Package ‘spectral’

October 14, 2022

Type Package
Title Common Methods of Spectral Data Analysis
Version 2.0
Author Martin Seilmayer
Maintainer Martin Seilmayer <martin.seilmayer@gmail.com>
Description On discrete data spectral analysis is performed by Fourier and Hilbert transforms as well as with model based analysis called Lomb-Scargle method. Fragmented and irregularly spaced data can be processed in almost all methods. Both, FFT as well as LOMB methods take multivariate data and return standardized PSD. For didactic reasons an analytical approach for deconvolution of noise spectra and sampling function is provided. A user friendly interface helps to interpret the results.
License GPL-2
Depends rasterImage,lattice,RhpcBLASctl,pbapply, R (>= 3.5.0)
LazyData FALSE
RoxygenNote 7.1.0
NeedsCompilation no
Repository CRAN
Date/Publication 2021-03-29 09:30:02 UTC

R topics documented:

.onAttach ................................................................. 2
.onDetach ............................................................... 3
amax ................................................................. 3
analyticFunction ...................................................... 4
BP ................................................................. 5
deconvolve .......................................................... 7
envelope ........................................................... 9
filter.fft .......................................................... 11
filter.lomb ......................................................... 12
gLmb .......................................................... 13
Description

The number of used cores is set to RhpcBLASctl::get_num_cores() on the attach event of the package.

Usage

.onAttach(libname, pkgname)

Arguments

libname       a character string giving the library directory where the package defining the
              namespace was found.

pkgname       a character string giving the name of the package.

Details

The algebra system of R relies on a BLAS library which can be set to use many threads / cores. This feature is considered as experimental since there are many differences across the operating systems R is running on. If there is an issue and there is a need to run R in multi-thread mode, consider to install a different optimized version of BLAS. If necessary, the number of cores required can also be changed manually by calling blas_set_num_threads(nCores) and omp_set_num_threads(nCores).

This function is invoked automatically
.onDetach

Reset multithread BLAS to default

Description
This function is invoked automatically

Usage
.onDetach(libpath)

Arguments
libpath  a character string giving the complete path to the package.

amax

Local Maxima

Description
Determines all local maxima from a real valued vector.

Usage
amax(x)

Arguments
x  numeric vector

Details
The purpose is to detect all local maxima in a real valued 1D vector. If the first element x[1] is the global maximum, it is ignored, because there is no information about the previous element. If there is a plateau, the first edge is detected.

Value
returns the indices of local maxima. If x[1] = max, then it is ignored.

Examples

a <- c(1,2,3,2,1,5,5,4)
amax(a) # 3, 6
analyticFunction  Analytic function

Description

In general a causal real valued signal in time has negative frequencies, when a Fourier transform is applied. To overcome this, a complex complement can be calculated to compensate the negative frequency spectrum. The result is called analytic signal or analytic function, which provides a one sided spectrum.

Usage

analyticFunction(x)

Arguments

x  real valued data vector

Details

An analytic function \( x_a \) is composed of the real valued signal representation \( y \) and its Hilbert transform \( H(y) \) as the complex complement

\[
x_a(t) = x(t) + iH(x(t))
\]

In consequence, the analytic function has a one sided spectrum, which is more natural. Calculating the discrete Fourier transform of such a signal will give a complex vector, which is only non zero until the half of the length. Every component higher than the half of the sampling frequency is zero. Still, the analytic signal and its spectrum are a unique representation of the original signal \( x(t) \). The new properties enables us to do certain filtering and calculations more efficient in the spectral space compared to the standard FFT approach. Some examples are:

Filtering  because the spectrum is one sided, the user must only modify values in the lower half of the vector. This strongly reduces mistakes in indexing. See filter.fft

Envelope functions  Since the Hilbert transform is a perfect phase shifter by \( \pi/2 \), the envelope of a band limited signal can be calculated. See envelope

Calculations  Deriving and integrating on band limited discrete data becomes possible, without taking the symmetry of the discrete Fourier transform into account. The secound example of the spec.fft function calculates the derivative as well, but plays with a centered spectrum and its corresponding "true" negative frequencies

A slightly different approach on the analytic signal can be found in R. Hoffmann "Signalanalyse und -erkennung" (Chap. 6.1.2). Here the signal \( x(t) \) is split into the even and odd part. According to Marko (1985) and Fritzsche (1995) this two parts can be composed to the analytic signal, which lead to the definition with the Hilbert transform above.
Value
Complex valued analytic function

References

BP
Simple bandpass function

Description
This function represents a simple weightening procedure for spectral filtering according to the type ("poly", "sinc", "bi-cubic", "gauss") provided.

