Package 'mwaved'

March 22, 2019

Type Package

Title Multichannel Wavelet Deconvolution with Additive Long Memory Noise

Version 1.1.6

Date 2019-03-22

Author Justin Rory Wishart

Maintainer Justin Rory Wishart <justin.wishart@mq.edu.au>

Description Computes the Wavelet deconvolution estimate of a common signal present in multiple channels that have possible different levels of blur and long memory additive error.

Encoding UTF-8

License GPL

LinkingTo Rcpp

Imports Rcpp, shiny, grid

SystemRequirements fftw3 (>= 3.3.4)

NeedsCompilation yes

Suggests fracdiff, ggplot2, testthat, knitr, rmarkdown

URL https://github.com/jrwishart/mwaved

BugReports https://github.com/jrwishart/mwaved/issues

VignetteBuilder knitr

RoxygenNote 6.1.1

Repository CRAN

Date/Publication 2019-03-22 22:10:03 UTC

R topics documented:

`blurSignal` *Blur an input signal*

Description

An input signal is blurred by a set of functions to obtain a blurred multichannel signal.

Usage

```
blurSignal(signal, G)
```

Arguments

`signal`	A numeric vector of length n of the signal of interest
`G`	An n by m matrix of blur functions to be applied to the signal of interest.

Details

Applies the convolution operator to the signal of interest with each column of G to create a multichannel blurred signal. This operation is done in the Fourier domain using the base R fft transforms.

See Also

`gammaBlur`, `boxcarBlur`

Examples

```
n <- 1024
m <- 3
blur <- gammaBlur(n, shape = seq(from = 0.5, to = 1, length = m), scale = rep(0.25, m))
x <- (1:n)/n
signal <- makeLIDAR(n)
par(mfrow = c(2,1))
plot(x, signal, type = 'l', main = 'Direct LIDAR signal')
indirectSignal <- blurSignal(signal, blur)
matplot(x, indirectSignal, type = 'l', main = 'Set of blurred LIDAR signals')
```

---

boxcarBlur *Multichannel box car blur*

Description

Create blur of the box car type.

Usage

```
boxcarBlur(n, width)
```

Arguments

n	An integer specifying the number of observations in each channel
width	A numeric vector of length m where each element specifies the box car widths to blur in each channel.

Details

Creates a matrix of dimension n by m which contains a normalised (unit energy in each column) set of box car blur functions. The generated box-car functions are governed by an input vector BA which specifies the box car width. For the method to adhere to the theory described in Wishart (2014), the box car widths should be a Badly Approximable (BA) number.

Value

A n by m matrix of normalised box car blur.

See Also

gammaBlur, blurSignal

Examples

```
n <- 1024
width <- 1/sqrt(c(89,353))

# Plot the box car blur
blurMat <- boxcarBlur(n, width)
x <- (1:n)/n
matplot(x, blurMat, type = 'l', main = paste('Set of box car blur functions'))

# Plot a LIDAR signal and its multichannel box car blurred version
signal <- makeLIDAR(n)
matplot(x, signal, type = 'l', main = 'LIDAR test signal')
blurredSignal <- blurSignal(signal, blurMat)
matplot(x, blurredSignal, type = 'l', main = 'Box car blurred LIDAR test signals')
```

---

detectBlur *Detect type of blur*

Description

Detect the form of the input blur matrix, G

Usage

```
detectBlur(G)
```

Arguments

G	The input blur matrix to be analysed and checked whether it corresponds to direct blur or box.car blur.

Details

Detects if the input blur matrix, G, has uniform structure in being of direct blur type everywhere or box.car type everywhere. In those cases, it will return a character string 'direct' or 'box.car' respectively, otherwise it returns 'smooth'. This is done in the direct blur case by checking that the mvfft(G) is equal to 1 everywhere (complex part is zero everywhere) and in the box.car case by checking that each column has two unique values, a zero and positive value. If the blur type is not identified to be direct or box.car, the string 'smooth' is returned.

directBlur *Direct kernel matrix*

Description

Creates appropriately sized blur matrix for the special case when no blurring is apparent.

Usage

```
directBlur(n, m = 1)
```

Arguments

n	Number of observations in each channel
m	Number of channels

Details

Function creates a matrix of dimension n by m which contains appropriate entries for the case when a direct multichannel signal is observed. That is, no blurring operator is apparent. This is the default argument for the blurring matrix to all the multichannel functions in the mwaved package.

gammaBlur *Multichannel Gamma density blur*

Description

Create blur of the regular smooth type generated from Gamma densities

Usage

```
gammaBlur(n, shape, scale)
```

Arguments

n	An integer specifying the number of observations in each channel.
shape	A numeric vector of length m where each element specifies the shape parameter for the Gamma density used to blur each channel.
scale	A numeric vector of length m where each element specifies the scale parameter for the Gamma density used to blur each channel.

Details

Function creates a matrix of dimension n by m which contains a normalised (unit energy in each column) set of blur functions of regular smooth type, generated by a Gamma density. These Gamma densities are generated using the dgamma base R function and then normalised to have unit energy.

