Package 'qfa'

December 16, 2024

Type Package

Title Quantile-Frequency Analysis (QFA) of Time Series

Version 3.1

Date 2024-12-16

Maintainer Ta-Hsin Li <thl024@outlook.com>

Description

Quantile-frequency analysis (QFA) of time series based on trigonometric quantile regression.
References:

[1] Li, T.-H. (2012) ``Quantile periodograms", Journal of the American Statistical Association, 107, 765–776, <doi:10.1080/01621459.2012.682815>.

[2] Li, T.-H. (2014) Time Series with Mixed Spectra, CRC Press, <doi:10.1201/b15154>

[3] Li, T.-H. (2022) ``Quantile Fourier transform, quantile series, and nonparametric estimation of quantile spectra", <doi:10.48550/arXiv.2211.05844>.

[4] Li, T.-H. (2024) ``Quantile crossing spectrum and spline autoregression estimation," <doi:10.48550/arXiv.2412.02513>.

Depends R (>= 3.5)

Imports RhpcBLASctl,
doParallel,
fields,
foreach,
mgcv,
nlme,
parallel,
quantreg,
splines,
stats,
graphics,
colorRamps,
MASS

License GPL (>=2)

URL https://github.com/IBM/qfa, https://github.com/thl2019/QFA

NeedsCompilation yes

Encoding UTF-8

RoxygenNote 7.3.2

Contents

`per` *Periodogram (PER)*

Description

This function computes the periodogram or periodogram matrix for univariate or multivariate time series.

Usage

```
per(y)
```

Arguments

`y`	vector (n) or matrix (n x nc) of time series

Value

A vector (n) or array (nc x nc x n) of periodogram

Examples

```
y <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y.per <- per(y)
plot(y.per)
```

---

qacf *Quantile Autocovariance Function (QACF)*

Description

This function computes quantile autocovariance function (QACF) from time series or quantile discrete Fourier transform (QDFT).

Usage

```
qacf(y, tau, y.qdft = NULL, n.cores = 1, cl = NULL)
```

Arguments

y	vector or matrix of time series (if matrix, nrow(y) = length of time series)
tau	sequence of quantile levels in (0,1)
y.qdft	matrix or array of pre-calculated QDFT (default = NULL: compute from y and tau); if y.qdft is supplied, y and tau can be left unspecified
n.cores	number of cores for parallel computing of QDFT if y.qdft = NULL (default = 1)
cl	pre-existing cluster for repeated parallel computing of QDFT (default = NULL)

Value

matrix or array of quantile autocovariance function

Examples

```
y <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
# compute from time series
y.qacf <- qacf(y,tau)
# compute from QDFT
y.qdft <- qdft(y,tau)
y.qacf <- qacf(y.qdft=y.qdft)
```

qcser *Quantile-Crossing Series (QCSER)*

Description

This function creates the quantile-crossing series (QCSER) for univariate or multivariate time series.

Usage

```
qcser(y, tau, normalize = FALSE)
```

Arguments

y	vector or matrix of time series
tau	vector of quantile levels in (0,1)
normalize	TRUE or FALSE (default): normalize QCSER to have unit variance

Value

A matrix or array of quantile-crossing series

Examples

```
y <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y.qser <- qcser(y,tau)
dim(y.qser)
```

qdft *Quantile Discrete Fourier Transform (QDFT)*

Description

This function computes quantile discrete Fourier transform (QDFT) for univariate or multivariate time series.

Usage

```
qdft(y, tau, n.cores = 1, cl = NULL)
```

Arguments

y	vector or matrix of time series (if matrix, nrow(y) = length of time series)
tau	sequence of quantile levels in (0,1)
n.cores	number of cores for parallel computing (default = 1)
cl	pre-existing cluster for repeated parallel computing (default = NULL)

Value

matrix or array of quantile discrete Fourier transform of y

Examples

```
y <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y.qdft <- qdft(y,tau)
# Make a cluster for repeated use
n.cores <- 2
cl <- parallel::makeCluster(n.cores)
parallel::clusterExport(cl, c("tqr.fit"))
doParallel::registerDoParallel(cl)
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y.qdft <- qdft(y1,tau,n.cores=n.cores,cl=cl)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y.qdft <- qdft(y2,tau,n.cores=n.cores,cl=cl)
parallel::stopCluster(cl)
```

---

qdft2qacf *Quantile Autocovariance Function (QACF)*

Description

This function computes quantile autocovariance function (QACF) from QDFT.