Usage
BP(f, fc, BW, n = 3, type = "poly")

Arguments
f vector of frequencies
fc center frequency
BW bandwidth, with \( w[ \text{abs}(f - fc) > BW] == \text{min} \)
n degree of the polynom, \( n \) can be real, e.g. \( n = 1.6 \) as sinc alike
type Type of weightening function: "poly", "sinc", "bi-cubic", "exp", can be abbreviated

Details
The band pass is represented throughout a function in the form of four different types:
1. polynominial function
   \[ w = 1 - |((f - fc)/BW)|^n \]
   with the degree \( n \). The parameter \( fc \) controls the center frequency and desired band width \( BW \).
   Outside the band width
   \[ |f - fc| > BW \]
   the result is forced to zero. With \( n = 1.6 \) a quasi sinc-filter without side bands can be constructed. A quasi rectangular window can be gained by setting \( n > 5 \).
2. sinc function corresponds to a rectangular observation window in time domain with
\[ \Delta T 1 / BW \]
It values ALL frequencies according to the si(x) function. Calculation speed might be reduced.
3. bi-cubic encounters 2nd order interpolation kernel, providing a quasi rectangular observation window.
4. exponential Gauss curve. Here the band width is defined as the value of 90

Value
This function returns a weight vector [0..1], which is to apply to the frequency vector \( f \) in a top level function

Examples

```r
f <- seq(-50, 50, by = 1e-2)
fc <- 0.3
BW <- 0.75

par(mfrow = c(2, 1))
curve(BP(x, fc = fc, BW = BW, type = "p"), -2, 2, ylim = c(-0.2, 1),
     main = "Filter weights",
     xlab = "fx", ylab = "w")
curve(BP(x, fc = fc, BW = BW, type = "s"), add = TRUE, lty = 2)
curve(BP(x, fc = fc, BW = BW, type = "b"), add = TRUE, lty = 3)
curve(BP(x, fc = fc, BW = BW, type = "g"), add = TRUE, lty = 4)

abline(v = c(fc, fc + BW, fc - BW), lty = 3, col = "grey")

# the corresponding Fourier-Transforms

ty <- c("p", "s", "b", "g")
A0 <- integrate(BP, fc = fc, BW = BW, type = "s", lower = -2, upper = 2)$value

plot(NA, NA, xlab = "x", ylab = "|A|",
     main = "corresponding convolution kernels",
     xlim = 2 * c(-1, 1), ylim = c(0, sqrt(2) * A0 / (length(f) * BW * min(diff(f)))))

for(i in 1:length(ty))
{
    FT <- spec.fft(y = BP(f, fc, BW, type = ty[i]))
    lines(FT$fx * length(FT$fx) / diff(range(f)), Mod(FT$A), lty = i)
}
```
deconvolve

*Deconvolve Sampling Spectrum for Equidistant Sampling*

**Description**

The function removes the probable alias peaks in the power spectral density. These projections originate from correlated gaps, missing values and interactions with noise. The function should be considered as *experimental* but with didactic background.

**Usage**

```r
deconvolve(x, y, SNR.enable = T, SNR.level = 1)
```

**Arguments**

- `x`: sampling instances
- `y`: values
- `SNR.enable`: binary value, include or exclude the noise
- `SNR.level`: threshold in the sense of a multiple of mean() noise level

**Details**

In the special case of a non complete equidistant grid containing the data and missing values (NA), this function performs the deconvolution of $Y = \text{fft}(y)$ from the sampling spectrum of the acquisition series $x$. The data is assumed to exist on a equidistant grid with missing values and gaps.

Given a one dimensional vector $y$ of data this function reverses the spectral convolution of $Y = S*X + N$, if * describes the convolution operation and $Y = F(y)$ denotes the discrete Fourier transform via the operator $F(.)$. If, the sampling series $x$ is considered to be purely deterministic, which should be the case for captured data, or the distortions (missing values, gaps) are *correlated* (see example), then there exists an analytic inversion of the convolution. Given the general definition of power spectral density $|Y|^2 = |S*X + N|^2$ the challenge is to prove $|S*X + N|^2 |S|^2 + |X|^2 + |N|^2$. Here $N$ describes a stochastic term of gaussian noise. This issue is solved in correlation space, where convolution becomes a multiplication. The auto correlation function (acf) of $y$ is given by $R_y = F(|Y|^2)$. As a remark, IF we consider the special case of equispaced sampling, modeled by the Dirac distribution $\delta(x)$, it is easy to show that the correlation function of a product is the product of individual correlation functions, $F(|S* X|^2) = F(|S|^2).F(|X|^2)$.

