the square root of the complex-valued variance. It’s real part is the square root of the variance of the real part of the estimator and the imaginary part is the square root of the imaginary part of the variance of the estimator.

Value

Returns the estimate described above.

---

**getSdNaive-SmoothedPG**

Get estimates for the standard deviation of the smoothed quantile periodogram.

---

Description

Determines and returns an array of dimension \([J, K1, K2]\), where \(J = \text{length}(\text{frequencies})\), \(K1 = \text{length}(\text{levels.1})\), and \(K2 = \text{length}(\text{levels.2})\). Whether available or not, bootstrap repetitions are ignored by this procedure. At position \((j, k1, k2)\) the returned value is the standard deviation estimated corresponding to \(\text{frequencies}[j]\), \(\text{levels.1}[k1]\) and \(\text{levels.2}[k2]\) that are closest to the \(\text{frequencies}\), \(\text{levels.1}\) and \(\text{levels.2}\) available in \(\text{object}\); closest.\pos is used to determine what closest to means.

Usage

```r
## S4 method for signature 'SmoothedPG'
getSdNaive(object, frequencies = 2 * pi *
  (0:(length(object@qPG@freqRep@Y) - 1))/length(object@qPG@freqRep@Y),
  levels.1 = getLevels(object, 1), levels.2 = getLevels(object, 2),
  impl = c("R", "C"))
```

Arguments

- **object** `SmoothedPG` of which to get the estimates for the standard deviation.
- **frequencies** a vector of frequencies for which to get the result
- **levels.1** the first vector of levels for which to get the result
- **levels.2** the second vector of levels for which to get the result
- **impl** choose "R" or "C" for one of the two implementations available
**getStdError-QuantileSD**

**Details**

Requires that the SmoothedPG is available at all Fourier frequencies from \((0, \pi]\). If this is not the case the missing values are imputed by taking one that is available and has a frequency that is closest to the missing Fourier frequency; closest.pos is used to determine which one this is.

A precise definition on how the standard deviations of the smoothed quantile periodogram are estimated is given in Kley et. al (2015+). The estimate returned is denoted by \(\sigma(\tau_1, \tau_2; \omega)\) on p. 26 of the arXiv preprint.

Note the “standard deviation” estimated here is not the square root of the complex-valued variance. It’s real part is the square root of the variance of the real part of the estimator and the imaginary part is the square root of the imaginary part of the variance of the estimator.

**Value**

Returns the estimate described above.

**References**


---

**getStdError-QuantileSD**

*Get stdError from a quantile spectral density kernel*

**Description**

The selection mechanism for frequencies and levels operates in the same way as described in getValues-QuantileSD. The format of the output is also described there.

**Usage**

```r
## S4 method for signature 'QuantileSD'
getStdError(object, frequencies = 2 * pi * (0:(object@N - 1))/object@N, levels.1 = getLevels(object, 1),
             levels.2 = getLevels(object, 2))
```

**Arguments**

- `object`: QuantileSD of which to get the stdError
- `frequencies`: a vector of frequencies for which to get the stdError
- `levels.1`: the first vector of levels for which to get the stdError
- `levels.2`: the second vector of levels for which to get the stdError

**Value**

Returns the array stdError that’s a slot of object.
**getTs-QuantileSD**

*Get ts from a quantile spectral density kernel*

**Description**

Get ts from a quantile spectral density kernel

**Usage**

```r
## S4 method for signature 'QuantileSD'
getTs(object)
```

**Arguments**

- `object` QuantileSD of which to get the ts

**Value**

Returns the attribute ts that’s a slot of object.

---

**getType-QuantileSD**

*Get type from a quantile spectral density kernel*

**Description**

Get type from a quantile spectral density kernel

**Usage**

```r
## S4 method for signature 'QuantileSD'
getType(object)
```

**Arguments**

- `object` QuantileSD of which to get the type

**Value**

Returns the attribute type that’s a slot of object.
getValues-FreqRep  

Get values from a frequency representation.

**Description**

For two vectors frequencies and levels the values from an object of type `FreqRep` are returned.

**Usage**

```r
## S4 method for signature 'FreqRep'
getValues(object, frequencies = 2 * pi *
           (0:(length(object@Y) - 1))/length(object@Y), levels = object@levels)
```

**Arguments**

- `object`: `FreqRep` of which to get the values
- `frequencies`: a vector of frequencies for which to get the values
- `levels`: a vector of levels for which to get the values

**Details**

The two parameters `frequencies` and `levels` are expected to be vectors of reals; an error is thrown otherwise. If any of the frequencies or levels requested is not available from `object` a warning is issued, and the values with frequencies and levels closest to the ones requested are returned. Note that the frequencies are transformed to 

\[ [0, \pi] \]

using `frequenciesValidator` when checking if they are available in `object`.

The returned array of values is of dimension \([J,K,B+1]\), where \(J=\text{length}(\text{frequencies}),K=\text{length}(\text{levels})\), and \(B\) denotes the value stored in slot \(B\) of `object`. At position \((j,k,b)\) the returned value is the one corresponding to \(\text{frequencies}[j]\) and \(\text{levels}[k]\) that are closest to the frequencies and levels available in `object`; `closest.pos` is used to determine what closest to means.

**Value**

Returns data from the array values that’s a slot of `object`.

**Examples**

```r
Y <- rnorm(32)
freq <- 2*pi*c(0:31)/32
levels <- c(0.25, 0.5, 0.75)
cFT <- clippedFT(Y, freq, levels)
V.all <- getValues(cFT)
V.coarse <- getValues(cFT, frequencies = 2*pi*c(0:15)/16, levels = levels)
V.fine <- getValues(cFT, frequencies = 2*pi*c(0:63)/64, levels = levels)
V.part <- getValues(cFT, frequencies = 2*pi*c(0:16)/32, levels = c(0.25))
```
getValues-IntegrQuantileSD

Get values from a simulated integrated quantile spectral density kernel

Description

If none of the optional parameters is specified then the values are returned for all Fourier frequencies in $[0, 2\pi)$ (base given by slot $N$) and all levels available. The frequencies and levels can be freely specified. The returned array then has, at position $(j,k1,k2,b)$, the value corresponding to the frequencies $[j]$, levels.1$[k1]$ and levels.2$[k2]$ that are closest to the frequencies, levels.1 and levels.2 available in object; closest.pos is used to determine what closest to means.

Usage

```r
## S4 method for signature 'IntegrQuantileSD'
getValues(object, frequencies = 2 * pi *
(0:(getN(object@qsd) - 1))/getN(object@qsd), levels.1 = getLevels(object, 1), levels.2 = getLevels(object, 2))
```

Arguments

- `object` IntegrQuantileSD of which to get the values
- `frequencies` a vector of frequencies for which to get the values
- `levels.1` the first vector of levels for which to get the values
- `levels.2` the second vector of levels for which to get the values

Value

Returns data from the array values that's a slot of object.

See Also

For examples on how to use this function go to IntegrQuantileSD.

getValues-KernelWeight

Get values from a weight object determined by a kernel function $w$ and a bandwidth $b$.

Description

For an object of type KernelWeight and an optional integer $N$ the weights $W_n$ are returned as a vector that has $W_n(2\pi(k - 1)/n)$ at position $k$. 
## Usage

```r
## S4 method for signature 'KernelWeight'
getValues(object, N = length(object@env$values))
```

### Arguments

- **object**: `KernelWeight` of which to get the values
- **N**: a numeric specifying the number of equally spaced Fourier frequencies from \([0, 2\pi]\) for which the weight will be computed; by default the number N specified on construction.

### Value

Returns a vector of size N as described in the Details section.

---

### Description

For vectors `frequencies`, `levels.1` and `levels.2` the values from an object of type `QuantilePG` are returned.

## Usage

```r
## S4 method for signature 'QuantilePG'
getValues(object, frequencies = 2 * pi *
          (0:(length(object@freqRep@Y) - 1))/length(object@freqRep@Y),
          levels.1 = getLevels(object, 1), levels.2 = getLevels(object, 2))
```

### Arguments

- **object**: `QuantilePG` of which to get the values
- **frequencies**: a vector of frequencies for which to get the values
- **levels.1**: the first vector of levels for which to get the values
- **levels.2**: the second vector of levels for which to get the values

### Details

Fetching of the periodogram values basically happens by passing `frequencies` and the union of `levels.1` and `levels.2` to `getValues`. Therefore, the parameters `frequencies`, `levels.1` and `levels.1` are expected to be vectors of reals; an error is thrown otherwise. If any of the `frequencies`, `levels.1` and `levels.2` requested is not available from `object` a warning is issued. Note that the frequencies are transformed to \([0, \pi]\) using `frequenciesValidator` when checking if they are available in `object`. 

---

### getValues-QuantilePG

Get values from a quantile periodogram.