Usage

```
qdft2qacf(y.qdft, return.qser = FALSE)
```

Arguments

`y.qdft`	matrix or array of QDFT from `qdft()`
`return.qser`	if `TRUE`, return quantile series (QSER) along with QACF

Value

matrix or array of quantile autocovariance function if `return.sqer = FALSE` (default), else a list with the following elements:

`qacf`	matirx or array of quantile autocovariance function
`qser`	matrix or array of quantile series

Examples

```
# single time series
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y.qdft <- qdft(y1,tau)
y.qacf <- qdft2qacf(y.qdft)
plot(c(0:9),y.qacf[c(1:10),1],type='h',xlab="LAG",ylab="QACF")
y.qser <- qdft2qacf(y.qdft,return.qser=TRUE)$qser
plot(y.qser[,1],type='l',xlab="TIME",ylab="QSER")
# multiple time series
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
y.qdft <- qdft(cbind(y1,y2),tau)
y.qacf <- qdft2qacf(y.qdft)
plot(c(0:9),y.qacf[1,2,c(1:10),1],type='h',xlab="LAG",ylab="QACF")
```

qdft2qper *Quantile Periodogram (QPER)*

Description

This function computes quantile periodogram (QPER) from QDFT.

Usage

```
qdft2qper(y.qdft)
```

Arguments

y.qdft	matrix or array of QDFT from qdft()

Value

matrix or array of quantile periodogram

Examples

```
# single time series
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y.qdft <- qdft(y1,tau)
y.qper <- qdft2qper(y.qdft)
n <- length(y1)
ff <- c(0:(n-1))/n
sel.f <- which(ff > 0 & ff < 0.5)
qfa.plot(ff[sel.f],tau,Re(y.qper[sel.f,]))
# multiple time series
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
y.qdft <- qdft(cbind(y1,y2),tau)
y.qper <- qdft2qper(y.qdft)
qfa.plot(ff[sel.f],tau,Re(y.qper[1,1,sel.f,]))
qfa.plot(ff[sel.f],tau,Re(y.qper[1,2,sel.f,]))
```

qdft2qser *Quantile Series (QSER)*

Description

This function computes quantile series (QSER) from QDFT.

Usage

```
qdft2qser(y.qdft)
```

Arguments

y.qdft	matrix or array of QDFT from qdft()

Value

matrix or array of quantile series

Examples

```
# single time series
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y.qdft <- qdft(y1,tau)
y.qser <- qdft2qser(y.qdft)
plot(y.qser[,1],type='l',xlab="TIME",ylab="QSER")
# multiple time series
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
y.qdft <- qdft(cbind(y1,y2),tau)
y.qser <- qdft2qser(y.qdft)
plot(y.qser[1,,1],type='l',xlab="TIME",ylab="QSER")
```

---

qfa.plot *Quantile-Frequency Plot*

Description

This function creates an image plot of quantile spectrum.

Usage

```
qfa.plot(
  freq,
  tau,
  rqper,
  rg.qper = range(rqper),
  rg.tau = range(tau),
  rg.freq = c(0, 0.5),
  color = colorRamps::matlab.like2(1024),
  ylab = "QUANTILE LEVEL",
  xlab = "FREQUENCY",
  tlab = NULL,
  set.par = TRUE,
  legend.plot = TRUE
)
```