Value

A numeric n by m matrix of normalised Gamma blur.

See Also

dgamma for details of the Gamma density

boxcarBlur, blurSignal

Examples

```
n <- 1024
m <- 3
shape <- seq(from = 0.5, to = 1, length = m)
scale <- rep(0.25,m)

# Plot the smooth (gamma) blur
x <- (1:n)/n
blurMat <- gammaBlur(n, shape, scale)
matplot(x, blurMat, type = 'l', main = paste('Set of Gamma', m,'Gamma blur densities.'))

# Plot a LIDAR signal and its multichannel smooth blurred version
signal <- makeLIDAR(n)
matplot(x, signal, type = 'l', main = 'LIDAR test signal')
blurredSignal <- blurSignal(signal, blurMat)
matplot(x, blurredSignal, type = 'l', main = 'Smooth blurred LIDAR test signals')
```

---

makeSignals *Generate test signals for simulation*

Description

Generates some of the test signals used the standard nonparametric deconvolution literature.

Usage

```
makeLIDAR(n)

makeBumps(n)

makeDoppler(n)

makeCusp(n)

makeBlocks(n)

makeHeaviSine(n)
```

Arguments

n	An integer specifying the length of the desired signal.

Value

A numeric vector of length n giving the desired test signal.

Examples

```
n <- 1024
x <- (1:n)/n
signal <- makeLIDAR(n)
plot(x, signal, main = 'LIDAR test signal', type = 'l')
signal <- makeBumps(n)
plot(x, signal, main = 'Bumps test signal', type = 'l')
signal <- makeDoppler(n)
plot(x, signal, main = 'Doppler test signal', type = 'l')
signal <- makeCusp(n)
plot(x, signal, main = 'Cusp test signal', type = 'l')
signal <- makeBlocks(n)
plot(x, signal, main = 'Blocks test signal', type = 'l')
signal <- makeHeaviSine(n)
plot(x, signal, main = 'HeaviSine test signal', type = 'l')
```

---

multiCoef *Wavelet coefficient estimation from a multichannel signal*

Description

Estimates the wavelet coefficients for the underlying signal of interest embedded in the noisy multichannel deconvolution model.

Usage

```
multiCoef(Y, G = directBlur(nrow(as.matrix(Y)), ncol(as.matrix(Y))),
  alpha = rep(1, dim(as.matrix(Y))[2]),
  resolution = resolutionMethod(detectBlur(G)), j0 = 3L,
  j1 = NA_integer_, eta = NA_real_, deg = 3L)
```

Arguments

Y	An input signal either an n by m matrix containing the multichannel signal to be analysed or single vector of n elements for the single channel. In the multichannel case, each of the m columns represents a channel of n observations.
G	The input multichannel blur matrix/vector (needs to be the same dimension/length as the signal input which is a matrix or vector for the multichannel or single channel case respectively). This argument dictates the form of blur present in each of the channels.

`alpha` A numeric vector, with m elements, specifying the level of long memory for the noise process within each channel of the form alpha = 2 - 2H, where H is the Hurst parameter. If alpha is a single element, that same element is repeated across all required channels.

`resolution` A character string describing which resolution selection method is to be applied.

- 'smooth': Smooth stopping rule in Fourier domain applied piecewise in each channel and maximum selected which is appropriate if blurring kernel is of regular smooth blur type or direct model (no convolution).
- 'block': Blockwise variance selection method is used which is appropriate for box car type.

The default choice uses the detectBlur function to identify what type of blur matrix, G, is input and then maps that identification to the resolution type via a simple switch statement in the hidden `resolutionMethod` function, whereby, identified 'smooth' and 'direct' blur use the smooth resolution selection while box.car uses the blockwise resolution selection method.

`j0` The coarsest resolution level for the wavelet expansion.

`j1` The finest resolution level for the wavelet expansion. If unspecified, the function will compute all thresholds up to the maximum possible resolution level at j1 = log2(n) - 1.

`eta` The smoothing parameter. The default level is $2\sqrt{(}\alpha^*)$ where α^* is an optimal level depending on the type of blur. (see Kulik, Sapatinas and Wishart (2014) for details and justification)

`deg` The degree of the auxiliary polynomial used in the Meyer wavelet.

Details

Returns an object of type *waveletCoef* which is a list including the following three objects

- `coef` a numeric vector of size n giving the estimated wavelet coefficients for the signal of interest
- `j0` an integer that specifies the initial coarse resolution for the inhomogeneous wavelet expansion.
- `deg` an integer that specifies the degree of the Meyer wavelet used in the estimation of the wavelet coefficients.