The aim is, to approximate $S$ as the "true" spectrum. To the cost of the phase information, the result is the standardized power spectral density. The spectral noise term $F(N)$ is approximated by a threshold in Fourier space. Here `SNR.level` sets the factor of `mean(fft(y))` below which noise level is assumed. Above this value, the signal should be present. As a parameter to play with, `SNR.enable` enables or disables the noise term. This parameter was introduced to be consistent with present approaches, not considering the presence of noise.

**Value**

list of frequency $f$ and spectral density function $S$
Examples

```r
# Deconvolution
#
# we define a test function with gaps and noise
# we show:
# - the aliased Fourier spectrum and for comparison Lomb Spectrum
# - the corrected spectrum
#
## definition of the sampling series
x <- seq(0, pi/2, by = 5e-3)
n <- rnorm(length(x), sd = 0.1)

## definition of the test function
## with 2 frequencies
yf <- function(x)
  {cos(2*pi*5.123*x) +
   cos(2*pi*17*x)}

y <- yf(x)
y <- y - mean(y)

## define strongly correlated gaps
i <- NULL
i <- c(i, which(sin(2*pi / 0.3 * x) - 0.5 > 0))
i <- c(i, which(sin(2*pi / 0.04 * x + 1.123) - 0.5 > 0))
i <- sort(unique(i))

xs <- x
ys <- yf(xs) + n # add some noise
ys[i] <- NA

## for comparison we calculate a Lomb-Spectrum
LT <- spec.lomb(x = xs, y = ys,
   f = seq(0,250,by = 0.02),
   mode = "generalized"
)

WS <- deconvolve(x = xs, y = ys, SNR.enable = 1, SNR.level = 1)
FT <- spec.fft(x = xs, y = ys)
FTS <- spec.fft(x = xs, y = is.na(ys))
LTS <- spec.lomb(x = xs, y = is.na(ys), f = seq(0,250,by = 0.02))

## for comparison we calculate a Lomb-Spectrum
```

# results
#
# - signal spectrum (solid) dominant peaks at around f0 = {5, 17}
# - (minor) alias peaks (grey line, FFT dots) at f0 +/- fs
envelope

Calculates the envelope of a band limited signal
Description

The envelope of an amplitude modulated signal can be calculated by using the Hilbert transform $H(y)$ of the signal or the analytic signal.

Usage

envelope(y)

Arguments

y numeric vector of the signal

Details

An amplitude modulated function $y(x) = A(x) \cdot \cos(\omega \cdot x)$ can be demodulated as follows:

$$A(x)^2 = y(x)^2 + H(y(x))^2$$

If the signal is not band limited, strange things can happen. See the ripple at the edges in the example below. Pay attention, that the envelope is always the real part of the returned value.

Value

real valued envelope function of the signal

Examples

```r
## noisy signal with amplitude modulation
x <- seq(0,1, length.out=2e2)
# original data
y <- (abs(x-0.5))*sin(20*2*pi*x)
ye <- base::Re(envelope(y))
# plot results
plot(x,y,type="l",lwd=1,col="darkgrey",lty=2,ylab="y",main="Spectral filtering")
lines(x,ye)
legend("bottomright",c("modulated","envelope"),col=c("grey","black"),lty=c(2,1))
```
filter.fft

Filter in the frequency domain

Description
This function provides a method to band pass filter in the frequency domain.

Usage
```
filter.fft(
  y = stop("y-value is missing"),
  x = NULL,
  fc = 0,
  BW = 0,
  n = 3,
  type = "poly"
)
```

Arguments
- `y`: numeric data vector
- `x`: optional x-coordinate
- `fc`: center frequency of the bandpass
- `BW`: bandwith of the bandpass
- `n`: parameter to control the stiffness of the bandpass
- `type`: type of weightening function: "poly", "sinc", "bi-cubic", "gauss", can be abbreviated

Details
A signal \( y \) is meant to be equaly spaced and causal, which means it starts at \( t = 0 \). For times \( y < 0 \) the signal is not defined. The filtering itself takes place with the analytic function of \( y \) which provides an one sided spectrum. Applying the Fourier transform, all properties of \( y \) will be preserved.

The band pass is represented throughout a function in the form of four different types, i.e. "poly-nom", "sin(x)/x", "bi-cubic", "gauss". A detailed description about these types can be found in \( \text{BP} \).

Setting \( fc = 0 \) one can achieve a low pass filter.