---

---

---

---
The returned array of values is of dimension $[J,K_1,K_2,B+1]$, where $J=\text{length}(\text{frequencies})$, $K_1=\text{length}(\text{levels.1})$, $K_2=\text{length}(\text{levels.2})$, and $B$ denotes the value stored in slot $B$ of freqRep that’s a slot of object. At position $(j,k_1,k_2,b)$ the returned value is the one corresponding to \text{frequencies}[$j$], \text{levels.1}[$k_1$] and \text{levels.2}[$k_2$] that are closest to the \text{frequencies}, \text{levels.1} and \text{levels.2} available in object; \text{closest.pos} is used to determine what closest to means.

**Value**

Returns data from the array values that’s a slot of object.

**Examples**

```r
Y <- rnorm(32)
freq <- 2*pi*c(0:31)/32
levels <- c(0.25,0.5,0.75)
qPG <- quantilePG(Y, levels.1=levels)
V.all <- getValues(qPG)
V.coarse <- getValues(qPG, frequencies = 2*pi*c(0:15)/16)
V.fine <- getValues(qPG, frequencies = 2*pi*c(0:63)/64)
V.part <- getValues(qPG, frequencies = 2*pi*c(0:16)/32,
                    levels.1 = c(0.25), levels.2 = c(0.5,0.75))
```

**Description**

If none of the optional parameters is specified then the values are returned for all Fourier frequencies in $[0,2\pi)$ (base given by slot $N$) and all levels available. The frequencies and levels can be freely specified. The returned array then has, at position $(j,k_1,k_2,b)$, the value corresponding to the \text{frequencies}[$j$], \text{levels.1}[$k_1$] and \text{levels.2}[$k_2$] that are closest to the \text{frequencies}, \text{levels.1} and \text{levels.2} available in object; \text{closest.pos} is used to determine what closest to means, $b=1$ corresponds to the estimator, while $b>1$ corresponds to the estimator determined from the $b$-1th bootstrap replicate.

**Usage**

```r
## S4 method for signature 'QuantileSD'
getValues(object, frequencies = 2 * pi * (0:(object@N - 1))/object@N, levels.1 = getLevels(object, 1),
          levels.2 = getLevels(object, 2))
```

**Arguments**

- **object** QuantileSD of which to get the values
- **frequencies** a vector of frequencies for which to get the values
- **levels.1** the first vector of levels for which to get the values
- **levels.2** the second vector of levels for which to get the values
**getValue-SmoothedPG**

**Value**

Returns data from the array `values` that's a slot of `object`.

**See Also**

For examples on how to use this function go to `QuantileSD`.

---

**getValue-SmoothedPG**  
*Get values from a smoothed quantile periodogram.*

**Description**

The returned array of values is of dimension \([j, k_1, k_2, B+1]\), where \(J=\text{length}(\text{frequencies})\), \(K_1=\text{length}(\text{levels.1})\), \(K_2=\text{length}(\text{levels.2})\), and \(B\) denotes the value stored in slot \(B\) of \(\text{freqRep}\) [that is the number of bootstrap repetitions performed on initialization]. At position \((j, k_1, k_2, b)\) the returned value is the one corresponding to \(\text{frequencies}[j], \text{levels.1}[k_1]\) and \(\text{levels.2}[k_2]\) that are closest to the \(\text{frequencies}\), \(\text{levels.1}\) and \(\text{levels.2}\) available in \(\text{object}\); \(\text{closest.pos}\) is used to determine what closest to means. \(b=1\) corresponds to the estimate without bootstrapping; \(b>1\) corresponds to the \(b=1\)st bootstrap estimate.

**Usage**

```r
## S4 method for signature 'SmoothedPG'
getValues(object, frequencies = 2 * pi *
             (0:(length(object$qPG$freqRep@Y) - 1))/length(object$qPG$freqRep@Y),
             levels.1 = getLevels(object, 1), levels.2 = getLevels(object, 2))
```

**Arguments**

- `object`  
  SmoothedPG of which to get the values
- `frequencies`  
  a vector of frequencies for which to get the values
- `levels.1`  
  the first vector of levels for which to get the values
- `levels.2`  
  the second vector of levels for which to get the values

**Value**

Returns data from the array `values` that's a slot of `object`.

**See Also**

An example on how to use this function is analogously to the example given in `getValue-QuantilePG`.
getValues-SpecDistrWeight

*Get values from a weight object of type SpecDistrWeight*

**Description**

For an object of type SpecDistrWeight and an optional integer N the weights $W_n$ are returned as a vector that has $W_n(2\pi(k - 1)/n)$ at position k.

**Usage**

```r
## S4 method for signature 'SpecDistrWeight'
getValues(object, N = length(object@env$values))
```

**Arguments**

- `object`: SpecDistrWeight of which to get the values
- `N`: a numeric specifying the number of equaly spaced Fourier frequencies from $[0, 2\pi)$ for which the weight will be computed; by default the number N specified on construction.

**Value**

Returns a vector of size N as described in the Description section.

getW-KernelWeight

*Get attribute W (kernel used for smoothing) from a KernelWeight.*

**Description**

Get attribute W (kernel used for smoothing) from a KernelWeight.

**Usage**

```r
## S4 method for signature 'KernelWeight'
getW(object)
```

**Arguments**

- `object`: KernelWeight from which to get the kernel W.

**Value**

Returns the W attribute.
**getWeight-SmoothedPG**

Get associated Weight from a SmoothedPG.

**Description**

Get associated Weight from a SmoothedPG.

**Usage**

```r
## S4 method for signature 'SmoothedPG'
getWeight(object)
```

**Arguments**

- `object`: SmoothedPG from which to get the Weight.

**Value**

Returns the Weight object associated.

---

**getWnj-KernelWeight**

Get attribute Wnj from a QSpecQuantity.

**Description**

If the optional parameter j is supplied, then only the jth element(s) of the vector will be returned, the entire vector otherwise.

**Usage**

```r
## S4 method for signature 'KernelWeight'
getWnj(object, j)
```

**Arguments**

- `object`: KernelWeight from which to get the Wnj.
- `j`: an integer or vector of indices specifying which Wnj[j] to return.

**Value**

Returns levels attribute, as a vector of real numbers.

**Examples**

```r
wgt <- kernelWeight(W=W1, N=2^3, bw=0.7)
getWnj(wgt)
getWnj(wgt, 2)
getWnj(wgt, c(2,7))
```
getY-FreqRep

Get Y from a FreqRep object.

**Description**

Get Y from a FreqRep object.

**Usage**

```r
## S4 method for signature 'FreqRep'
getY(object)
```

**Arguments**

- `object` FreqRep of which to get the Y

**Value**

Returns the attribute Y that's a slot of object.

---

increasePrecision-QuantileSD

Increase the precision of a QuantileSD

**Description**

The precision is increased by generating an additional R QuantilePG objects (independent of the previous ones) and then including them in the average.

**Usage**

```r
## S4 method for signature 'QuantileSD'
increasePrecision(object, R = 1, quiet = FALSE)
```

**Arguments**

- `object` The QuantileSD of which to increase the precision.
- `R` value of which to enlarge R
- `quiet` Don’t report progress to console when computing the R independent quantile periodograms.

**Value**

Returns an QuantileSD object determined from oldR + R independent repetitions.
IntegrQuantileSD-class

Examples

# First simulate a copula spectral density from R=20 independent runs.
csd <- quantileSD(N=2^9, ts=ts1, levels=1=c(0.25,0.5), type="copula", R=20)

# Check out the result:
getR(csd)
plot(csd)

# Now increase the number of independent simulation runs to 50.
csd <- increasePrecision(csd, R=30)

# Check out the (more precise) result:
getR(csd)
plot(csd)

IntegrQuantileSD-class

Class for a simulated integrated quantile (i.e., Laplace or copula) density kernel.

Description

IntegrQuantileSD is an S4 class that implements the necessary calculations to determine an integrated version of the quantile spectral density kernel (computed via QuantileSD). In particular it can be determined for any model from which a time series of length \( n \) can be sampled via a function call `ts(N)`.

Details

In the simulation the quantile spectral density is first determined via QuantileSD, its values are recovered using `getValues-QuantileSD` and then cumulated using `cumsum`.

Note that, all remarks made in the documentation of the super-class `QSpecQuantity` apply.

Slots

- `qsd` a QuantileSD from which to begin the computations.

Examples

### Simulate a time series \( Y_1, \ldots, Y_{128} \) from the QAR(1) process discussed in
### Dette et. al (2014).

```r
set.seed(2581)
Y <- ts(128)
```

### For a defined set of quantile levels ...

```r
# Simulate a time series Y1,...,Y128 from the QAR(1) process discussed in
# Dette et. al (2014).
set.seed(2581)
Y <- ts(128)
```
IntegrQuantileSD-constructor

Create an instance of the IntegrQuantileSD class.

Description

Create an instance of the IntegrQuantileSD class.