Examples

```
library(mwaved)
# Simulate the multichannel doppler signal.
m <- 3
n <- 2^10
signal <- makeDoppler(n)
# Noise levels per channel
e <- rnorm(m*n)
# Create Gamma blur
shape <- seq(from = 0.5, to = 1, length = m)
scale <- rep(0.25,m)
```

```
G <- gammaBlur(n, shape = shape, scale = scale)
# Convolve the signal
X <- blurSignal(signal, G)
# Create error with custom signal to noise ratio
SNR <- c(10,15,20)
sigma <- sigmaSNR(X, SNR)
if (requireNamespace("fracdiff", quietly = TRUE)) {
  alpha <- c(0.75, 0.8, 1)
} else {
  alpha <- rep(1, m)
}
E <- multiNoise(n, sigma, alpha)
# Create noisy & blurred multichannel signal
Y <- X + E
plot(signal, type='l', lty='dashed', main='dashed: True signal, solid: multichannel signals')
matlines(Y, lty = 1)
# Estimate the wavelet coefficients
estimatedCoefs <- multiCoef(Y, G, alpha = alpha)
plot(estimatedCoefs)
# Compute true wavelet coefficients
trueCoefs <- multiCoef(signal)
plot(trueCoefs)
```

`multiEstimate` *Wavelet deconvolution signal estimate from the noisy multichannel convoluted signal*

Description

Estimates the underlying signal of interest from a multichannel noisy deconvolution model.

Usage

```
multiEstimate(Y, G = directBlur(nrow(as.matrix(Y)), ncol(as.matrix(Y))),
  alpha = rep(1, dim(as.matrix(Y))[2]),
  resolution = resolutionMethod(detectBlur(G)),
  sigma = as.numeric(c()), j0 = 3L, j1 = NA_integer_,
  eta = NA_real_, thresh = multiThresh(as.matrix(Y), G = G, alpha =
  alpha, j0 = j0, j1 = j1, eta = eta, deg = 3L), shrinkType = "hard",
  deg = 3L)
```

Arguments

`Y`	An input signal either an n by m matrix containing the multichannel signal to be analysed or single vector of n elements for the single channel. In the multichannel case, each of the m columns represents a channel of n observations.
`G`	The input multichannel blur matrix/vector (needs to be the same dimension/length as the signal input which is a matrix or vector for the multichannel or single channel case respectively). This argument dictates the form of blur present in each of the channels.

`alpha` A numeric vector, with m elements, specifying the level of long memory for the noise process within each channel of the form alpha = 2 - 2H, where H is the Hurst parameter. If alpha is a single element, that same element is repeated across all required channels.

`resolution` A character string describing which resolution selection method is to be applied.

- 'smooth': Smooth stopping rule in Fourier domain applied piecewise in each channel and maximum selected which is appropriate if blurring kernel is of regular smooth blur type or direct model (no convolution).
- 'block': Blockwise variance selection method is used which is appropriate for box car type.

The default choice uses the detectBlur function to identify what type of blur matrix, G, is input and then maps that identification to the resolution type via a simple switch statement in the hidden `resolutionMethod` function, whereby, identified 'smooth' and 'direct' blur use the smooth resolution selection while box.car uses the blockwise resolution selection method.

`sigma` A numeric vector with m elements that specifies the level of noise (standard deviation) in each channel. The default method uses the Median Absolute Deviation of wavelet coefficients in the finest resolution (see `multiSigma`) for details.

`j0` The coarsest resolution level for the wavelet expansion.

`j1` The finest resolution level for the wavelet expansion. If unspecified, the function will compute all thresholds up to the maximum possible resolution level at j1 = log2(n) - 1.

`eta` The smoothing parameter. The default level is $2\sqrt{(}\alpha^*)$ where α^* is an optimal level depending on the type of blur. (see Kulik, Sapatinas and Wishart (2014) for details and justification)

`thresh` A numeric vector of resolution level thresholds to use in the wavelet thresholded estimator of the true signal. It should have enough elements to construct the required expansion with all resolutions. That is, have `j1 - j0 + 2` elements. If a single element is input, it is repeated to be the universal threshold across all resolutions.

`shrinkType` A character string that specifies which thresholding regime to use. Available choices are the 'hard', 'soft' or 'garrote'.

`deg` The degree of the auxiliary polynomial used in the Meyer wavelet.

Details

Function requires input of a noisy multichannel signal matrix, Y, which contains the information for each channel in each of the m columns. Optional inputs are a matrix, G, the same dimension as Y, that gives the multichannel blur information.

Value

A numeric vector of the estimate of the underlying signal of interest.