Examples
```
## noisy signal with amplitude modulation
x <- seq(0,1, length.out=500)
# original data
```
y_org <- (1+sin(2*2*pi*x))*sin(20*2*pi*x)

# overlay some noise
y_noise <- y_org+rnorm(length(x),sd=0.2)

# filter the noisy data
y_filt <- filter.fft(y_noise,x,fc=20,BW=4,n=50)

# plot results
plot(x,y_noise,type="l",lwd=1,col="darkgrey",lty=2,ylab="y",main="Spectral filtering")
lines(x,y_org,lwd=5,col="grey")
lines(x,y_filt)
legend("topright",c("org","noisy","filtered"),col=c("grey","darkgrey","black"),lty=c(1,2,1),lwd=c(5,1,1))

---

filter.lomb

Filter and reconstruction of data analysed via spec.lomb

Description

Given an object of class lomb, this function allows the reconstruction of the input signal using (a) a frequency selection of single or multiple frequency (ranges), and/or (b) the most significant peaks in the periodogram.

Usage

filter.lomb(l=nolomb-Data),
newx = NULL,
threshold = 6,
filt = NULL,
phase = "nextnb"
)

Arguments

l

lomb object
ewnx

vector of new values at which the restored function is to be evaluated
threshold

statistical threshold in terms of a standard deviation of the amplitudes. It determines which frequencies are used. Lower values give more frequencies.
filt

vector or matrix of frequencies (ranges) in which to select the frequencies
phase

set the method to determine the phase at a given frequency
Details

To properly reconstruct the signal out of the calculated lomb-object, three different methods are available, which are controlled by the filt-argument.

1. If filt=NULL, the most significant values in the (dense) spectrum are used.
2. If filt=c(f1, ..., fn), the given frequencies are used. The corresponding phase is approximated.
3. If class(filt)=="matrix", each row of the 2 x n matrix defines a frequency range. With in each range the "significant" frequencies are selected for reconstruction.

Prior to the reconstruction the filter.lomb-function calculates the most significant amplitudes and corresponding phases. As a measure to select the "correct" frequencies, the threshold argument can be adjusted. The corresponding phases of the underlying sine/cosine-waves are estimated by one of the four following methods.

1. phase="nextnb"... use the phase of the bin of nearest neighbour.
2. phase="lin"... linear interpolation between the two closest bins.
3. phase="lockin"... principle of lock-in amplification, also known as quadrature-demodulation technique.
4. phase="fit"... non-linear least squares fit with stats::nls

Value

This function returns a list which contains the reconstruction according to the lomb-object and newx for the given data x and y. The returned object contains the following:

- x, y reconstructed signal
- f, A, phi used parameters from the lomb-object
- p corresponding significance values

Description

calculates the generalized Lomb-Scargle estimation after Zechmeister et al. (2009)

Usage

gLmb(f, dat, w, Y, hYY)

Arguments

- f frequency
- dat spatial vector including locations and values
- w vector of weights
- Y weighted sum of values
- hYY weighted sum of squared values
Details
This method is based on the generalized approach
\[ y(t) = a \cos(w \cdot t) + b \sin(w \cdot t) + c \]
which contains the floating average value \( c \) of the model function above. The calculation is vectorized to enhance calculation speed.

\[ H \]
\( \text{The Hilbert transformation} \)

Description
The Hilbert transform is a phase shifter, which represents the complex complement to a real valued signal. It is calculated in the complex frequency space of the signal by using the Fourier transform. Finally, calculating \( f = y + i \cdot H(y) \) gives the analytic signal, with a one sided spectrum. (See \texttt{analyticFunction})

Usage
\( H(x) \)

Arguments
\( x \quad \text{real valued time series} \)

Value
A numeric real valued vector is returned

interpolate.fft interpolates data using the Fourier back transform

Description
There are two ways to interpolate data from a given spectrum. First, one can do zero padding to cover \( n \) new data points. Or, second the complex amplitude with the associated frequency is taken and evaluated at given points \( x_{\text{out}} \). Doing that for all frequencies and amplitudes will give the interpolation. The result is compared to linear approximation for didactic reasons.

Usage
\( \text{interpolate.fft}(y, x = \text{NULL}, n = \text{NULL}, x_{\text{out}} = \text{NULL}) \)
**Arguments**

- `y` numeric data vector to be interpolated
- `x` numeric data vector with reference points
- `n` number of new points
- `xout` a vector new points

**Value**

A list with a `x` and `y` component is returned. The `e99` value evaluates the error of the interpolation with respect to linear approximation with the `approx()` function.

---

**Description**

calculates the standard Lomb-Scargle estimation. The calculation is vectorized to enhance calculation speed.

**Usage**

```r
lmb(f, dat, var_val)
```

**Arguments**

- `f` frequency
- `dat` spatial vector including locations and values
- `var_val` variance of the data

---

**plot.fft**

Plot `fft`-objects

---

**Description**

This is a wrapper function to plot `fft`-class objects.