Usage

integrQuantileSD(object = 2^8, type = c("copula", "Laplace"), ts = rnorm,
seed.init = 2581, levels.1 = 0.5, levels.2 = levels.1, R = 1,
quiet = FALSE)

Arguments

object         the number N of Fourier frequencies to be used; alternatively a QuantileSD
               object can be supplied (then all the other parameters will be ignored)

type           can be either "Laplace" or "copula"; indicates whether the marginals are to be
               assumed uniform [0, 1] distributed.

ts             a function that has one argument n and, each time it is invoked, returns a new
               time series from the model for which the integrated quantile spectral density
               kernel is to be simulated.

seed.init      an integer serving as an initial seed for the simulations.

levels.1       A vector of length K1 containing the levels x1 at which the QuantileSD is to be
               determined.
levels.2  A vector of length \( K \) containing the levels \( x \) at which the QuantileSD is to be determined.

\( R \)  an integer that determines the number of independent simulations; the larger this number the more precise is the result.

quiet  Don’t report progress to console when computing the \( R \) independent quantile periodograms.

**Value**

Returns an instance of `IntegrQuantileSD`.

**See Also**

For an example see `IntegrQuantileSD`.

---

**is.wholenumber**

Checks whether \( x \) contains integer numbers.

**Description**

Borrowed from the example in `integer`.

**Usage**

```r
is.wholenumber(x, tol = .Machine$double.eps^0.5)
```

**Arguments**

- \( x \) a vector to be checked for integers
- \( tol \) an optional parameter specifying to which precision the check is to be performed.

**Value**

Returns a vector of logicals with the same length as \( x \); each element \( i \) is TRUE iff \( x[i] \) is an integer.

**Examples**

```r
## Not run:
is.wholenumber(1) # is TRUE
(x <- seq(1, 5, by = 0.5))
is.wholenumber(x) #--> TRUE FALSE TRUE ...
## End(Not run)
```
kernels

Kernel function.

Description
Implementations of kernel functions

Usage
$w_0(x)$
$w_1(x)$
$w_2(x)$
$w_3(x)$
$w_{\text{Daniell}}(x, a = (\pi/2))$

Arguments
$x$ real-valued argument to the function; can be a vector
$a$ real number between 0 and $\pi$

Details
Daniell kernel function $w_0$:
\[
\frac{1}{2\pi} I\{|x| \leq \pi\}.
\]
Epanechnikov kernel $w_1$ (i.e., variance minimizing kernel function of order 2):
\[
\frac{3}{4\pi}(1 - \frac{x}{\pi})^2 I\{|x| \leq \pi\}.
\]
Variance minimizing kernel function $w_2$ of order 4:
\[
\frac{15}{32\pi}(-99(x/\pi)^6 + 189(x/\pi)^4 - 105(x/\pi)^2 + 15) I\{|x| \leq \pi\}.
\]
Variance minimizing kernel function $w_3$ of order 6:
\[
\frac{35}{256\pi}(-99(x/\pi)^6 + 189(x/\pi)^4 - 105(x/\pi)^2 + 15) I\{|x| \leq \pi\}.
\]
Kernel yield by convolution of two Daniell kernels:
\[
\frac{1}{\pi + a} \left(1 - \frac{|x| - a}{\pi - a} I\{|a| \leq |x| \leq \pi\}\right).
\]
KernelWeight-class

**Examples**

```r
plot(x=seq(-8,8,0.05), y=W0(seq(-8,8,0.05)), type="l")
plot(x=seq(-8,8,0.05), y=W1(seq(-8,8,0.05)), type="l")
plot(x=seq(-8,8,0.05), y=W2(seq(-8,8,0.05)), type="l")
plot(x=seq(-8,8,0.05), y=W3(seq(-8,8,0.05)), type="l")
plot(x=seq(-pi,pi,0.05), y=WDanielI(seq(-pi,pi,0.05), a=(pi/2)), type="l")
```

---

**KernelWeight-class**

*Class for Brillinger-type Kernel weights.*

**Description**

KernelWeight is an S4 class that implements a weighting function by specification of a kernel function \( w \) and a scale parameter \( bw \).

**Details**

It extends the class `Weight` and writes

\[
W_N(2\pi(k-1)/N) := \sum_{j \in \mathbb{Z}} bw^{-1} W(2\pi bw^{-1}[(k-1)/N + j])
\]

to values[k] [nested inside env] for k=1,...,N. The number length(values) of Fourier frequencies for which \( W_N \) will be evaluated may be set on construction or updated when evoking the method `getValues`. To standardize the weights used in the convolution to unity

\[
W^j_N := \sum_{j \neq s=0}^{N-1} W_n(2\pi s/N)
\]

is stored to \( WN[j] \) for s=1,...,N, for later usage.

**Slots**

- `w` a kernel function
- `bw` bandwidth
- `env` An environment to allow for slots which need to be accessible in a call-by-reference manner:
  - `values` A vector storing the weights; see the Details section.
  - `WNj` A vector storing the terms used for normalization; see the Details section.

**References**


**See Also**

Examples for implementations of kernels \( w \) can be found at: kernels.
KernelWeight-constructor

Create an instance of the KernelWeight class.

Description

Create an instance of the KernelWeight class.

Usage

```r
kernelWeight(W = W0, N = 1, bw = 0.1 * N^(-1/5), descr = paste("bw=",
round(bw, 3), ", N=", N, sep = ""))
```

Arguments

- `W`: A kernel function
- `N`: Fourier basis; number of grid points in \([0, 2\pi]\) for which the weights will be computed.
- `bw`: bandwidth; if a vector, then a list of weights is returned
- `descr`: a description to be used in some plots

Value

Returns an instance of KernelWeight.

See Also

- `kernels`

Examples

```r
wgt1 <- kernelWeight(W=W0, N=16, bw=c(0.1,0.3,0.7))
print(wgt1)
wgt2 <- kernelWeight(W=W1, N=2^8, bw=0.1)
plot(wgt2, main="Weights determined from Epanechnikov kernel")
```
MovingBlocks-class

Class for Moving Blocks Bootstrap implementation.

Description

MovingBlocks is an S4 class that implements the moving blocks bootstrap described in Künsch (1989).

Details

MovingBlocks extends the S4 class BootPos and the remarks made in its documentation apply here as well.

The Moving Blocks Bootstrap method of Künsch (1989) resamples blocks randomly, with replacement from the collection of overlapping blocks of length $l$ that start with observation 1, 2, ..., $N-l+1$. A more precise description of the procedure can also be found in Lahiri (1999), p. 389.

References


See Also

getPositions-MovingBlocks

MovingBlocks-class

Create an instance of the MovingBlocks class.

Description

Create an instance of the MovingBlocks class.

Usage

movingBlocks(l, N)

Arguments

- `l`: the block length for the block bootstrap methods
- `N`: number of available observations to bootstrap from

Value

Returns an instance of MovingBlocks.
plot-FreqRep  

Plot the values of the FreqRep.

Description

Creates a K x 2 plot depicting a FreqRep object. Each of the K “lines” of subplots shows the frequency representation for one value of \( \tau \). The real and imaginary part are shown on the left and the right, respectively.

Usage

```r
## S4 method for signature 'FreqRep,ANY'
plot(x, ratio = 2, frequencies = 2 * pi *
     (1:(floor(length(x@Y)/2)))/length(x@Y), levels = x@levels)
```

Arguments

- **x**: The FreqRep to plot.
- **ratio**: quotient of width over height of the subplots; use this parameter to produce landscape or portrait shaped plots.
- **frequencies**: a set of frequencies for which the values are to be plotted.
- **levels**: a set of levels for which the values are to be plotted.

Value

Plots the FreqRep for all frequencies and levels specified.

plot-IntegrQuantileSD  

Plot the values of the IntegrQuantileSD.

Description

Creates a K x K plot depicting an integrated quantile spectral density. In each of the subplots either the real part (on and below the diagonal; i.e., \( \tau_1 \leq \tau_2 \)) or the imaginary part (above the diagonal; i.e., \( \tau_1 > \tau_2 \)) of

- the integrated quantile spectral density (black line),

for the combination of levels \( \tau_1 \) and \( \tau_2 \) denoted on the left and bottom margin of the plot are displayed.
Usage

```r
## S4 method for signature 'IntegrQuantileSD,ANY'
plot(x, ratio = 3/2, widthlab = lcm(1),
     xlab = expression(omega/2 * pi), ylab = NULL, frequencies = 2 * pi *
     (1:(floor(getN(getQuantileSD(x))/2)))/getN(getQuantileSD(x)),
     levels = getLevels(x, 1))
```

Arguments

- **x**: The `IntegrQuantileSD` to plot
- **ratio**: quotient of width over height of the subplots; use this parameter to produce landscape or portrait shaped plots.
- **widthlab**: width for the labels (left and bottom); default is lcm(1), cf. `layout`.
- **xlab**: label that will be shown on the bottom of the plots; can be an expression (for formulas), characters or NULL to force omission (to save space).
- **ylab**: label that will be shown on the left side of the plots; can be an expression (for formulas), characters or NULL to force omission (to save space).
- **frequencies**: a set of frequencies for which the values are to be plotted.
- **levels**: a set of levels for which the values are to be plotted.

Value

Plots the simulated integrated quantile spectral density for all frequencies and levels specified.