Examples

```
library(mwaved)
# Simulate the multichannel doppler signal.
m <- 3
n <- 2^10
x <- (1:n)/n
signal <- makeDoppler(n)
# Noise levels per channel
e <- rnorm(m * n)
# Create Gamma blur
shape <- seq(from = 0.5, to = 1, length = m)
scale <- rep(0.25, m)
G <- gammaBlur(n, shape = shape, scale = scale)
# Convolve the signal
X <- blurSignal(signal, G)
# Create error with custom signal to noise ratio
SNR <- c(10, 15, 20)
sigma <- sigmaSNR(X, SNR)
if (requireNamespace("fracdiff", quietly = TRUE)) {
  alpha <- c(0.75, 0.8, 1)
} else {
  alpha <- rep(1, m)
}
E <- multiNoise(n, sigma, alpha)
# Create noisy & blurred multichannel signal
Y <- X + E
# Estimate the underlying doppler signal
dopplerEstimate <- multiEstimate(Y, G = G, alpha = rep(1, m))
# Plot the result and compare with truth
par(mfrow=c(2, 1))
matplot(x, Y, type = 'l', main = 'Noisy multichannel signal')
plot(x, signal, type = 'l', lty = 2, main = 'True Doppler signal and estimate', col = 'red')
lines(x, dopplerEstimate)
```

---

multiNoise *Generate multichannel noise*

Description

Generate a matrix of multichannel (possibly long memory) noise variables

Usage

```
multiNoise(n, sigma = 1, alpha = length(sigma), ...)
```

Arguments

n	An integer specifying the number of observations per channel.
sigma	A vector giving the noise levels (standard deviation) for each channel in the multichannel model (see details).

alpha	A vector specifying the dependence level in each channel.
...	Additional arguments to pass to the fracdiff package to tightly control the long memory noise.

Details

Generates a n by m matrix of noise variables. Long memory variables can be generated by the use of the optional fracdiff package (if installed). The dependence is specified using the alpha parameter where alpha = 2 - 2H where H = Hurst parameter. Long memory is ensured when alpha is between 0 and 1 (H between 1/2 and 1). If alpha is a single element and sigma has more than one element (multichannel), then the same dependence level of alpha is used amongst all of the channels. Otherwise the size of alpha and sigma should be the same size.

See Also

sigmaSNR

Examples

```
n <- 1024
m <- 3
signal <- makeLIDAR(n)
blur <- gammaBlur(n, c(0.5, 0.75, 1), rep(1, m))
X <- blurSignal(signal, blur)
SNR <- 10*1:3
sigma <- sigmaSNR(X, SNR)
E <- multiNoise(n, sigma, alpha = c(0.5, 0.75, 1))
matplot(X + E, type = 'l')
```

---

multiProj *Meyer wavelet projection given a set of wavelet coefficients*

Description

Reconstructs a function using wavelet coefficients (waveletCoef object) as input.

Usage

```
multiProj(beta, j1 = log2(length(beta$coef)) - 1)
```

Arguments

beta	A waveletCoef object that contains a vector of wavelet coefficients and the coarsest resolution level, j0 to create the required output function expansion.
j1	The finest resolution to be used in the projection (specifies which resolution that the wavelet expansion is truncated).

Details

Function that takes an input of wavelet coefficients in the form of a `waveletCoef` object (see `multiCoef` for details) and optionally a desired maximum resolution level, `j1`, to create an inhomogeneous wavelet expansion starting from resolution `j0` up to resolution `j1`. Namely, it creates the wavelet expansion,

$$\sum_{k=0}^{2^{j_0}-1} \beta_k \phi_{j_0,k} + \sum_{j=j_0}^{j_1} \sum_{k=0}^{2^j-1} \beta_{j,k} \psi_{j,k}.$$

where (ϕ, ψ) denote the father and mother periodised Meyer wavelet functions and $\beta_{j,k}$ denotes the mother wavelet coefficient at resolution j and location k and β_k denotes the father wavelet coefficients at resolution $j = j0$ and location k. The coefficients `beta` need to be ordered so that the first $2^{\texttt{j0}}$ elements correspond to father wavelet coefficients at resolution $j =$ `j0` and the remaining elements correspond to the mother wavelet coefficients from resolution $j =$ `j0` to $j = log_2 n - 1$. If the maximum resolution level *j1* is not specified, the full wavelet expansion will be given.

Value

A numeric vector of size n giving the wavelet function expansion.

See Also

`multiCoef`

Examples

```
library(mwaved)
# Make a noiseless doppler function
n <- 2^8
x <- (1:n)/n
y <- makeDoppler(n)
# Determine the wavelet coefficients
beta <- multiCoef(y)
# plot three raw wavelet expansions truncating in each case at j1 = 3, 4 and 5 respectively
plot(x, y, type = 'l', main = 'Doppler and wavelet projections at three different truncations')
j0 <- 3
j1 <- 5
j <- j0:j1
lcols <- c(1, j - j0 + 2)
ltys <-  c(1, 1:length(j))
matlines(x, sapply(j, function(i) multiProj(beta, j1 = i)), type = 'l', col = lcols[-1])
legend("bottomright", legend = c("Signal", paste('j1 =', j)), col = lcols, lty =ltys)
```

`multiSigma` *Noise level estimation among multichannel signal*

Description

Estimates the level of noise (standard deviation of noise) in each channel.

Usage

```
multiSigma(Y, deg = 3L)
```

Arguments

Y	An input signal either an n by m matrix containing the multichannel signal to be analysed or single vector of n elements for the single channel. In the multichannel case, each of the m columns represents a channel of n observations.
deg	The degree of the auxiliary polynomial used in the Meyer wavelet.