Arguments

freq	sequence of frequencies in (0,0.5) at which quantile spectrum is evaluated
tau	sequence of quantile levels in (0,1) at which quantile spectrum is evaluated
rqper	real-valued matrix of quantile spectrum evaluated on the freq x tau grid
rg.qper	zlim for qper (default = range(qper))
rg.tau	ylim for tau (default = range(tau))
rg.freq	xlim for freq (default = c(0,0.5))
color	colors (default = colorRamps::matlab.like2(1024))

ylab	label of y-axis (default = "QUANTILE LEVEL")
xlab	label of x-axis (default = "FREQUENCY")
tlab	title of plot (default = NULL)
set.par	if TRUE, par() is set internally (single image)
legend.plot	if TRUE, legend plot is added

Value

no return value

---

qkl.divergence *Kullback-Leibler Divergence of Quantile Spectral Estimate*

Description

This function computes Kullback-Leibler divergence (KLD) of quantile spectral estimate.

Usage

```
qkl.divergence(y.qper, qspec, sel.f = NULL, sel.tau = NULL)
```

Arguments

y.qper	matrix or array of quantile spectral estimate from, e.g., qspec.lw()
qspec	matrix of array of true quantile spectrum (same dimension as y.qper)
sel.f	index of selected frequencies for computation (default = NULL: all frequencies)
sel.tau	index of selected quantile levels for computation (default = NULL: all quantile levels)

Value

real number of Kullback-Leibler divergence

---

qper *Quantile Periodogram (QPER)*

Description

This function computes quantile periodogram (QPER) from time series or quantile discrete Fourier transform (QDFT).

Usage

```
qper(y, tau, y.qdft = NULL, n.cores = 1, cl = NULL)
```

Arguments

y	vector or matrix of time series (if matrix, nrow(y) = length of time series)
tau	sequence of quantile levels in (0,1)
y.qdft	matrix or array of pre-calculated QDFT (default = NULL: compute from y and tau); if y.qdft is supplied, y and tau can be left unspecified
n.cores	number of cores for parallel computing of QDFT if y.qdft = NULL (default = 1)
cl	pre-existing cluster for repeated parallel computing of QDFT (default = NULL)

Value

matrix or array of quantile periodogram

Examples

```
y <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
# compute from time series
y.qper <- qper(y,tau)
# compute from QDFT
y.qdft <- qdft(y,tau)
y.qper <- qper(y.qdft=y.qdft)
```

---

qper2 *Quantile Periodogram Type II (QPER2)*

Description

This function computes type-II quantile periodogram for univariate time series.

Usage

```
qper2(y, freq, tau, weights = NULL, n.cores = 1, cl = NULL)
```

Arguments

y	univariate time series
freq	sequence of frequencies in [0,1)
tau	sequence of quantile levels in (0,1)
weights	sequence of weights in quantile regression (default = NULL: weights equal to 1)
n.cores	number of cores for parallel computing (default = 1)
cl	pre-existing cluster for repeated parallel computing (default = NULL)

Value

matrix of quantile periodogram evaluated on freq * tau grid

Examples

```
y <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
n <- length(y)
ff <- c(0:(n-1))/n
sel.f <- which(ff > 0 & ff < 0.5)
y.qper2 <- qper2(y,ff,tau)
qfa.plot(ff[sel.f],tau,Re(y.qper2[sel.f,]))
```

---

qser *Quantile Series (QSER)*

Description

This function computes quantile series (QSER) from time series or quantile discrete Fourier transform (QDFT).

Usage

```
qser(y, tau, y.qdft = NULL, n.cores = 1, cl = NULL)
```

Arguments

y	vector or matrix of time series (if matrix, nrow(y) = length of time series)
tau	sequence of quantile levels in (0,1)
y.qdft	matrix or array of pre-calculated QDFT (default = NULL: compute from y and tau); if y.qdft is supplied, y and tau can be left unspecified
n.cores	number of cores for parallel computing of QDFT if y.qdft = NULL (default = 1)
cl	pre-existing cluster for repeated parallel computing of QDFT (default = NULL)

Value

matrix or array of quantile series

Examples

```
y <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
# compute from time series
y.qser <- qser(y,tau)
# compute from QDFT
y.qdft <- qdft(y,tau)
y.qser <- qser(y.qdft=y.qdft)
```

qser2ar *Autoregression (AR) Model of Quantile Series*

Description

This function fits an autoregression (AR) model to quantile series (QSER) separately for each quantile level using `stats::ar()`.