---

**plot-KernelWeight**

*Plot the values of the KernelWeight.*

Description

Creates a plot visualizing the weights $W_n(\omega)$ [cf. `KernelWeight-class`] that are used to estimate the quantile spectral density.

Usage

```r
## S4 method for signature 'KernelWeight,missing'
plot(x, y, ylab = expression(W[n](omega)),
     xlab = expression(omega), main = x@descr, ...)
```

Arguments

- **x**: The `KernelWeight` to plot.
- **y**: missing arg from the generic; will be ignored.
- **ylab**: label for the y-axis; optional
- **xlab**: label for the x-axis; optional
- **main**: title (on top) of the plot; optional
- **...**: optional parameters used for plotting
Details

In the plot the values at the frequencies $2\pi j/N, j = L + 1 - N, \ldots, L, L := \lfloor N/2 \rfloor$ are shown, where $N$ is the parameter specified on construction of the object or $N := 3$, if that parameter was smaller than three. A warning is given in the later case.

Value

Plots the KernelWeight.

Examples

```r
plot(kernelWeight(W1, bw=0.3),
     ylab=expression(W[n](x)),
     xlab=expression(x),
     main="Weights to an Epanechnikov kernel", sub="bw=0.3")
```

Description

Creates a $K \times K$ plot depicting a quantile periodogram. Optionally, a simulated copula spectral density can be displayed. In each of the subplots either the real part (on and below the diagonal; i.e., $\tau_1 \leq \tau_2$) or the imaginary parts (above the diagonal; i.e., $\tau_1 > \tau_2$) of

- the quantile periodogram (black line),
- a simulated quantile spectral density (red line),

for the combination of levels $\tau_1$ and $\tau_2$ denoted on the left and bottom margin of the plot are displayed.

Usage

```r
## S4 method for signature 'QuantilePG,ANY'
plot(x, qsd, ratio = 3/2, widthlab = lcm(1),
     xlab = expression(omega/2 * pi), ylab = NULL,
     type.scaling = c("individual", "real-imaginary", "all"),
     frequencies = x@frequencies[-which(x@frequencies == 0)],
     levels = intersect(x@levels[[1]], x@levels[[2]]))
```

Arguments

- **x** The QuantilePG object to plot
- **qsd** a QuantileSD object; will be plotted if not missing.
- **ratio** quotient of width over height of the subplots; use this parameter to produce landscape or portrait shaped plots.
- **widthlab** width for the labels (left and bottom); default is lcm(1), cf. layout.
plot-QuantileSD

xlab label that will be shown on the bottom of the plots; can be an expression (for formulas), characters or NULL to force omission (to save space).

ylab label that will be shown on the left side of the plots; can be an expression (for formulas), characters or NULL to force omission (to save space).

type.scaling a method for scaling of the subplots; currently there are three options: "individual" will scale each of the \(K^2\) subplots to minimum and maximum of the values in that plot, "real-imaginary" will scale each of the subplots displaying real parts and each of the subplots displaying imaginary parts to the minimum and maximum of the values display in these subportion of plots. The option "all" will scale the subplots to the minimum and maximum in all of the subplots.

frequencies a set of frequencies for which the values are to be plotted; default is all available frequencies but 0; if 0 is the only available frequency, then only 0 will be used.

levels a set of levels for which the values are to be plotted.

Value

Returns the plot described in the Description section.

Description

Creates a \(K \times K\) plot depicting a quantile spectral density. In each of the subplots either the real part (on and below the diagonal; i.e., \(\tau_1 \leq \tau_2\)) or the imaginary parts (above the diagonal; i.e., \(\tau_1 > \tau_2\)) of

- the quantile spectral density (red line),
- the means of the quantile periodograms used in the simulation (black line),

for the combination of levels \(\tau_1\) and \(\tau_2\) denoted on the left and bottom margin of the plot are displayed.

Usage

```r
## S4 method for signature 'QuantileSD,ANY'
plot(x, ratio = 3/2, widthlab = lcm(1),
     xlab = expression(omega/2 * pi), ylab = NULL, frequencies = 2 * pi *
     (1:(floor(x@N/2))/x@N, levels = getLevels(x, 1))
```

Arguments

- **x** The `QuantileSD` to plot
- **ratio** quotient of width over height of the subplots; use this parameter to produce landscape or portrait shaped plots.
- **widthlab** width for the labels (left and bottom); default is lcm(1), cf. layout.
plot-SmoothedPG

xlab
label that will be shown on the bottom of the plots; can be an expression (for formulas), characters or NULL to force omission (to save space).

ylab
label that will be shown on the left side of the plots; can be an expression (for formulas), characters or NULL to force omission (to save space).

frequencies
a set of frequencies for which the values are to be plotted.

levels
a set of levels for which the values are to be plotted.

Value
Plots the simulated quantile spectral density for all frequencies and levels specified.

Description
Creates a K x K plot depicting a smoothed quantile periodogram. Optionally, the quantile periodogram on which the smoothing was performed, a simulated quantile spectral density, and pointwise confidence intervals can be displayed. In each of the subplots either the real part (on and below the diagonal; i.e., $\tau_1 \leq \tau_2$) or the imaginary parts (above the diagonal; i.e., $\tau_1 > \tau_2$) of

- the smoothed quantile periodogram (blue line),
- the quantile periodogram that was used for smoothing (gray line),
- a simulated quantile spectral density (red line),
- pointwise (asymptotic) confidence intervals (light gray area),

for the combination of levels $\tau_1$ and $\tau_2$ denoted on the left and bottom margin of the plot are displayed.

Usage
```r
## S4 method for signature 'SmoothedPG,ANY'
plot(x, plotPG = FALSE, qsd, ptw.CIs = 0.1,
     type.CIs = c("naive.sd", "boot.sd", "boot.full"), ratio = 3/2,
     widthlab = lcm(1), xlab = expression(omega/2 * pi), ylab = NULL,
     type.scaling = c("individual", "real-imaginary", "all"),
     frequencies = x@frequencies, levels = intersect(x@levels[[1]], x@levels[[2]]))
```

Arguments

- `x`
  The `SmoothedPG` object to plot

- `plotPG`
a flag indicating whether the QuantilePG object associated with the `SmoothedPG` x is also to be plotted.

- `qsd`
a `QuantileSD` object; will be plotted if not missing.
The confidence level for the confidence intervals to be displayed; must be a number from [0.1]; if null, then no confidence intervals will be plotted.

describes the method to be used for determining the confidence intervals; the methods available are those provided by `getPointwiseCIs`.

The quotient of width over height of the subplots; use this parameter to produce landscape or portrait shaped plots.

width for the labels (left and bottom); default is `lcm(1)`, cf. `layout`.

label that will be shown on the bottom of the plots; can be an expression (for formulas), characters or `NULL` to force omission (to save space).

label that will be shown on the left side of the plots; can be an expression (for formulas), characters or `NULL` to force omission (to save space).

a method for scaling of the subplots; currently there are three options: "individual" will scale each of the \(K \times 2\) subplots to minimum and maximum of the values in that plot, "real-imaginary" will scale each of the subplots displaying real parts and each of the subplots displaying imaginary parts to the minimum and maximum of the values display in these subportion of plots. The option "all" will scale the subplots to the minimum and maximum in all of the subplots.

a set of frequencies for which the values are to be plotted.

a set of levels for which the values are to be plotted.

Returns the plot described in the Description section.

---

**plot-SpecDistrWeight**

**Plot the values of the SpecDistrWeight.**

**Description**

Creates a plot visualizing the weights \(W_n(\omega)\) [cf. SpecDistrWeight-class] that are used to estimate the integrated quantile spectral density.

**Usage**

```r
## S4 method for signature 'SpecDistrWeight,missing'
plot(x, y, ylab = expression(W[n](omega)),
     xlab = expression(omega), main = x@descr, ...)
```

**Arguments**

- **x**  
The `SpecDistrWeight` to plot.
- **y**  
missing arg from the generic; will be ignored.
- **ylab**  
label for the y-axis; optional
- **xlab**  
label for the x-axis; optional
- **main**  
titel (on top) of the plot; optional
- **...**  
optional parameters used for plotting
Details

In the plot the values at the frequencies \(2\pi j/128, j = -63, \ldots, 64\) are shown.

Value

Plots the SpecDistrWeight.

Examples

```r
plot(specDistrWeight(),
     ylab=expression(W[n](x)),
     xlab=expression(x))
```

Description

`QRegEstimator-class` is an S4 class that implements the necessary calculations to determine the frequency representation based on the weighted \(L_1\)-projection of a time series as described in Dette et. al (2014+). As a subclass to `FreqRep` it inherits slots and methods defined there.

Details

For each frequency \(\omega\) from frequencies and level \(\tau\) from levels the statistic

\[
\hat{b}_{n,\tau}(\omega) := \arg \max_{a \in \mathbb{R}, b \in \mathbb{C}} \sum_{t=0}^{n-1} \rho_{\tau}(Y_t - a - Re(b) \cos(\omega t) - Im(b) \sin(\omega t)),
\]

is determined and stored to the array values.