Details

This function estimates the noise present in each channel by computing the Meyer wavelet transform of each channel of the multichannel signal. In particular, the noise level is computed by using the Median Absolute Deviation (MAD) of the wavelet coefficients at the highest possible resolution at J = floor(log2(n) - 1).

Value

A numeric vector of estimates of the standard deviation of the noise in each of the m channels.

Examples

```
library(mwaved)
# Simulate matrix of Gaussian variables with three different noise levels
sig <- c(0.25, 0.5, 1.25)
n <- 1024
Y <- sapply(1:3, function(i) sig[i]* rnorm(n))
# Estimate the noise levels
multiSigma(Y, deg = 3)
```

---

multiThresh *Resolution level thresholds for hard thresholded wavelet deconvolution estimator*

Description

Computes the estimated resolution level thresholds for the hard-thresholding wavelet deconvolution estimate of the desired signal in the multichannel signals.

Usage

```
multiThresh(Y, G = directBlur(nrow(as.matrix(Y)), ncol(as.matrix(Y))),
  alpha = rep(1, dim(as.matrix(Y))[2]),
  resolution = resolutionMethod(detectBlur(G)), j0 = 3L,
  j1 = NA_integer_, eta = NA_real_, deg = 3L)
```

Arguments

Y	An input signal either an n by m matrix containing the multichannel signal to be analysed or single vector of n elements for the single channel. In the multichannel case, each of the m columns represents a channel of n observations.
G	The input multichannel blur matrix/vector (needs to be the same dimension/length as the signal input which is a matrix or vector for the multichannel or single channel case respectively). This argument dictates the form of blur present in each of the channels.
alpha	A numeric vector, with m elements, specifying the level of long memory for the noise process within each channel of the form alpha = 2 - 2H, where H is the Hurst parameter. If alpha is a single element, that same element is repeated across all required channels.
resolution	A character string describing which resolution selection method is to be applied. • 'smooth': Smooth stopping rule in Fourier domain applied piecewise in each channel and maximum selected which is appropriate if blurring kernel is of regular smooth blur type or direct model (no convolution). • 'block': Blockwise variance selection method is used which is appropriate for box car type. The default choice uses the detectBlur function to identify what type of blur matrix, G, is input and then maps that identification to the resolution type via a simple switch statement in the hidden resolutionMethod function, whereby, identified 'smooth' and 'direct' blur use the smooth resolution selection while box.car uses the blockwise resolution selection method.
j0	The coarsest resolution level for the wavelet expansion.
j1	The finest resolution level for the wavelet expansion. If unspecified, the function will compute all thresholds up to the maximum possible resolution level at j1 = log2(n) - 1.
eta	The smoothing parameter. The default level is $2\sqrt{(}\alpha^*)$ where α^* is an optimal level depending on the type of blur. (see Kulik, Sapatinas and Wishart (2014) for details and justification)
deg	The degree of the auxiliary polynomial used in the Meyer wavelet.

Details

Given an input matrix of a multichannel signal (n rows and n columns) with m channels and n observations per channel, the function returns the required thresholds for the hard-thresholding estimator of the underlying function, f.

Value

A numeric vector of the resolution level thresholds for the hard-thresholding nonlinear wavelet estimator from the multichannel model.

References

Kulik, R., Sapatinas, T. and Wishart, J.R. (2014) *Multichannel wavelet deconvolution with long range dependence. Upper bounds on the L_p risk* Appl. Comput. Harmon. Anal. (to appear in). http://dx.doi.org/10.1016/j.acha.2014.04.004

Examples

```
library(mwaved)
# Simulate the multichannel doppler signal.
m <- 3
n <- 2^10
signal <- makeDoppler(n)
# Noise levels per channel
e <- rnorm(m * n)
# Create Gamma blur
shape <- seq(from = 0.5, to = 1, length = m)
scale <- rep(0.25, m)
G <- gammaBlur(n, shape = shape, scale = scale)
# Convolve the signal
X <- blurSignal(signal, G)
# Create error with custom signal to noise ratio
SNR <- c(10, 15, 20)
sigma <- sigmaSNR(X, SNR)
alpha <- c(0.75, 0.8, 1)
E <- multiNoise(n, sigma, alpha)
# Create noisy & blurred multichannel signal
Y <- X + E
# Determine thresholds blur = 'smooth'
thresh <- multiThresh(Y, G)
```

`multiWaveD` *Full mWaveD analysis*

Description

Returns a mWaveD object that contains all the required information for the multichannel analysis.