Usage

```
qser2ar(y.qser, p = NULL, order.max = NULL, method = c("none", "gamm", "sp"))
```

Arguments

`y.qser`	matrix or array of pre-calculated QSER, e.g., using `qser()`
`p`	order of AR model (default = `NULL`: selected by AIC)
`order.max`	maximum order for AIC if `p` = `NULL` (default = `NULL`: determined by `stats::ar()`)
`method`	quantile smoothing method: `"gamm"`, `"sp"`, or `"NA"` (default)

Value

a list with the following elements:

`A`	matrix or array of AR coefficients
`V`	vector or matrix of residual covariance
`p`	order of AR model
`n`	length of time series
`residuals`	matrix or array of residuals

qser2qacf *ACF of Quantile Series (QSER) or Quantile-Crossing Series (QCACF)*

Description

This function creates the ACF of quantile series or quantile-crossing series

Usage

```
qser2qacf(y.qser)
```

Arguments

`y.qser`	matrix or array of quantile-crossing series

Value

A matrix or array of ACF

Examples

```
y <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y.qser <- qcser(y,tau)
y.qacf <- qser2qacf(y.qser)
dim(y.qacf)
```

`qser2sar` *Spline Autoregression (SAR) Model of Quantile Series*

Description

This function fits spline autoregression (SAR) model to quantile series (QSER).

Usage

```
qser2sar(
  y.qser,
  tau,
  d = 1,
  p = NULL,
  order.max = NULL,
  spar = NULL,
  method = c("GCV", "AIC", "BIC"),
  weighted = FALSE
)
```

Arguments

`y.qser`	matrix or array of pre-calculated QSER, e.g., using `qser()`
`tau`	sequence of quantile levels where `y.qser` is calculated
`d`	subsampling rate of quantile levels (default = 1)
`p`	order of SAR model (default = `NULL`: automatically selected by AIC)
`order.max`	maximum order for AIC if `p` = `NULL` (default = `NULL`: determined by `stats::ar()`)
`spar`	penalty parameter alla `smooth.spline` (default = `NULL`: automatically selected)
`method`	criterion for penalty parameter selection: `"AIC"` (default), `"BIC"`, or `"GCV"`
`weighted`	if `TRUE`, penalty function is weighted (default = `FALSE`)

Value

a list with the following elements:

`A`	matrix or array of SAR coefficients
`V`	vector or matrix of SAR residual covariance
`p`	order of SAR model
`spar`	penalty parameter
`tau`	sequence of quantile levels
`n`	length of time series

d	subsampling rate of quantile levels
weighted	option for weighted penalty function
fit	object containing details of SAR fit

---

qspec.ar *Autoregression (AR) Estimator of Quantile Spectrum*

Description

This function computes autoregression (AR) estimate of quantile spectrum from time series or quantile series (QSER).

Usage

```
qspec.ar(
  y,
  tau,
  y.qser = NULL,
  p = NULL,
  order.max = NULL,
  freq = NULL,
  method = c("none", "gamm", "sp"),
  n.cores = 1,
  cl = NULL
)
```

Arguments

y	vector or matrix of time series (if matrix, nrow(y) = length of time series)
tau	sequence of quantile levels in (0,1)
y.qser	matrix or array of pre-calculated QSER (default = NULL: compute from y and tau);
p	order of AR model (default = NULL: automatically selected by AIC)
order.max	maximum order for AIC if p = NULL (default = NULL: determined by stats::ar())
freq	sequence of frequencies in [0,1) (default = NULL: all Fourier frequencies)
method	quantile smoothing method: "gamm" for mgcv::gamm(), "sp" for stats::smooth.spline(), or "none" (default) if y.qser is supplied, y and tau can be left unspecified
n.cores	number of cores for parallel computing of QDFT if y.qser = NULL (default = 1)
cl	pre-existing cluster for repeated parallel computing of QDFT (default = NULL)