The solution to the minimization problem is determined using the function `rq` from the `quantreg` package.

All remarks made in the documentation of the super-class `FreqRep` apply.

Slots

- `method` method used for computing the quantile regression estimates. The choice is passed to `qr`; see the documentation of `quantreg` for details.
- `parallel` a flag that signalizes that parallelization mechanisms from the package `snowfall` may be used.

References

QRegEstimator-constructor

Create an instance of the QRegEstimator class.

Description

The parameter type.boot can be set to choose a block bootstrapping procedure. If "none" is chosen, a moving blocks bootstrap with l=length(Y) and N=length(Y) would be done. Note that in that case one would also chose B=0 which means that getPositions would never be called. If B>0 then each bootstrap replication would be the undisturbed time series.

Usage

QRegEstimator(Y, frequencies = 2 * pi/length(Y) * 0:(length(Y) - 1),
levels = 0.5, isRankBased = TRUE, B = 0, l = 0,
type.boot = c("none", "mbb"), method = c("br", "fn", "pfn", "fnc",
"lasso", "scad"), parallel = FALSE)

Arguments

Y A vector of real numbers containing the time series from which to determine the quantile periodogram or a ts object or a zoo object.
frequencies A vector containing frequencies at which to determine the QRegEstimator.
levels A vector of length K containing the levels x at which the QRegEstimator is to be determined.
isRankBased If true the time series is first transformed to pseudo data [cf.FreqRep].
B number of bootstrap replications
l (expected) length of blocks
type.boot A flag to choose a method for the block bootstrap; currently two options are implemented: "none" and "mbb" which means to do a moving blocks bootstrap with B and l as specified.
method method used for computing the quantile regression estimates. The choice is passed to qr; see the documentation of quantreg for details.
parallel a flag to allow performing parallel computations.

Value

Returns an instance of QRegEstimator.

Examples

library(snowfall)

Y <- rnorm(100) # Try 2000 and parallel computation will in fact be faster.
# Compute without using snowfall capabilities

```r
system.time(
  qRegEst1 <- qRegEstimator(Y, levels=seq(0.25, 0.75, 0.25), method="fn", parallel=FALSE)
)
```

# Set up snowfall

```r
sfInit(parallel=TRUE, cpus=2, type="SOCK")
sfLibrary(quantreg)
sfExportAll()
```

# Compare how much faster the computation is when done in parallel

```r
system.time(
  qRegEst2 <- qRegEstimator(Y, levels=seq(0.25, 0.75, 0.25), method="fn", parallel=TRUE)
)
sfStop()
```

# Compare results

```r
V1 <- getValues(qRegEst1)
V2 <- getValues(qRegEst2)
sum(abs(V1-V2)) # Returns: [1] 0
```

---

### QSpecQuantity-class

**Class for a Quantile Spectral Estimator.**

#### Description

QSpecQuantity is an S4 class that provides a common interface to objects that are of the functional form \( f(\omega; x_1, x_2) \), where \( \omega \) is a frequency parameter and \( x_1, x_2 \) are level parameters. For each combination of parameters a complex number can be stored. Examples for objects of this kind currently include the quantile (i.e., Laplace or copula) spectral density kernel [cf. QuantileSD for an implementation], an integrated version of the quantile spectral density kernels [cf. IntegrQuantileSD for an implementation], and estimators of it [cf. QuantilePG and SmoothedPG for implementations].

#### Slots

- **values** The array holding the values \( f(\omega; x_1, x_2) \).
- **frequencies** The frequencies \( \omega \) for which the values are available.
- **levels** A list of vectors containing the levels \( x_i \) serving as argument for the estimator.
QuantilePG-class

Class for a quantile (i.e., Laplace or copula) periodogram.

Description

QuantilePG is an S4 class that implements the necessary calculations to determine one of the periodogram-like statistics defined in Dette et al (2014+) and Kley et al (2015+).

Details

Performs all the calculations to determine a quantile periodogram from a FreqRep object upon initialization (and on request stores the values for faster access). The two methods available for the estimation are the ones implemented as subclasses of FreqRep:

- the Fourier transformation of the clipped time series \((\{I\{Y_t \leq q\}\})\) [cf. ClippedFT], or
- the weighted \(L_1\)-projection of \((Y_t)\) onto an harmonic basis [cf. QRegEstimator].

All remarks made in the documentation of the super-class QSpecQuantity apply.

References


Examples

```
# This script illustrates how to work with QuantilePG objects

# Simulate a time series Y1,...,Y128 from the QAR(1) process discussed in
# Dette et. al (2014).
Y <- ts(128)

# For a defined set of quantile levels
levels <- c(0.25,0.5,0.75)

# the various quantile periodograms can be calculated calling quantilePG:

# For a copula periodogram as in Dette et. al (2014) the option 'type="qr"'
# has to be used:
system.time(
    qPG.qr <- quantilePG(Y, levels.1 = levels, type="qr"))

# For the CR-periodogram as in Kley et. al (2014) the option 'type="clipped"'
# has to be used. If bootstrap estimates are to be used the parameters
# type.boot, B and l need to be specified.
```
QuantilePG-constructor

Create an instance of the QuantilePG class.

Description

The parameter type.boot can be set to choose a block bootstrapping procedure. If "none" is chosen, a moving blocks bootstrap with l=length(Y) and N=length(Y) would be done. Note that
in that case one would also choose $B=0$ which means that getPositions would never be called. If $B>0$ then each bootstrap replication would be the undisturbed time series.

Usage

```r
quantilePG(Y, frequencies = 2 * pi/length(Y) * 0:(length(Y) - 1),
levels.1 = 0.5, levels.2 = levels.1, isRankBased = TRUE,
type = c("clipped", "qr"), type.boot = c("none", "mbb"), B = 0, l = 0,
method = c("br", "fn", "pfn", "fnc", "lasso", "scad"), parallel = FALSE)
```

Arguments

- `Y`: A vector of real numbers containing the time series from which to determine the quantile periodogram or a `ts` object or a `zoo` object.
- `frequencies`: A vector containing frequencies at which to determine the quantile periodogram.
- `levels.1`: A vector of length $k_1$ containing the levels $x_1$ at which the QuantilePG is to be determined.
- `levels.2`: A vector of length $k_2$ containing the levels $x_2$.
- `isRankBased`: If true the time series is first transformed to pseudo data [cf. `freqrep`].
- `type`: A flag to choose the type of the estimator. Can be either "clipped" or "qr". In the first case `ClippedFT` is used as a frequency representation, in the second case `QRegEstimator` is used.
- `type.boot`: A flag to choose a method for the block bootstrap; currently two options are implemented: "none" and "mbb" which means to do a moving blocks bootstrap with $B$ and $l$ as specified.
- `B`: number of bootstrap replications
- `l`: (expected) length of blocks
- `method`: method used for computing the quantile regression estimates. The choice is passed to `qr`; see the documentation of `quantreg` for details.
- `parallel`: a flag to allow performing parallel computations, where possible.

Value

Returns an instance of `QuantilePG`.

---

**QuantileSD-class**

*Class for a simulated quantile (i.e., Laplace or copula) density kernel.*

**Description**

`QuantileSD` is an S4 class that implements the necessary calculations to determine a numeric approximation to the quantile spectral density kernel of a model from which a time series of length $N$ can be sampled via a function call `ts(N)`.
Details

In the simulation a number of \( R \) independent quantile periodograms based on the clipped time series are simulated. If \( \text{type} = \text{"copula"} \), then the rank-based version is used. The sum and the sum of the squared absolute value is stored to the slots \( \text{sumPG} \) and \( \text{sumSqPG} \). After the simulation is completed the mean and it’s standard error (of the simulated quantile periodograms) are determined and stored to \( \text{meanPG} \) and \( \text{stderr} \). Finally, the (copula) spectral density kernel is determined by smoothing real and imaginary part of \( \text{meanPG} \) separately for each combination of levels using \texttt{smooth.spline}.

Note that, all remarks made in the documentation of the super-class \texttt{QSpecQuantity} apply.

Slots

\( N \) a numeric specifying the number of equally spaced Fourier frequencies from \([0, 2\pi]\) for which the (copula) spectral density will be simulated; note that due to the simulation mechanism a larger number will also yield a better approximation.

\( R \) the number of independent repetitions performed; note that due to the simulation mechanism a larger number will also yield a better approximation; can be enlarged using \texttt{increasePrecision-Q quantileSD}.

type can be either \texttt{laplace} or \texttt{copula}; indicates whether the marginals are to be assumed uniform \([0, 1]\) distributed.

ts a function that allows to draw independent samples \( Y_0, \ldots, Y_{n-1} \) from the process for which the (copula) spectral density kernel is to be simulated.

\texttt{seed} last used internally to store the state of the pseudo random number generator, so the precision can be increased by generating more pseudo random numbers that are independent from the ones previously used.

