Package ‘qfa’

November 26, 2024

Type Package

Title Quantile-Frequency Analysis (QFA) of Time Series

Version 3.0

Date 2024-11-26

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

Description

Quantile-frequency analysis (QFA) of univariate or multivariate time series based on trigonometric quantile regression. See Li, T.-H. (2012) ``Quantile periodograms", Journal of the American Statistical Association, 107, 765–776, <doi:10.1080/01621459.2012.682815>; Li, T.-H. (2014) Time Series with Mixed Spectra, CRC Press, <doi:10.1201/b15154>; Li, T.-H. (2022) ``Quantile Fourier transform, quantile series, and nonparametric estimation of quantile spectra", <doi:10.48550/arXiv.2211.05844>.

Depends R (>= 3.5)

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

License GPL (>=2)

URL `https://www.r-project.org`, `https://github.com/IBM/qfa`

NeedsCompilation yes

Encoding UTF-8

RoxygenNote 7.3.2

Contents

`ar2qspec` *Quantile Spectrum from AR Model of Quantile Series or Quantile-Crossing Series*

Description

This function computes quantile spectrum (QSPEC) or quantile-crossing spectrum (QCSPEC) from an AR model of quantile series (QSER) or quantile-crossing series (QCSER).

Usage

```
ar2qspec(fit, freq = NULL)
```

Arguments

`fit`	object of AR model from `qser2sar()` or `qser2ar()`
`freq`	sequence of frequencies in [0,1) (default = `NULL`: all Fourier frequencies)

Value

a list with the following elements:

`spec`	matrix or array of quantile spectrum or quantile-crossing spectrum
`freq`	sequence of frequencies

`per` *Periodogram (PER)*

Description

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

Usage

```
per(y)
```

Arguments

`y`	time series (if multivariate, `nrow(y)` = length of time series)

Value

A vector or array 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	time series (if multivariate, `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	time series (if multivariate, `nrow(y)` = length of time series)
tau	sequence 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	time series (if multivariate, 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=32)
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=32)
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=32)
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()` or SQDFT from `sqdft()`
`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 and Cross-Periodogram (QPER)*

Description

This function computes quantile periodogram/cross-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/cross-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() or SQDFT from sqdft()

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/cross-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 and Cross-Periodogram (QPER)*

Description

This function computes quantile periodogram/cross-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`	time series (if multivariate, `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/cross-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`	time series (if multivariate, `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 or Quantile-Crossing Series with or without Post-Smoothing*

Description

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

Usage

```
qser2ar(y.qser, p = NULL, order.max = NULL, method = c("NA", "gamm", "sp"))
```

Arguments

`y.qser`	matrix or array of pre-calculated QSER or QCSER using `qser()` or `qcser()`
`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`	post-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

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)
fit.ar <- qser2ar(y.qser,p=1,method="NA")
plot(tau,fit.ar$A[1,])
fit.ars <- qser2ar(y.qser,p=1,method="sp")
lines(tau,fit.ars$A[1,])
```

`qser2qacf` *ACF of Quantile Series or Quantile-Crossing Series*

Description

This function creates the ACF of quantile series (QSER) or quantile-crossing series (QCSER)

Usage

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

Arguments

`y.qser`	QSER or QCSER computed from `qser()` or `qcser()`

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 or Quantile-Crossing Series*

Description

This function fits spline autoregression (SAR) model to quantile series (QSER) or quantile-crossing series (QCSER).

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 or QCSER using `qser()` or `qcser()`
`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

qsmooth.qdft *Quantile Smoothing of Quantile Discrete Fourier Transform*

Description

This function computes quantile-smoothed version of quantile discrete Fourier transform (QDFT).

Usage

```
qsmooth.qdft(
  y.qdft,
  method = c("gamm", "sp"),
  spar = "GCV",
  n.cores = 1,
  cl = NULL
)
```

Arguments

y.qdft	matrix or array of QDFT from `qdft()`
method	smoothing method: `"gamm"` for `mgcv::gamm()` (default), `"sp"` for `stats::smooth.spline()`
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

matrix or array of quantile-smoothed QDFT

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.qdft <- qdft(cbind(y1,y2),tau)
y.qdft <- qsmooth.qdft(y.qdft,method="sp",spar=0.9)
y.qacf <- qdft2qacf(y.qdft)
y.qper.qslw <- qspec.lw(y.qacf,M=5)$spec
qfa.plot(ff[sel.f],tau,Re(y.qper.qslw[1,1,sel.f,]))
```

qsmooth.qper *Quantile Smoothing of Quantile Periodogram or Spectral Estimate*

Description

This function computes quantile-smoothed version of quantile periodogram/cross-periodogram (QPER) or other quantile spectral estimate.