Value

a list with the following elements:

spec	matrix or array of AR quantile spectrum
freq	sequence of frequencies
fit	object of AR model
qser	matrix or array of quantile series if y.qser = NULL

Examples

```
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
y <- cbind(y1,y2)
tau <- seq(0.1,0.9,0.05)
n <- length(y1)
ff <- c(0:(n-1))/n
sel.f <- which(ff > 0 & ff < 0.5)
y.qspec.ar <- qspec.ar(y,tau,p=1)$spec
qfa.plot(ff[sel.f],tau,Re(y.qspec.ar[1,1,sel.f,]))
y.qser <- qcser(y1,tau)
y.qspec.ar <- qspec.ar(y.qser=y.qser,p=1)$spec
qfa.plot(ff[sel.f],tau,Re(y.qspec.ar[sel.f,]))
y.qspec.arqs <- qspec.ar(y.qser=y.qser,p=1,method="sp")$spec
qfa.plot(ff[sel.f],tau,Re(y.qspec.arqs[sel.f,]))
```

---

qspec.lw *Lag-Window (LW) Estimator of Quantile Spectrum*

Description

This function computes lag-window (LW) estimate of quantile spectrum with or without quantile smoothing from time series or quantile autocovariance function (QACF).

Usage

```
qspec.lw(
  y,
  tau,
  y.qacf = NULL,
  M = NULL,
  method = c("none", "gamm", "sp"),
  spar = "GCV",
  n.cores = 1,
  cl = NULL
)
```

Arguments

y	vector or matrix of time series (if matrix, nrow(y) = length of time series)
tau	sequence of quantile levels in (0,1)
y.qacf	matrix or array of pre-calculated QACF (default = NULL: compute from y and tau); if y.qacf is supplied, y and tau can be left unspecified
M	bandwidth parameter of lag window (default = NULL: quantile periodogram)
method	quantile smoothing method: "gamm" for mgcv::gamm(), "sp" for stats::smooth.spline(), or "none" (default)
spar	smoothing parameter in smooth.spline() if method = "sp" (default = "GCV")
n.cores	number of cores for parallel computing (default = 1)
cl	pre-existing cluster for repeated parallel computing (default = NULL)

Value

A list with the following elements:

spec	matrix or array of spectral estimate
spec.lw	matrix or array of spectral estimate without quantile smoothing
lw	lag-window sequence
qacf	matrix or array of quantile autocovariance function if y.qacf = NULL

Examples

```
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
tau <- seq(0.1,0.9,0.05)
n <- length(y1)
ff <- c(0:(n-1))/n
sel.f <- which(ff > 0 & ff < 0.5)
y.qacf <- qacf(cbind(y1,y2),tau)
y.qper.lw <- qspec.lw(y.qacf=y.qacf,M=5)$spec
qfa.plot(ff[sel.f],tau,Re(y.qper.lw[1,1,sel.f,]))
y.qper.lwqs <- qspec.lw(y.qacf=y.qacf,M=5,method="sp",spar=0.9)$spec
qfa.plot(ff[sel.f],tau,Re(y.qper.lwqs[1,1,sel.f,]))
```

---

qspec.sar *Spline Autoregression (SAR) Estimator of Quantile Spectrum*

Description

This function computes spline autoregression (SAR) estimate of quantile spectrum.