Usage

```
multiWaveD(Y, G = directBlur(nrow(as.matrix(Y)), ncol(as.matrix(Y))),
  alpha = rep(1, dim(as.matrix(Y))[2]), j0 = 3L, j1 = NA_integer_,
  resolution = resolutionMethod(detectBlur(G)), eta = NA_real_,
  thresh = as.numeric(c()), shrinkType = "hard", deg = 3L)
```

Arguments

Y	An input signal either an n by m matrix containing the multichannel signal to be analysed or single vector of n elements for the single channel. In the multichannel case, each of the m columns represents a channel of n observations.
G	The input multichannel blur matrix/vector (needs to be the same dimension/length as the signal input which is a matrix or vector for the multichannel or single channel case respectively). This argument dictates the form of blur present in each of the channels.
alpha	A numeric vector, with m elements, specifying the level of long memory for the noise process within each channel of the form alpha = 2 - 2H, where H is the Hurst parameter. If alpha is a single element, that same element is repeated across all required channels.
j0	The coarsest resolution level for the wavelet expansion.
j1	The finest resolution level for the wavelet expansion. If unspecified, the function will compute all thresholds up to the maximum possible resolution level at j1 = log2(n) - 1.
resolution	A character string describing which resolution selection method is to be applied. • 'smooth': Smooth stopping rule in Fourier domain applied piecewise in each channel and maximum selected which is appropriate if blurring kernel is of regular smooth blur type or direct model (no convolution). • 'block': Blockwise variance selection method is used which is appropriate for box car type. The default choice uses the detectBlur function to identify what type of blur matrix, G, is input and then maps that identification to the resolution type via a simple switch statement in the hidden resolutionMethod function, whereby, identified 'smooth' and 'direct' blur use the smooth resolution selection while box.car uses the blockwise resolution selection method.
eta	The smoothing parameter. The default level is $2\sqrt{(}\alpha^*)$ where α^* is an optimal level depending on the type of blur. (see Kulik, Sapatinas and Wishart (2014) for details and justification)
thresh	A numeric vector of resolution level thresholds to use in the wavelet thresholded estimator of the true signal. It should have enough elements to construct the required expansion with all resolutions. That is, have j1 - j0 + 2 elements. If a single element is input, it is repeated to be the universal threshold across all resolutions.
shrinkType	A character string that specifies which thresholding regime to use. Available choices are the 'hard', 'soft' or 'garrote'.
deg	The degree of the auxiliary polynomial used in the Meyer wavelet.

See Also

plot.mWaveD and summary.mWaveD

Examples

```
library(mwaved)
# Simulate the multichannel doppler signal.
m <- 3
n <- 2^10
signal <- makeDoppler(n)
# Noise levels per channel
e <- rnorm(m * n)
# Create Gamma blur
shape <- seq(from = 0.5, to = 1, length = m)
scale <- rep(0.25, m)
G <- gammaBlur(n, shape = shape, scale = scale)
# Convolve the signal
X <- blurSignal(signal, G)
# Create error with custom signal to noise ratio
SNR <- c(10, 15, 20)
sigma <- sigmaSNR(X, SNR)
if (requireNamespace("fracdiff", quietly = TRUE)) {
  alpha <- c(0.75, 0.8, 1)
} else {
  alpha <- rep(1, m)
}
E <- multiNoise(n, sigma, alpha)
# Create noisy & blurred multichannel signal
Y <- X + E
# Compute mWaveD object
mWaveDObj <- multiWaveD(Y, G = G, alpha = alpha)
plot(mWaveDObj)
summary(mWaveDObj)
```

---

mwaved *Multichannel wavelet deconvolution with long memory using mwaved.*

Description

mwaved computes the Wavelet deconvolution estimate of a common signal present in multiple channels that have possible different levels of blur and additive error. More information about each function can be found in its help documentation.

Details

mwaved uses the WaveD wavelet deconvolution paradigm and is based on the waved R-package given by Raimondo and Stewart (2007). It generalises the approach by allowing a multichannel signal instead of a single channel signal and allows long memory errors within each channel (independent of each channel). See Kulik, Sapatinas and Wishart (2014) for theoretical results and a short numerical investigation. The mwaved package also uses the external C FFTW library described in Frigo and Johnson, (2005) to dramatically increase the speed of the computations. Detailed information and instructions for implementation are available at http://www.fftw.org.

References

Frigo, M and Johnson, S.G. (2005) *The design and implementation of FFTW3*, Proceedings of the IEEE **93**, 216–231. http://dx.doi.org/10.1109/JPROC.2004.840301

Kulik, R., Sapatinas, T. and Wishart, J.R. (2014) *Multichannel wavelet deconvolution with long range dependence. Upper bounds on the L_p risk* Appl. Comput. Harmon. Anal. (to appear in). http://dx.doi.org/10.1016/j.acha.2014.04.004

Raimondo, M. and Stewart, M. (2007) *The WaveD Transform in R: Performs Fast Translation-Invariant Wavelet Deconvolution*, Journal of Statistical Software **21**, 1–28 http://www.jstatsoft.org/v21/i02

`mWaveDDemo` *Interactive Demonstration*

Description

Interactive Demonstration

Usage

```
mWaveDDemo()
```

`plot.mWaveD` *Plot Output for the mWaveD object*

Description

Creates plot output that summarises the `mWaveD` object produced by the `multiWaveD` function.

