Package ‘qfa’

January 7, 2023

Type Package

Title Quantile-Frequency Analysis (QFA) of Time Series

Version 1.2

Date 2023-01-06

Author Ta-Hsin Li <thl@us.ibm.com>

Maintainer Ta-Hsin Li <thl@us.ibm.com>

Description Quantile-frequency analysis (QFA) of univariate or multivariate time series based on the method of 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

License GPL (>=2)

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

NeedsCompilation yes

Encoding UTF-8

RoxygenNote 7.2.1

R topics documented:

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 the 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() 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)
qacf <- qdft2qacf(y.qdft)
plot(c(0:9),qacf[c(1:10),1],type='h',xlab="LAG",ylab="QACF")
qser <- qdft2qacf(y.qdft,return.qser=TRUE)$qser
plot(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)
qacf <- qdft2qacf(y.qdft)
plot(c(0:9),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)
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(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)
qper <- qdft2qper(y.qdft)
qfa.plot(ff[sel.f],tau,Re(qper[1,1,sel.f,]))
qfa.plot(ff[sel.f],tau,Re(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)
qser <- qdft2qser(y.qdft)
plot(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)
qser <- qdft2qser(y.qdft)
plot(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,
  qper,
  rg.qper = range(qper),
  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
`qper`	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

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

Description

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

Usage

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

Arguments

`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 `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

qper2qcoh — *Quantile Coherence Spectrum (QCOH)*

Description

This function computes quantile coherence spectrum (QCOH) from quantile spectrum and cross-spectrum.

Usage

```
qper2qcoh(qspec, k = 1, kk = 2)
```

Arguments

`qspec`	array of quantile spectrum and 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.qdft <- qdft(cbind(y1,y2),tau)
qacf <- qdft2qacf(y.qdft)
qper.lw <- qspec.lw(qacf,M=5)$spec
qcoh <- qper2qcoh(qper.lw,k=1,kk=2)
qfa.plot(ff[sel.f],tau,Re(qcoh))
```

`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()`, `"sp"` for `stats::smooth.spline()`
`spar`	smoothing parameter in `smooth.spline()` (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)
qacf <- qdft2qacf(y.qdft)
qper.qslw <- qspec.lw(qacf,M=5)$spec
qfa.plot(ff[sel.f],tau,Re(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(
  qper,
  method = c("gamm", "sp"),
  spar = "GCV",
  n.cores = 1,
  cl = NULL
)
```

Arguments

`qper`	matrix or array of quantile periodogram/cross-periodogram or spectral estimate
`method`	smoothing method: `"gamm"` for `mgcv::gamm()`, `"sp"` for `stats::smooth.spline()`
`spar`	smoothing parameter in `smooth.spline()` (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)
qacf <- qdft2qacf(y.qdft)
qper.lw <- qspec.lw(qacf,M=5)$spec
qfa.plot(ff[sel.f],tau,Re(qper.lw[1,1,sel.f,]))
qper.lwqs <- qsmooth.qper(qper.lw,method="sp",spar=0.9)
qfa.plot(ff[sel.f],tau,Re(qper.lwqs[1,1,sel.f,]))
```

qspec.lw	*Lag-Window Estimator of Quantile Spectrum and Cross-Spectrum (QSPEC)*

Description

This function computes lag-window (LW) estimate of quantile spectrum/cross-spectrum (QSPEC) from QACF.

Usage

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

Arguments

qacf	matrix or array of QACF from `qdft2qacf()`
M	bandwidth parameter of lag window (default = `NULL`: quantile periodogram)

Value

A list with the following elements:

spec	matrix or array of LW estimate
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)
qacf <- qdft2qacf(y.qdft)
qper.lw <- qspec.lw(qacf,M=5)$spec
n <- length(y1)
ff <- c(0:(n-1))/n
sel.f <- which(ff > 0 & ff < 0.5)
qfa.plot(ff[sel.f],tau,Re(qper.lw[sel.f,]))
# multiple time series
y2 <- stats::arima.sim(list(order=c(1,0,0), ar=-0.5), n=64)
y.qdft <- qdft(cbind(y1,y2),tau)
qacf <- qdft2qacf(y.qdft)
qper.lw <- qspec.lw(qacf,M=5)$spec
qfa.plot(ff[sel.f],tau,Re(qper.lw[1,2,sel.f,]))
```

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

Description

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

Usage

```
sqdft(y, tau, c0 = 0.02, d = 4, weighted = FALSE, 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)
`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)
qacf <- qdft2qacf(y.sqdft)
qper.sqrlw <- qspec.lw(qacf,M=5)$spec
qfa.plot(ff[sel.f],tau,Re(qper.sqrlw[sel.f,]))
```

sqr.fit — *Spline Quantile Regression (SQR)*

Description

This function computes the spline quantile regression (SQR) solution given response vector and design matrix. It uses the code `rqfnb.f` in the "quantreg" package with the 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')
```

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