Usage

```
qspec.sar(
  y,
  y.qser = NULL,
  tau,
  d = 1,
  p = NULL,
  order.max = NULL,
  spar = NULL,
  method = c("GCV", "AIC", "BIC"),
  weighted = FALSE,
  freq = NULL,
  n.cores = 1,
  cl = NULL
)
```

Arguments

y	vector or matrix of time series (if matrix, nrow(y) = length of time series)
y.qser	matrix or array of pre-calculated QSER (default = NULL: compute from y and tau); if y.qser is supplied, y can be left unspecified
tau	sequence of quantile levels in (0,1)
d	subsampling rate of quantile levels (default = 1)
p	order of SAR model (default = NULL: automatically selected by AIC)
order.max	maximum order for AIC if p = NULL (default = NULL: determined by stats::ar())
spar	penalty parameter alla smooth.spline (default = NULL: automatically selected)
method	criterion for penalty parameter selection: "GCV", "AIC" (default), or "BIC"
weighted	if TRUE, penalty function is weighted (default = FALSE)
freq	sequence of frequencies in [0,1) (default = NULL: all Fourier frequencies)
n.cores	number of cores for parallel computing of QDFT if y.qser = NULL (default = 1)
cl	pre-existing cluster for repeated parallel computing of QDFT (default = NULL)

Value

a list with the following elements:

spec	matrix or array of SAR quantile spectrum
freq	sequence of frequencies
fit	object of SAR model
qser	matrix or array of quantile series if y.qser = NULL

Examples

```
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
tau <- seq(0.1,0.9,0.05)
n <- length(y1)
ff <- c(0:(n-1))/n
sel.f <- which(ff > 0 & ff < 0.5)
# compute from time series
y.sar <- qspec.sar(cbind(y1,y2),tau=tau,p=1)
qfa.plot(ff[sel.f],tau,Re(y.sar$spec[1,1,sel.f,]))
# compute from quantile series
y.qser <- qser(cbind(y1,y2),tau)
y.sar <- qspec.sar(y.qser=y.qser,tau=tau,p=1)
qfa.plot(ff[sel.f],tau,Re(y.sar$spec[1,1,sel.f,]))
```

qspec2qcoh *Quantile Coherence Spectrum*

Description

This function computes quantile coherence spectrum (QCOH) from quantile spectrum of multiple time series.

Usage

```
qspec2qcoh(qspec, k = 1, kk = 2)
```

Arguments

qspec	array of quantile spectrum
k	index of first series (default = 1)
kk	index of second series (default = 2)

Value

matrix of quantile coherence evaluated at Fourier frequencies in (0,0.5)

Examples

```
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
tau <- seq(0.1,0.9,0.05)
n <- length(y1)
ff <- c(0:(n-1))/n
sel.f <- which(ff > 0 & ff < 0.5)
y.qacf <- qacf(cbind(y1,y2),tau)
y.qper.lw <- qspec.lw(y.qacf=y.qacf,M=5)$spec
y.qcoh <- qspec2qcoh(y.qper.lw,k=1,kk=2)
qfa.plot(ff[sel.f],tau,y.qcoh)
```

sar.eq.bootstrap *Bootstrap Simulation of SAR Coefficients for Testing Equality of Granger-Causality in Two Samples*

Description

This function simulates bootstrap samples of selected spline autoregression (SAR) coefficients for testing equality of Granger-causality in two samples based on their SAR models under H0: effect in each sample equals the average effect.

Usage

```
sar.eq.bootstrap(
  y.qser,
  fit,
  fit2,
  index = c(1, 2),
  nsim = 1000,
  method = c("ar", "sar"),
  n.cores = 1,
  mthreads = FALSE,
  seed = 1234567
)
```

Arguments

y.qser	matrix or array of QSER from qser() or qspec.sar()$qser
fit	object of SAR model from qser2sar() or qspec.sar()$fit
fit2	object of SAR model for the other sample
index	a pair of component indices for multiple time series or a sequence of lags for single time series (default = c(1,2))
nsim	number of bootstrap samples (default = 1000)
method	method of residual calculation: "ar" (default) or "sar"
n.cores	number of cores for parallel computing (default = 1)
mthreads	if TRUE, multithread BLAS is enabled when available (default = FALSE, required for parallel computing)
seed	seed for random sampling (default = 1234567)