\( \text{sumPG} \) an array used to store the sum of the simulated quantile periodograms

\( \text{sumSqPG} \) an array used to store the sum of the squared absolute values of the simulated quantile periodograms

\( \text{meanPG} \) an array used to store the mean of the simulated quantile periodograms

\( \text{stderr} \) an array used to store the estimated standard error of the mean of the simulated quantile periodograms

References


See Also

Examples for implementations of functions \texttt{ts} can be found at: \texttt{ts-models}.
QuantileSD-constructor

**Examples**

```r
## This script can be used to create and store a QuantileSD object

## Parameters for the simulation:
R <- 50  # number of independent repetitions;
        # R should be much larger than this in practice!
N <- 2^8 # number of Fourier frequencies in [0,2pi)
ts <- ts1 # time series model
levels <- seq(0.1,0.9,0.1) # quantile levels
type <- "copula" # copula, not Laplace, spectral density kernel
seed.init <- 2581  # seed for the pseudo random numbers

## Simulation takes place once the constructor is invoked
qsd <- quantileSD(N=N, seed.init = 2581, type = type,
                  ts = ts, levels.1=levels, R = R)

## The simulated copula spectral density kernel can be called via
V1 <- getValues(qsd)

## It is also possible to fetch the result for only a few levels
levels.few <- c(0.2,0.5,0.7)
V2 <- getValues(qsd, levels.1=levels.few, levels.2=levels.few)

## If desired additional repetitions can be performed to yield a more precise
## simulation result by calling; here the number of independent runs is doubled.
qsd <- increasePrecision(qsd,R)

## Often the result will be stored for later usage.
save(qsd, file="QAR1.rdata")

## Take a brief look at the result of the simulation
plot(qsd, levels=levels.few)

## When plotting more than only few levels it may be a good idea to plot to
## another device; e.g., a pdf-file
K <- length(levels)
plot(qsd)
dev.off()
```

---

**QuantileSD-constructor**

Create an instance of the **QuantileSD** class.

---

**Description**

Create an instance of the **QuantileSD** class.
Usage

quantileSD(N = 2^8, type = c("copula", "Laplace"), ts = rnorm,
    seed.init = runif(1), levels.1, levels.2 = levels.1, R = 1,
    quiet = FALSE)

Arguments

- **N**: the number of Fourier frequencies to be used.
- **type**: can be either Laplace or copula; indicates whether the marginals are to be assumed uniform \([0, 1]\) distributed.
- **ts**: a function that has one argument \(n\) and, each time it is invoked, returns a new time series from the model for which the copula spectral density kernel is to be simulated.
- **seed.init**: an integer serving as an initial seed for the simulations.
- **levels.1**: A vector of length \(k_1\) containing the levels \(x_1\) at which the QuantileSD is to be determined.
- **levels.2**: A vector of length \(k_2\) containing the levels \(x_2\) at which the QuantileSD is to be determined.
- **R**: an integer that determines the number of independent simulations; the larger this number the more precise is the result.
- **quiet**: Don’t report progress to console when computing the \(R\) independent quantile periodograms.

Value

Returns an instance of QuantileSD.

See Also

For examples see QuantileSD.

---

**quantspec-defunct**

*Defunct functions in package quantspec*

Description

These functions have been declared defunct since Version 1.0-1.
Usage

ci(i1, i2, n)

LaplacePeriodogram(X, taus, omegas = 1:(ceiling(length(X)/2) - 1),
    fromRanks = TRUE, showProgressBar = FALSE)

plotLaplacePeriodogram(LPG, taus, F = 1:length(LPG[, 1]),
    CL = 1:length(taus), hRange = FALSE, hOffset = FALSE,
    ylabel = expression(hat(f)[n]^{\text{list}(\text{tau}[1], \text{tau}[2])})(omega)), oma = c(2.5, 2.5, 2.5, 2.5), mar = c(4.5, 4.5, 1, 0) + 0.1, cex.lab = 1.5)

smoothedLaplacePeriodogram(LPG, taus, W)

Arguments

i1 Parameter of DEFUNCT function.
i2 Parameter of DEFUNCT function.
n Parameter of DEFUNCT function.
X Parameter of DEFUNCT function.
taus Parameter of DEFUNCT function.
omegas Parameter of DEFUNCT function.
fromRanks Parameter of DEFUNCT function.
showProgressBar Parameter of DEFUNCT function.
LPG Parameter of DEFUNCT function.
F Parameter of DEFUNCT function.
CL Parameter of DEFUNCT function.
hRange Parameter of DEFUNCT function.
hOffset Parameter of DEFUNCT function.
ylabel Parameter of DEFUNCT function.
oma Parameter of DEFUNCT function.
mar Parameter of DEFUNCT function.
cex.lab Parameter of DEFUNCT function.
W Parameter of DEFUNCT function.
SmoothedPG-class

Class for a smoothed quantile periodogram.

Description

SmoothedPG is an S4 class that implements the necessary calculations to determine a smoothed version of one of the quantile periodograms defined in Dette et al. (2014) and Kley et al. (2014).

Details

For a QuantilePG $Q_n(\omega, x_1, x_2)$ and a Weight $W_n(\cdot)$ the smoothed version

$$\frac{2\pi}{n} \sum_{s=1}^{n-1} W_n(\omega - 2\pi s/n) Q_n(2\pi s/n, x_1, x_2)$$

is determined.

The convolution required to determine the smoothed periodogram is implemented using convolve.

Slots

| env | An environment to allow for slots which need to be accessible in a call-by-reference manner:
|-----|------------------------|
| sdNaive | An array used for storage of the naively estimated standard deviations of the smoothed periodogram.
| sdNaive.done | a flag indicating whether sdNaive has been set yet.
| sdBoot | An array used for storage of the standard deviations of the smoothed periodogram, estimated via bootstrap.
| sdBoot.done | a flag indicating whether sdBoot.naive has been set yet.
| qPG | the QuantilePG to be smoothed
| weight | the Weight to be used for smoothing

SmoothedPG-constructor

Create an instance of the SmoothedPG class.

Description

A SmoothedPG object can be created from either

- a numeric, a ts, or a zoo object
- a QuantilePG object.

If a QuantilePG object is used for smoothing, only the weight, frequencies and levels.1 and levels.2 parameters are used; all others are ignored. In this case the default values for the levels are the levels of the QuantilePG used for smoothing. Any subset of the levels available there can be chosen.
Usage

smoothedPG(object, frequencies = 2 * pi/length(object) * 0:(length(object) - 1), levels.1 = 0.5, levels.2 = levels.1, isRankBased = TRUE,
  type = c("clipped", "qr"), type.boot = c("none", "mmb"),
  method = c("br", "fn", "fn", "fnc", "lasso", "scad"), parallel = FALSE,
  B = 0, l = 1, weight = kernelWeight())

Arguments

object a time series (numeric, ts, or zoo object) from which to determine the smoothed periodogram; alternatively a QuantilePG object can be supplied.
frequencies A vector containing frequencies at which to determine the smoothed periodogram.
levels.1 A vector of length K1 containing the levels x1 at which the SmoothedPG is to be determined.
levels.2 A vector of length K2 containing the levels x2.
isRankBased If true the time series is first transformed to pseudo data [cf. FreqRep].
type A flag to choose the type of the estimator. Can be either "clipped" or "qr". In the first case ClippedFT is used as a frequency representation, in the second case QRegEstimator is used.
type.boot A flag to choose a method for the block bootstrap; currently two options are implemented: "none" and "mmb" which means to do a moving blocks bootstrap with B and l as specified.
method method used for computing the quantile regression estimates. The choice is passed to qr; see the documentation of quantreg for details.
parallel a flag to allow performing parallel computations, where possible.
B number of bootstrap replications
l (expected) length of blocks
weight Object of type Weight to be used for smoothing.

Details

The parameter type.boot can be set to choose a block bootstrapping procedure. If "none" is chosen, a moving blocks bootstrap with l=1=\text{length}(Y) and N=\text{length}(Y) would be done. Note that in that case one would also chose B=0 which means that getPositions would never be called. If B>0 then each bootstrap replication would be the undisturbed time series.

Value

Returns an instance of SmoothedPG.

Examples

Y <- rnorm(64)
levels.1 <- c(0.25, 0.5, 0.75)
weight <- kernelWeight(W=W0)
SpecDistrWeight-class  

Class for weights to estimate integrated spectral density kernels.

Description

SpecDistrWeight is an S4 class that implements a weighting function given by

\[ W_n(\alpha) := I\{\alpha \leq 0\} \]

Details

At position \( k \) the value \( W_n(2\pi(k - 1)/n) \) is stored [in a vector values nested inside env] for \( k=1,\ldots,T \). The number length(values) of Fourier frequencies for which \( W_n \) will be evaluated may be set on construction or updated when evoking the method getValues.

SpecDistrWeight-constructor

Create an instance of the SpecDistrWeight class.

Description

Create an instance of the SpecDistrWeight class.

Usage

specDistrWeight(descr = "Spectral Distribution Weights")

Arguments

descr  a description for the weight object
Value

an instance of SpecDistrWeight.