Usage

```
## S3 method for class 'mWaveD'
plot(x, ..., which = 1L:4L, singlePlot = TRUE,
  ask = !singlePlot, ggplot = TRUE)
```

Arguments

`x`	A mWaveD object to be plotted (list created by `multiWaveD`)
`...`	Arguments to be passed to methods.
`which`	A numeric vector that specifies which plots to output. Default value is `1:4` which specifies that all four plots are to be displayed.
`singlePlot`	A logical value that controls whether all plots should appear on a single window. The plot window is resized depending on the value of `which`.
`ask`	A logical value that specifies whether the user is *ask*ed before each plot output.
`ggplot`	A logical value to specify if the user wants to use base graphics (FALSE) or ggplot2 graphics (TRUE).

Details

Four plots are output that summarise the multichannel input, a visualisation of the characteristics of the channels and the output estimate and a multi-resolution analysis plot.

- Plot 1: Multichannel input signal overlayed.
- Plot 2: Estimated output signal using the mWaveD approach.
- Plot 3: Plot of the log decay of Fourier coefficients against the log bounds (direct and smooth case) or the blockwise resolution levels against their limit (box car case)
- Plot 4: Multi-resolution plot of the raw wavelet coefficients and the trimmed wavelet coefficients

References

Kulik, R., Sapatinas, T. and Wishart, J.R. (2014) *Multichannel wavelet deconvolution with long range dependence. Upper bounds on the L_p risk* Appl. Comput. Harmon. Anal. (to appear in). http://dx.doi.org/10.1016/j.acha.2014.04.004

Wishart, J.R. (2014) *Data-driven wavelet resolution choice in multichannel box-car deconvolution with long memory*, Proceedings of COMPSTAT 2014, Geneva Switzerland, Physica Verlag, Heidelberg (to appear)

See Also

multiWaveD

---

plot.waveletCoef *Multi-Resolution Analysis plot of wavelet coefficients*

Description

Plots the wavelet coefficient object in the multiresolution analysis

Usage

```
## S3 method for class 'waveletCoef'
plot(x, y = NULL, labels = NULL, ...,
  lowest = NULL, highest = NULL, scaling = 1, ggplot = TRUE)
```

Arguments

x	A list of class waveletCoef.
y	An optional numeric vector of trimmed wavelet coefficients to be overlayed on top of the plot for comparison with the x wavelet coefficients.
labels	Optional character vector with two elements to give name labels to x and y respectively.
...	Arguments to be passed to methods.

lowest	Specifies the coarsest resolution to display in the Multi-resolution plot.
highest	Specifies the finest resolution to display in the Multi-resolution plot.
scaling	A numeric value that acts as a graphical scaling parameter to rescale the wavelet coefficients in the plot. A larger scaling value will reduce the size of the coefficients in the plot.
ggplot	A logical value to specify if the user wants to use base graphics (FALSE) or ggplot2 graphics (TRUE).

See Also

multiCoef for generating a list of class 'waveletCoef'

---

resolutionMethod *Select appropriate resolution method for blur type*

Description

Simple function that maps the blur type to the appropriate resolution selection method

Usage

```
resolutionMethod(blur)
```

Arguments

blur	A character string that specifies the behaviour of the blur function • 'direct': No blur or direct model is used. • 'smooth': Blur that has smooth decay in the Fourier domain. • 'box.car': Blur that is of box.car type.

---

sigmaSNR *Determine noise scale levels from specified **S**ignal to **N**oise **R**atios*

Description

Compute the noise scale levels for each channel using the **S**ignal to **N**oise **R**atios

Usage

```
sigmaSNR(signal, SNR)
```

Arguments

signal	Noisefree multichannel input signal
SNR	A numeric vector specifying the desired **S**ignal to **N**oise **R**atio for each channel.

Details

The output noise scale levels (theoretical standard deviation for the process noise process in each channel) is governed by the blurred **S**ignal-to-**N**oise **R**atio (SNR) measured in decibels (dB) where,

$$SNR = 10log_{10}(\frac{||k * f||^2}{\sigma^2})$$

and k*f is the blurred signal, $||\cdot||$ is the norm operator and σ is the standard deviation of the noise. Roughly speaking, noise levels are considered high, medium and low for the cases 10 dB, 20 dB and 30 dB respectively.

Value

A numeric vector with m elements giving the scales (standard deviation of the noise in each channel) to achieve the desired SNR.

See Also

`multiNoise` `multiSigma`

Examples

```
n <- 1024
m <- 3
signal <- makeLIDAR(n)
blur <- gammaBlur(n, c(0.5, 0.75, 1), rep(1, m))
X <- blurSignal(signal, blur)
SNR <- 10*1:3
sigma <- sigmaSNR(X, SNR)
E <- multiNoise(n, sigma)
sigmaEst <- multiSigma(E)
```

---

summary.mWaveD *Summary Output for the mWaveD object*

Description

Gives some numerical summaries of a `mWaveD` object.