Value

array of simulated bootstrap samples of selected SAR coefficients

Examples

```
y11 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y21 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
y12 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y22 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y1.sar <- qspec.sar(cbind(y11,y21),tau=tau,p=1)
y2.sar <- qspec.sar(cbind(y12,y22),tau=tau,p=1)
A1.sim <- sar.eq.bootstrap(y1.sar$qser,y1.sar$fit,y2.sar$fit,index=c(1,2),nsim=5)
A2.sim <- sar.eq.bootstrap(y2.sar$qser,y2.sar$fit,y1.sar$fit,index=c(1,2),nsim=5)
```

`sar.eq.test` *Wald Test and Confidence Band for Equality of Granger-Causality in Two Samples*

Description

This function computes Wald test and confidence band for equality of Granger-causality in two samples using bootstrap samples generated by `sar.eq.bootstrap()` based on the spline autoregression (SAR) models of quantile series (QSER).

Usage

```
sar.eq.test(A1, A1.sim, A2, A2.sim, sel.lag = NULL, sel.tau = NULL)
```

Arguments

`A1`	matrix of selected SAR coefficients for sample 1
`A1.sim`	simulated bootstrap samples from `sar.eq.bootstrap()` for sample 1
`A2`	matrix of selected SAR coefficients for sample 2
`A2.sim`	simulated bootstrap samples from `sar.eq.bootstrap()` for sample 2
`sel.lag`	indices of time lags for Wald test (default = `NULL`: all lags)
`sel.tau`	indices of quantile levels for Wald test (default = `NULL`: all quantiles)

Value

a list with the following elements:

`test`	list of Wald test result containing `wald` and `p.value`
`D.u`	matrix of upper limits of 95% confidence band for `A1 - A2`
`D.l`	matrix of lower limits of 95% confidence band for `A1 - A2`

Examples

```
y11 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y21 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
y12 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y22 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y1.sar <- qspec.sar(cbind(y11,y21),tau=tau,p=1)
y2.sar <- qspec.sar(cbind(y12,y22),tau=tau,p=1)
A1.sim <- sar.eq.bootstrap(y1.sar$qser,y1.sar$fit,y2.sar$fit,index=c(1,2),nsim=5)
A2.sim <- sar.eq.bootstrap(y2.sar$qser,y2.sar$fit,y1.sar$fit,index=c(1,2),nsim=5)
A1 <- sar.gc.coef(y1.sar$fit,index=c(1,2))
A2 <- sar.gc.coef(y2.sar$fit,index=c(1,2))
test <- sar.eq.test(A1,A1.sim,A2,A2.sim,sel.lag=NULL,sel.tau=NULL)
```

`sar.gc.bootstrap` *Bootstrap Simulation of SAR Coefficients for Granger-Causality Analysis*

Description

This function simulates bootstrap samples of selected spline autoregression (SAR) coefficients for Granger-causality analysis based on the SAR model of quantile series (QSER) under H0: (a) for multiple time series, the second series specified in `index` is not causal for the first series specified in `index`; (b) for single time series, the series is not causal at the lags specified in `index`.

Usage

```
sar.gc.bootstrap(
  y.qser,
  fit,
  index = c(1, 2),
  nsim = 1000,
  method = c("ar", "sar"),
  n.cores = 1,
  mthreads = FALSE,
  seed = 1234567
)
```

Arguments

`y.qser`	matrix or array of QSER from `qser()` or `qspec.sar()$qser`
`fit`	object of SAR model from `qser2sar()` or `qspec.sar()$fit`
`index`	a pair of component indices for multiple time series or a sequence of lags for single time series (default = `c(1,2)`)
`nsim`	number of bootstrap samples (default = 1000)
`method`	method of residual calculation: `"ar"` (default) or `"sar"`
`n.cores`	number of cores for parallel computing (default = 1)
`mthreads`	if `TRUE`, multithread BLAS is enabled when available (default = `FALSE`, required for parallel computing)
`seed`	seed for random sampling (default = `1234567`)

Value

array of simulated bootstrap samples of selected SAR coefficients

Examples

```
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y.sar <- qspec.sar(cbind(y1,y2),tau=tau,p=1)
A.sim <- sar.gc.bootstrap(y.sar$qser,y.sar$fit,index=c(1,2),nsim=5)
```

sar.gc.coef *Extraction of SAR Coefficients for Granger-Causality Analysis*

Description

This function extracts the spline autoregression (SAR) coefficients from an SAR model for Granger-causality analysis. See `sar.gc.bootstrap` for more details regarding the use of `index`.