Usage

```
qsmooth.qper(
  y.qper,
  method = c("gamm", "sp"),
  spar = "GCV",
  n.cores = 1,
  cl = NULL
)
```

Arguments

`y.qper`	matrix or array of quantile periodogram/cross-periodogram or spectral estimate
`method`	smoothing method: `"gamm"` for `mgcv::gamm()` (default), `"sp"` for `stats::smooth.spline()`
`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

matrix or array of quantile-smoothed quantile spectral estimate

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.qdft <- qdft(cbind(y1,y2),tau)
y.qacf <- qdft2qacf(y.qdft)
y.qper.lw <- qspec.lw(y.qacf,M=5)$spec
qfa.plot(ff[sel.f],tau,Re(y.qper.lw[1,1,sel.f,]))
y.qper.lwqs <- qsmooth.qper(y.qper.lw,method="sp",spar=0.9)
qfa.plot(ff[sel.f],tau,Re(y.qper.lwqs[1,1,sel.f,]))
```

qspec.ar — *Autoregression (AR) Estimator of Quantile Spectrum or Quantile-Crossing Spectrum*

Description

This function computes autoregression (AR) estimate of quantile spectrum (QSPEC) or quantile-crossing spectrum (QCSPEC) from a time series (for QSPEC only) or from pre-calculated quantile series (QSER) or quantile-crossing series (QCSER).

Usage

```
qspec.ar(
  y,
  tau,
  y.qser = NULL,
  p = NULL,
  order.max = NULL,
  freq = NULL,
  method = c("NA", "gamm", "sp"),
  n.cores = 1,
  cl = NULL
)
```

Arguments

`y`	time series (if multivariate, `nrow(y)` = length of time series) for QSPEC only
`tau`	sequence of quantile levels in (0,1)
`y.qser`	pre-calculated QSER or QCSER from `qdft2qser()`, `qser()`, or `qcser()` (default = `NULL`: compute from `y` and `tau` for QSPEC only); if `y.qser` is supplied, `y` and `tau` can be left unspecified
`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`	post-smoothing method: `"gamm"`, `"sp"`, or `"NA"` (default)
`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/cross-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,]))
```

---

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

Description

This function computes lag-window (LW) estimate of quantile spectrum (QSPEC) or quantile-crossing spectrum (QCSPEC) from the ACF of quantile series (QSER) or quantile-crossing series (QCSER).

Usage

```
qspec.lw(y.qacf, M = NULL)
```

Arguments

`y.qacf`	pre-calculated ACF of QSER or QCSER from `qdft2qacf()` or `qser2qacf()`
`M`	bandwidth parameter of lag window (default = `NULL`: quantile periodogram)

Value

A list with the following elements:

`spec`	matrix or array of quantile spectrum/cross-spectrum
`lw`	lag-window sequence

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)
y.qspec.lw <- qspec.lw(y.qacf,M=5)$spec
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.qspec.lw[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.qacf <- qdft2qacf(y.qdft)
y.qspec.lw <- qspec.lw(y.qacf,M=5)$spec
qfa.plot(ff[sel.f],tau,Re(y.qspec.lw[1,2,sel.f,]))
```

---

qspec.lwqs — *Lag-Window-Quantile-Smoothing (LWQS) Estimator of Quantile Spectrum*

Description

This function computes lag-window-quantile-smoothing (LWQS) estimate of quantile spectrum/cross-spectrum from time series or quantile autocovariance function (QACF).

Usage

```
qspec.lwqs(
  y,
  tau,
  y.qacf = NULL,
  M = NULL,
  method = c("gamm", "sp"),
  spar = "GCV",
  n.cores = 1,
  cl = NULL
)
```

Arguments

y	time series (if multivariate, `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	smoothing method: `"gamm"` for `mgcv::gamm()` (default), `"sp"` for `stats::smooth.spline()`
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 quantile spectrum/cross-spectrum
spec.lw	matrix or array of quantile spectrum/cross-spectrum 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.qper.lwqs <- qspec.lwqs(cbind(y1,y2),tau,M=5,method="sp",spar=0.9)$spec
qfa.plot(ff[sel.f],tau,Re(y.qper.lwqs[1,1,sel.f,]))
```

---

qspec.qslw — *Quantile-Smoothing-Lag-Window (QSLW) Estimator of Quantile Spectrum*

Description

This function computes quantie-smoothing-lag-window (QSLW estimate of quantile spectrum/cross-spectrum from time series or quantile discrete Fourier transform (QDFT).

Usage

```
qspec.qslw(
  y,
  tau,
  y.qdft = NULL,
  M = NULL,
  method = c("gamm", "sp"),
  spar = "GCV",
  n.cores = 1,
  cl = NULL
)
```