Examples

wgt <- specDistrWeight()

---

timeSeriesValidator  Validates if Y is of an appropriate type and converts to a numeric.

Description

Checks whether Y is either

• numeric,
• a ts object, or
• a zoo object.

If not, an error is returned. If it is one of the three the data is returned as a numeric.

Usage

timeSeriesValidator(Y)

Arguments

Y the time series to be validated.

Value

Returns the time series as a numeric.

Examples

Y <- timeSeriesValidator(sp500)
Y <- timeSeriesValidator(wheatprices)
Y <- timeSeriesValidator(rnorm(10))
## Not run: Y <- timeSeriesValidator("Not a valid input")
Functions to simulate from the time series models in Kley et. al (2014).

Description

Functions to simulate from the time series models in Kley et. al (2014).

Usage

\texttt{ts1(n)}

\texttt{ts2(n)}

\texttt{ts3(n)}

Arguments

n  

length of the time series to be returned

Details

\texttt{ts1} QAR(1) model from Dette et. al (2014+).

\texttt{ts2} AR(2) model from Li (2012):

\texttt{ts3} ARCH(1) model from Lee and Subba Rao (2012):

References


Examples

\texttt{# Plot sample paths:}

\texttt{plot(ts1(100), type="l")}

\texttt{plot(ts2(100), type="l")}

\texttt{plot(ts3(100), type="l")}
Simulation of an AR(1) time series.

Description

Returns a simulated time series \( (Y_t) \) that fulfills the following equation:

\[
Y_t = a Y_{t-1} + \epsilon_t,
\]

where \( a \) is a parameter and \( \epsilon_t \) is independent white noise with marginal distribution specified by the parameter \( \text{innov} \).

Usage

\( \text{AR1}(n, a, \text{overhead} = 500, \text{innov} = \text{rnorm}) \)

Arguments

- \( n \): length of the time series to be returned
- \( a \): parameter of the model
- \( \text{overhead} \): an integer specifying the “warmup” period to reach an approximate stationary start for the times series
- \( \text{innov} \): a function that generates a random number each time \( \text{innov}(1) \) is called; used to specify the distribution of the innovations; \( \text{rnorm} \) by default

Value

Returns an AR(1) time series with specified parameters.

Examples

\( \text{plot}(\text{AR1}(100, a=-0.7), \text{type}="l") \)

Simulation of an AR(2) time series.

Description

Returns a simulated time series \( (Y_t) \) that fulfills the following equation:

\[
Y_t = a_1 Y_{t-1} + a_2 Y_{t-2} + \epsilon_t,
\]

where \( a_1 \) and \( a_2 \) are parameters and \( \epsilon_t \) is independent white noise with marginal distribution specified by the parameter \( \text{innov} \).
Usage

Usage

AR2(n, a1, a2, overhead = 500, innov = rnorm)

Arguments

Arguments

n  length of the time series to be returned
a1 parameter
a2 parameter
overhead an integer specifying the "warmup" period to reach an approximate stationary
start for the times series
innov a function with one parameter n that yields n independent pseudo random numbers each time it is called.

Value

Value

Return an AR(2) time series with specified parameters.

Examples

Examples

plot_AR2(100, a1=0, a2=0.5), type="l"

Description

Description

Returns a simulated time series \( Y_t \) that fulfills the following equation:

\[ Y_t = Z_t \sigma_t, \quad \sigma_t^2 = a_0 + a_1 Y_{t-1}^2 + \epsilon_t \]

where \( a_0 \) and \( a_1 \) are parameters and \( \epsilon_t \) is independent white noise with marginal distribution specified by the parameter innov.

Usage

Usage

ARCH1(n, a0, a1, overhead = 500, innov = rnorm)

Arguments

Arguments

n  length of the time series to be returned
a0 parameter
a1 parameter
overhead an integer specifying the "warmup" period to reach an approximate stationary
start for the times series
innov a function with one parameter n that yields n independent pseudo random numbers each time it is called.
Value

Return an ARCH(1) time series with specified parameters.

Examples

plot(Arch1(100, a0=1/1.9, a1=0.9), type="l")

Description

Returns a simulated time series \( Y_t \) that fulfills the following equation:

\[
Y_t = \theta_1(U_t)Y_{t-1} + \theta_0(U_t),
\]

where \( \theta_1 \) and \( \theta_0 \) are parameters and \( U_t \) is independent white noise with uniform \([0,1]\) marginal distributions.

Usage

QAR1(n, th1 = function(u) { 1.9 * ((u - 0.5)) }, overhead = 1000, th0 = qnorm)

Arguments

- \( n \) length of the time series to be returned
- \( \text{th1} \) parameter function with one argument \( u \) defined on \([0,1]\)
- overhead an integer specifying the “warmup” period to reach an approximate stationary start for the times series
- \( \text{th0} \) parameter function with one argument \( u \) defined on \([0,1]\)

Value

Returns an QAR(1) time series with specified parameters.

Examples

plot(QAR1(100), type="l")
Weights is an S4 class that provides a common interface to implementations of a weighting function $W_n(\omega)$.

Currently two implementations are available: (1) \texttt{KernelWeight} and (2) \texttt{SpecDistrWeight}.

Values: an array containing the weights.

Description: Interface Class to access different types of weighting functions.

descr: a description to be used in some plots.
Index

*Topic **Access-association-functions**
  getBootPos-FreqRep, 17
  getFreqRep-QuantilePG, 19
  getQuantilePG-QuantileSD, 25
  getQuantilePG-SmoothedPG, 26
  getQuantileSD-IntegrQuantileSD, 26
  getWeight-SmoothedPG, 37

*Topic **Access-functions**
  getB-FreqRep, 17
  getBw-KernelWeight, 18
  getDescr-Weight, 18
  getFrequencies-FreqRep, 19
  getFrequencies-QSpecQuantity, 20
  getIsRankBased-FreqRep, 20
  getLevels-FreqRep, 21
  getLevels-QSpecQuantity, 21
  getMeanPG-QuantileSD, 22
  getN-QuantileSD, 22
  getParallel-QRegEstimator, 23
  getPointwiseCIs-SmoothedPG, 23
  getR-QuantileSD, 27
  getSdBoot-SmoothedPG, 27
  getSdNaive-SmoothedPG, 28
  getStdError-QuantileSD, 29
  getTs-QuantileSD, 30
  getType-QuantileSD, 30
  getValues-FreqRep, 31
  getValues-IntegrQuantileSD, 32
  getValues-KernelWeight, 32
  getValues-QuantilePG, 33
  getValues-QuantileSD, 34
  getValues-SmoothedPG, 35
  getW-KernelWeight, 36
  getWnj-KernelWeight, 37

*Topic **Constructors**
  ClippedFT-constructor, 8
  IntegrQuantileSD-constructor, 40
  KernelWeight-constructor, 44
  MovingBlocks-constructor, 45
  QRegEstimator-constructor, 53
  QuantilePG-constructor, 56
  QuantileSD-constructor, 59
  SmoothedPG-constructor, 62
  SpecDistrWeight-constructor, 64

*Topic **Defunct**
  qantspec-defunct, 60

*Topic **S4-classes**
  BootPos-class, 7
  ClippedFT-class, 8
  FreqRep-class, 11
  IntegrQuantileSD-class, 39
  KernelWeight-class, 43
  MovingBlocks-class, 45
  QRegEstimator-class, 52
  QSpecQuantity-class, 54
  QuantilePG-class, 55
  QuantileSD-class, 57
  SmoothedPG-class, 62
  SpecDistrWeight-class, 64
  Weight-class, 70

*Topic **Validator-functions**
  frequenciesValidator, 13
  timeSeriesValidator, 65