Usage

```
## S3 method for class 'mWaveD'
summary(object, ...)
```

Arguments

`object`	A mWaveD object which is a list containing all the information for a multichannel deconvolution analysis produced by the `multiWaveD` function.
`...`	Arguments to be passed to methods.

Value

Text output giving summary information of the input and output analysis including,

- Degree of Meyer wavelet used in the analysis.
- Number of observations, within each channel and number of channels present.
- Resolution levels used (j0 to j1)
- Blur type assumed in the analysis (direct, smooth or box.car)
- Matrix summarising the noise levels in each channel (and Fourier decay information for the smooth case)
- Summaries of the severity of the thresholding applied amongst the resolutions.

See Also

multiWaveD

Examples

```
library(mwaved)
# Simulate the multichannel doppler signal.
m <- 3
n <- 2^10
t <- (1:n)/n
signal <- makeDoppler(n)
# Create multichannel version with smooth blur
shape <- seq(from = 0.5, to = 1, length = m)
scale <- rep(0.25, m)
G <- gammaBlur(n, shape, scale)
X <- blurSignal(signal, G)
# Add noise with custom signal to noise ratio
SNR <- c(10,15,20)
E <- multiNoise(n, sigma = sigmaSNR(X, SNR), alpha = c(0.5, 0.75, 1))
# Create noisy & blurred multichannel signal
Y <- X + E
mWaveDObject <- multiWaveD(Y, G)
summary(mWaveDObject)
```

---

theoreticalEta *Find optimal theoretical Eta*

Description

Finds the optimal theoretical smoothing paramter for the thresholding

Usage

```
theoreticalEta(alpha, blur, G, sigma)
```

Arguments

alpha	A numeric vector, with m elements, specifying the level of long memory for the noise process within each channel of the form alpha = 2 - 2H, where H is the Hurst parameter. If alpha is a single element, that same element is repeated across all required channels.
blur	A character string that specifies the behaviour of the blur function • 'direct': No blur or direct model is used. • 'smooth': Blur that has smooth decay in the Fourier domain. • 'box.car': Blur that is of box.car type.
G	The input blur matrix
sigma	A numeric vector with m elements that specifies the level of noise (standard deviation) in each channel. The default method uses the Median Absolute Deviation of wavelet coefficients in the finest resolution (see multiSigma) for details.

Details

The theory (see Kulik, Sapatinas and Wishart (2014)) suggests that the optimal smoothing parameter depends on the best channel in the smooth blur case while depends on the level of dependence in the box.car blur case. This function finds the theoretically best eta from the data suggested by the theory in that paper.

Value

The theoretical eta (smoothing parameter) to be used in the thresholding.

References

Kulik, R., Sapatinas, T. and Wishart, J.R. (2014) *Multichannel wavelet deconvolution with long range dependence. Upper bounds on the L_p risk* Appl. Comput. Harmon. Anal. (to appear in). http://dx.doi.org/10.1016/j.acha.2014.04.004

---

waveletThresh *Apply thresholding regime to a set of wavelet coefficients*

Description

Applies a resolution level thresholding technique to a set of wavelet coefficients, embedded in a wavelet coefficient object.

Usage

```
waveletThresh(beta, thresh, shrinkType = "hard")
```

Arguments

beta	A waveletCoef object.
thresh	A numeric vector containing the thresholds to be applied to the coefficients at each resolution.
shrinkType	A character string that specifies which thresholding regime to use. Available choices are the 'hard', 'soft' or 'garrote'.

Details

Applies one of three specified wavelet thresholding regimes to a waveletCoef object (wavelet coefficient object created by multiCoef). If thresh is not specified, no thresholding is applied. The formulae applied for 'hard', 'soft' or 'garrote' are given by,

- Hard: $\delta(x) = x1(|x| > t)$
- Soft: $\delta(x) = (x - t)1(x > t) + (x + t)1(x > -t)$
- Garrote: $\delta(x) = (x - t^2/x)1(|x| > t)$

where 1 represents the indicator function and *t > 0* represents the threshold.

Examples

```
library(mwaved)
# Simulate the multichannel doppler signal.
m <- 3
n <- 2^10
signal <- makeDoppler(n)
# Noise levels per channel
e <- rnorm(m * n)
# Create Gamma blur
shape <- seq(from = 0.5, to = 1, length = m)
scale <- rep(0.25, m)
G <- gammaBlur(n, shape = shape, scale = scale)
# Convolve the signal
X <- blurSignal(signal, G)
# Create error with custom signal to noise ratio
SNR <- c(10, 15, 20)
sigma <- sigmaSNR(X, SNR)
if (requireNamespace("fracdiff", quietly = TRUE)){
  alpha <- c(0.75, 0.8, 1)
} else {
  alpha <- rep(1, m)
}
E <- multiNoise(n, sigma, alpha)
# Create noisy & blurred multichannel signal
Y <- X + E
# Determine thresholds
thresh <- multiThresh(Y, G)
beta <- multiCoef(Y, G)
betaShrunk <- waveletThresh(beta, thresh)
plot(beta, betaShrunk)
```

Index