Arguments

`y`	time series (if multivariate, `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
`M`	bandwidth parameter of lag window (default = `NULL`: quantile periodogram)
`method`	smoothing method: `"gamm"` for `mgcv::gamm()` (default), `"sp"` for `stats::smooth.spline()`
`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 quantile spectrum/cross-spectrum
`lw`	lag-window sequence
`qdft`	matrix or array of quantile discrete Fourier transform if `y.qdft = 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.qper.qslw <- qspec.qslw(cbind(y1,y2),tau,M=5,method="sp",spar=0.9)$spec
qfa.plot(ff[sel.f],tau,Re(y.qper.qslw[1,1,sel.f,]))
```

---

qspec.sar — *Spline Autoregression (SAR) Estimator of Quantile Spectrum or Quatile-Crossing Spectrum*

Description

This function computes spline autoregression (SAR) estimate of quantile spectrum (QSPEC) or quantile-crossing spectrum (QCSPEC) from a time series (for QSPEC only) or from pre-calculated quantile series (QSER) or quantile-crossing series (QCSER).

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`	time series (if multivarite, `nrow(y)` = length of time series) for QSPEC only
`y.qser`	pre-calculated QSER or QCSER from `qdft2qser()`, `qser()`, or `qcser()` (default = `NULL`: compute from `y` and `tau`) for QSPEC); 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 or quantile-crossing 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$spec[1,1,sel.f,]))
# compute from quantile series
y.qser <- qser(cbind(y1,y2),tau)
y.qspec.sar <- qspec.sar(y.qser=y.qser,tau=tau,p=1)$spec
qfa.plot(ff[sel.f],tau,Re(y.qspec.sar[1,1,sel.f,]))
```

---

qspec.sqrlw — *Spline-Quantile-Regression-Lag-Window (SQRLW) Estimator of Quantile Spectrum*

Description

This function computes spline-quantile-regression-lag-window (SQRLW) estimate of quantile spectrum/cross-spectrum from time series or spline quantile discrete Fourier transform (SQDFT).

Usage

```
qspec.sqrlw(
  y,
  tau,
  y.sqdft = NULL,
  M = NULL,
  c0 = 0.02,
  d = 4,
  weighted = FALSE,
  n.cores = 1,
  cl = NULL
)
```

Arguments

y	time series (if multivariate, nrow(y) = length of time series)
tau	sequence of quantile levels in (0,1)
y.sqdft	matrix or array of pre-calculated SQDFT (default = NULL: compute from y and tau); if y.sqdft is supplied, y and tau can be left unspecified
M	bandwidth parameter of lag window (default = NULL: quantile periodogram)
c0	penalty parameter for SQDFT
d	subsampling rate of quantile levels for SQDFT (default = 1)
weighted	if TRUE, SQR penalty function is weighted (default = FALSE)
n.cores	number of cores for parallel computing of SQDFT (default = 1)
cl	pre-existing cluster for repeated parallel computing of SQDFT (default = NULL)

Value

A list with the following elements:

spec	matrix or array of quantile spectrum/cross-spectrum
lw	lag-window sequence
sqdft	matrix or array of spline quantile discrete Fourier transform if y.sqdft = 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.qper.sqrlw <- qspec.sqrlw(cbind(y1,y2),tau,M=5,c0=0.02,d=4)$spec
qfa.plot(ff[sel.f],tau,Re(y.qper.sqrlw[1,1,sel.f,]))
```

---

qspec2qcoh *Quantile Coherence Spectrum*

Description

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

Usage

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

Arguments

qspec	array of quantile spectrum/cross-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,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 = TRUE,
  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` (default), multithread BLAS is enabled
`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=32)
y21 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=32)
y12 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=32)
y22 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=32)
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)
dim(A1.sim)
```

---

sar.eq.test — *Wald Test and Confidence Band for Equality of SAR-Based 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=32)
y21 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=32)
y12 <- stats::arima.sim(list(order=c(1,0,0), ar=0.5), n=32)
y22 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=32)
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))
sar.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 = TRUE,
  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` (default), multithread BLAS is enabled
`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=32)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=32)
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 SAR-Based 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=32)
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=32)
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)
```

sqdft — *Spline Quantile Discrete Fourier Transform (SQDFT)*

Description

This function computes spline quantile discrete Fourier transform (SQDFT) for univariate or multivariate time series through trigonometric spline quantile regression.

Usage

```
sqdft(y, tau, c0 = 0.02, d = 4, weighted = FALSE, n.cores = 1, cl = NULL)
```

Arguments

`y`	time series (if multivariate, `nrow(y)` = length of time series)
`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`)
`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 the spline 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.sqdft <- sqdft(y,tau,c0=0.02,d=4)
n <- length(y)
ff <- c(0:(n-1))/n
sel.f <- which(ff > 0 & ff < 0.5)
y.qacf <- qdft2qacf(y.sqdft)
y.qper.sqrlw <- qspec.lw(y.qacf,M=5)$spec
qfa.plot(ff[sel.f],tau,Re(y.qper.sqrlw[sel.f,]))
```

`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 = TRUE)
```

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` (default), multithread BLAS is enabled

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`	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')
```

`tsqr.fit` *Trigonometric Spline Quantile Regression (TSQR)*

Description

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

Usage

```
tsqr.fit(y, f0, tau, c0, d = 1, weighted = FALSE, prepared = TRUE)
```

Arguments

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

Value

object of `sqr.fit()` (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)
fit.sqr <- tsqr.fit(y,f0=0.1,tau=tau,c0=0.02,d=4)
plot(tau,fit$coef[1,],type='p',xlab='QUANTILE LEVEL',ylab='TQR COEF')
lines(tau,fit.sqr$coef[1,],type='l')
```

Index