Usage

```
sar.gc.coef(fit, index = c(1, 2))
```

Arguments

`fit`	object of SAR model from `qser2sar()` or `qspec.sar()$fit`
`index`	a pair of component indices for multiple time series or a sequence of lags for single time series (default = `c(1,2)`)

Value

matrix of selected SAR coefficients (number of lags by number of quantiles)

Examples

```
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y.sar <- qspec.sar(cbind(y1,y2),tau=tau,p=1)
A <- sar.gc.coef(y.sar$fit,index=c(1,2))
```

sar.gc.test *Wald Test and Confidence Band for Granger-Causality Analysis*

Description

This function computes Wald test and confidence band for Granger-causality using bootstrap samples generated by `sar.gc.bootstrap()` based the spline autoregression (SAR) model of quantile series (QSER).

Usage

```
sar.gc.test(A, A.sim, sel.lag = NULL, sel.tau = NULL)
```

Arguments

`A`	matrix of selected SAR coefficients
`A.sim`	simulated bootstrap samples from `sar.gc.bootstrap()`
`sel.lag`	indices of time lags for Wald test (default = `NULL`: all lags)
`sel.tau`	indices of quantile levels for Wald test (default = `NULL`: all quantiles)

Value

a list with the following elements:

test	list of Wald test result containing wald and p.value
A.u	matrix of upper limits of 95% confidence band of A
A.l	matrix of lower limits of 95% confidence band of A

Examples

```
y1 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
tau <- seq(0.1,0.9,0.05)
y.sar <- qspec.sar(cbind(y1,y2),tau=tau,p=1)
A <- sar.gc.coef(y.sar$fit,index=c(1,2))
A.sim <- sar.gc.bootstrap(y.sar$qser,y.sar$fit,index=c(1,2),nsim=5)
y.gc <- sar.gc.test(A,A.sim)
```

---

sqr.fit *Spline Quantile Regression (SQR)*

Description

This function computes spline quantile regression (SQR) solution from response vector and design matrix. It uses the FORTRAN code rqfnb.f in the "quantreg" package with the kind permission of Dr. R. Koenker.

Usage

```
sqr.fit(y, X, tau, c0, d = 1, weighted = FALSE, mthreads = FALSE)
```

Arguments

y	response vector
X	design matrix (nrow(X) = length(y))
tau	sequence of quantile levels in (0,1)
c0	penalty parameter
d	subsampling rate of quantile levels (default = 1)
weighted	if TRUE, penalty function is weighted (default = FALSE)
mthreads	if TRUE, multithread BLAS is enabled when available (default = FALSE, required for parallel computing)

Value

A list with the following elements:

coefficients	matrix of regression coefficients
nit	number of iterations

`tqr.fit` *Trigonometric Quantile Regression (TQR)*

Description

This function computes trigonometric quantile regression (TQR) for univariate time series at a single frequency.

Usage

```
tqr.fit(y, f0, tau, prepared = TRUE)
```

Arguments

`y`	vector of time series
`f0`	frequency in [0,1)
`tau`	sequence of quantile levels in (0,1)
`prepared`	if `TRUE`, intercept is removed and coef of cosine is doubled when `f0 = 0.5`

Value

object of `rq()` (coefficients in `$coef`)

Examples

```
y <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=64)
tau <- seq(0.1,0.9,0.05)
fit <- tqr.fit(y,f0=0.1,tau=tau)
plot(tau,fit$coef[1,],type='o',pch=0.75,xlab='QUANTILE LEVEL',ylab='TQR COEF')
```

Index