*Topic **data**
  data-sp500, 10
  data-wheatprices, 11

*Topic **internals**
  is.wholenumber, 41

AR1 (ts-models-AR1), 67
AR2 (ts-models-AR2), 67
ARCH1 (ts-models-ARCH1), 68

BootPos, 12, 17, 45
BootPos (BootPos-class), 7
BootPos-class, 7

ClippedFT, 8, 9, 11, 55, 57, 63
ClippedFT (ClippedFT-class), 8
clippedFT (ClippedFT-constructor), 8
ClippedFT-class, 8
ClippedFT-­constructor, 8
closest.pos, 9, 23, 27, 28, 31, 32, 34, 35
convolve, 62
ct (Quantspec-defunct), 60

data-sp500, 10
data-wheatprices, 11

fft, 8
FreqRep, 8, 9, 17, 19–21, 38, 46, 52, 53, 55, 57, 63
FreqRep (FreqRep-class), 11
FreqRep-class, 11
frequenciesValidator, 12, 13, 31, 33

generics-accessors, 14
generics-associations, 15
generics-functions, 16
getB (generics-accessors), 14
getB, FreqRep-method (getB-FreqRep), 17
getB-FreqRep, 17
getBootPos (generics-associations), 15
getBootPos, FreqRep-method
 (getBootPos-FreqRep), 17
getBootPos-FreqRep, 17
getBw (generics-accessors), 14
getBw, KernelWeight-method
 (getBw-KernelWeight), 18
getBw-KernelWeight, 18
getDescr (generics-accessors), 14
getDescr, Weight-method
 (getDescr-Weight), 18
getDescr-Weight, 18
getFreqRep (generics-associations), 15
getFreqRep, QuantilePG-method
 (getFreqRep-QuantilePG), 19
getFreqRep-QuantilePG, 19
getFrequencies (generics-accessors), 14
getFrequencies, FreqRep-method
 (getFrequencies-FreqRep), 19
getFrequencies, QSpecQuantity-method
 (getFrequencies-QSpecQuantity), 20
getFrequencies-FreqRep, 19
getFrequencies-QSpecQuantity, 20
getIsRankBased (generics-accessors), 14

getIsRankBased, FreqRep-method
 (getIsRankBased-FreqRep), 20
getIsRankBased-FreqRep, 20
getLevels (generics-accessors), 14
getLevels, FreqRep-method
 (getLevels-FreqRep), 21
getLevels-FreqRep, 21
getLevels, QSpecQuantity-method
 (getLevels-QSpecQuantity), 21
getLevels-FreqRep, 21
getLevels-QSpecQuantity, 21
getMeanPG (generics-accessors), 14
getMeanPG, QuantileSD-method
 (getMeanPG-QuantileSD), 22
getMeanPG-QuantileSD, 22
getN (generics-accessors), 14
getN, QuantileSD-method
 (getN-QuantileSD), 22
getN-QuantileSD, 22
getParallel (generics-accessors), 14
getParallel, QRegEstimator-method
 (getParallel-QRegEstimator), 23
getParallel-QRegEstimator, 23
getPointwiseCIs (generics-accessors), 14
getPointwiseCIs, SmoothedPG-method
 (getPointwiseCIs-SmoothedPG), 23
getPointwiseCIs-SmoothedPG, 23
getPositions (generics-functions), 16
getPositions, MovingBlocks-method
 (getPositions-MovingBlocks), 24
getPositions-MovingBlocks, 24
getQuantilePG (generics-accessors), 15
getQuantilePG, QuantileSD-method
 (getQuantilePG-QuantileSD), 25
getQuantilePG-QuantileSD, 25
getQuantilePG-SmoothedPG-method
 (getQuantilePG-SmoothedPG), 26
getQuantilePG-SmoothedPG, 26
getQuantileSD (generics-accessors), 15
getQuantileSD, IntegrQuantileSD-method
 (getQuantileSD-IntegrQuantileSD), 26
getQuantileSD-IntegrQuantileSD, 26
getR (generics-accessors), 14
getR, QuantileSD-method
(getR-QuantileSD), 27
getR-QuantileSD, 27
getSdBoot, 24
getSdBoot (generics-accessors), 14
getSdBoot, SmoothedPG-method (getSdBoot-SmoothedPG), 27
getSdBoot-SmoothedPG, 27
getSdNaive, 24
getSdNaive (generics-accessors), 14
getSdNaive, SmoothedPG-method (getSdNaive-SmoothedPG), 28
getSdNaive-SmoothedPG, 28
getStdError (generics-accessors), 14
getStdError, QuantileSD-method (getStdError-QuantileSD), 29
getStdError-QuantileSD, 29
getTs (generics-accessors), 14
getTs, QuantileSD-method (getTs-QuantileSD), 30
getTs-QuantileSD, 30
gtype (generics-accessors), 14
gtype, QuantileSD-method (gtype-QuantileSD), 30
gtype-QuantileSD, 30
gValues, 33
gValues (generics-accessors), 14
gValues, FreqRep-method (gValues-FreqRep), 31
gValues, IntegrQuantileSD-method (gValues-IntegrQuantileSD), 32
gValues, KernelWeight-method (gValues-KernelWeight), 32
gValues, QuantilePG-method (gValues-QuantilePG), 33
gValues, QuantileSD-method (gValues-QuantileSD), 34
gValues, SmoothedPG-method (gValues-SmoothedPG), 35
gValues, SpecDistrWeight-method (gValues-SpecDistrWeight), 36
gValues-FreqRep, 31
gValues-IntegrQuantileSD, 32
gValues-KernelWeight, 32
gValues-QuantilePG, 33
gValues-QuantileSD, 34
gValues-SmoothedPG, 35
gValues-SpecDistrWeight, 36
gW (generics-accessors), 14
gW, KernelWeight-method (gW-KernelWeight), 36
gW-KernelWeight, 36
getWeight (generics-associations), 15
gWeight, SmoothedPG-method (gWeight-SmoothedPG), 37
gWeight-SmoothedPG, 37
gWnj (generics-accessors), 14
gWnj, KernelWeight-method (gWnj-KernelWeight), 37
gWnj-KernelWeight, 37
gY (generics-accessors), 14
gY, FreqRep-method (gY-FreqRep), 38
gY-FreqRep, 38
increasePrecision (generics-functions), 16
increasePrecision, QuantileSD-method (increasePrecision-QuantileSD), 38
increasePrecision-QuantileSD, 38
Integer, 41
IntegrQuantileSD, 26, 32, 40, 41, 46, 47, 54
IntegrQuantileSD (IntegrQuantileSD-class), 39
IntegrQuantileSD (IntegrQuantileSD-constructor), 40
IntegrQuantileSD-class, 39
IntegrQuantileSD-constructor, 40
is.wholenumber, 41
kernels, 42, 43, 44
KernelWeight, 44, 47, 48, 70
KernelWeight (KernelWeight-class), 43
KernelWeight (KernelWeight-constructor), 44
KernelWeight-class, 43
KernelWeight-constructor, 44
LaplacePeriodogram (quantspec-defunct), 60
layout, 47–49, 51
MovingBlocks, 7, 45
MovingBlocks (MovingBlocks-class), 45
MovingBlocks (MovingBlocks-constructor), 45
INDEX

MovingBlocks-class, 45
MovingBlocks-constructor, 45

mvfft, 8

plot,FreqRep,ANY-method (plot-FreqRep), 46
plot,IntegrQuantileSD,ANY-method (plot-IntegrQuantileSD), 46
plot,KernelWeight,missing-method (plot-KernelWeight), 47
plot,QuantilePG,ANY-method (plot-QuantilePG), 48
plot,QuantileSD,ANY-method (plot-QuantileSD), 49
plot,SmoothedPG,ANY-method (plot-SmoothedPG), 50
plot,SpecDistrWeight,missing-method (plot-SpecDistrWeight), 51
plot-FreqRep, 46
plot-IntegrQuantileSD, 46
plot-KernelWeight, 47
plot-QuantilePG, 48
plot-QuantileSD, 49
plot-SmoothedPG, 50
plot-SpecDistrWeight, 51
plotLaplacePeriodogram (quantspec-defunct), 60

QAR1 (ts-models-QAR1), 69
QRegEstimator, 11, 23, 55, 57, 63
QRegEstimator (QRegEstimator-class), 52
qRegEstimator (QRegEstimator-constructor), 53
QRegEstimator-class, 52
QRegEstimator-constructor, 53
QSpecQuantity, 39, 55, 58
QSpecQuantity (QSpecQuantity-class), 54
QSpecQuantity-class, 54
QuantilePG, 19, 25, 26, 38, 48, 54, 56, 62, 63
QuantilePG (QuantilePG-class), 55
quantilePG (QuantilePG-constructor), 56
QuantilePG-class, 55
QuantilePG-constructor, 56
QuantileSD, 25, 35, 38-40, 48-50, 54, 59, 60
QuantileSD (QuantileSD-class), 57
quantileSD (QuantileSD-constructor), 59
QuantileSD-class, 57
QuantileSD-constructor, 59
quantspec (quantspec-package), 3
quantspec-defunct, 60
quantspec-package, 3

rq, 52
smooth.spline, 58
smoothedLaplacePeriodogram (quantspec-defunct), 60
SmoothedPG, 26-29, 37, 50, 54
SmoothedPG (SmoothedPG-class), 62
SmoothedPG (SmoothedPG-constructor), 62
SmoothedPG-class, 62
SmoothedPG-constructor, 62
sp500 (data-sp500), 10
SpecDistrWeight, 36, 51, 52, 64, 70
SpecDistrWeight (SpecDistrWeight-class), 64
SpecDistrWeight (SpecDistrWeight-constructor), 64
SpecDistrWeight-class, 64
SpecDistrWeight-constructor, 64
timeSeriesValidator, 65
ts-models, 66
ts-models-AR1, 67
ts-models-AR2, 67
ts-models-ARCH1, 68
ts-models-QAR1, 69
ts1 (ts-models), 66
ts2 (ts-models), 66
ts3 (ts-models), 66

W0 (kernels), 42
W1 (kernels), 42
W2 (kernels), 42
W3 (kernels), 42
WDanieli (kernels), 42
Weight, 37, 43, 62, 63
Weight (Weight-class), 70
Weight-class, 70
wheatprices (data-wheatprices), 11