**Usage**

```r
## S3 method for class 'fft'
plot(x, ...)
```

**Arguments**

- `x` Object of the class `fft`
- `...` further arguments to the plot functions
See Also

`spec.fft`

Examples

```r
# See spec.fft
```

---

**plot.lomb**  
*plot method for Lomb-Scargle periodograms*

**Description**

This method plots a standard Lomb-Scargle periodogram, which contains the normalized power spectra PSD and the corresponding false alarm probability \( p \). For more details refer to Zechmeister et al. (2009).

**Usage**

```r
## S3 method for class 'lomb'
plot(
  x,
  FAPcol = 1,
  FAPlwd = 1,
  FAPlty = "dashed",
  FAPlim = c(1, 0.001),
  FAPlab = "FAP",
  legend.pos = "topleft",
  legend.cex = 1,
  legend.on = T,
  legend.text = c("Spectrum", "False Alarm Probability"),
  legend.lwd = NULL,
  legend.lty = NULL,
  legend.col = NULL,
  xlab = "Frequency",
  ylab = "Normalized PSD",
  main = "",
  ...
)
```

**Arguments**

- `x`  
  object of class `lomb`
- `FAPcol`  
  color of the FAP line
- `FAPlwd`  
  line width of the FAP line
- `FAPlty`  
  line type for the FAP graph
- `FAPlim`  
  limits to the FAP
print.fft

Details

The `plot.lomb` function is a wrapper function for R’s standard scatter plot. To switch off certain properties, simply overwrite the parameter. For example `log = ""` will reset the plot axis back to non-log scale.

References


See Also

`spec.lomb`

Examples

```r
# See spec.lomb

print.fft

FFT-Plotting Function

Description

It calls the summary function.

Usage

```r
## S3 method for class 'fft'
print(x, ...)
```
print.lomb

Arguments

x lomb object
...

Value

This function returns nothing

Examples

# see summary.lomb() function

Description

It calls the summary function.

Usage

## S3 method for class 'lomb'
print(x, ...)

Arguments

x lomb object
...

Value

This function returns nothing

Examples

# see summary.lomb() function
spec.fft

1D/2D/nD (multivariate) spectrum of the Fourier transform

Description

This function calculates the Fourier spectrum and power spectral density of a given data object. The dimension of the array can be of arbitrary size e. g. 3D or 4D.

Usage

spec.fft(y = NULL, x = NULL, z = NULL, center = T)

Arguments

y  
1D data vector, y coordinate of a 2D matrix, nD (even 2D) array or object of class fft

x  
x-coordinate of the data in y or z. If y is an array, x must be a named list x = list(x = ..., y = ...).

z  
optional 2D matrix

center  
logical vector, indicating which axis to center in frequency space

Details

The function returns an user friendly object, which contains as much frequency vectors as ordinates of the array. spec.fft provides the ability to center the spectrum along multiple axis. The amplitude output is already normalized to the sample count and the frequencies are given in terms of 1/\Delta x-units.

Value

An object of the type fft is returned. It contains the spectrum A, with "reasonable" frequency vectors along each ordinate. psd represents the standardized power spectral density, [0,1]. The false alarm probability (FAP) p is given similar to the Lomb-Scargle method, see spec.lomb.

Missing Values

Given a regular grid \( x_i = \delta x \cdot i \) there might be missing values marked with NA, which are treated by the function as 0's. This "zero-padding" leads to a loss of signal energy being roughly proportional to the number of missing values. The correction factor is then \((1 - N_{na}/N)\) as long as \(N_{na}/N < 0.2\). If the locations of missing values are randomly distributed the implemented procedure works quite robust. If correalted gaps are present, the proposed correction is faulty and scales wrong. This is because a convolution of the incomplete sampling vector with the the signal takes place. An aliasing effect takes place distorting the spectral content.

To be compatible with the underlying Fourier transform, the amplitudes are not affected by this rescaling. Only the power spectral density (PSD) is corrected in terms of the energy content, which is experimental for the moment.
See Also

plot.fft

Examples

# 1D Example with two frequencies
#******************************************************************************

x <- seq(0, 1, length.out = 1e3)
y <- sin(4 * 2 * pi * x) + 0.5 * sin(20 * 2 * pi * x)
FT <- spec.fft(y, x)
par(mfrow = c(2, 1))
plot(x, y, type = "l", main = "Signal")
plot(
    FT,
    ylab = "Amplitude",
    xlab = "Frequency",
    type = "l",
    xlim = c(-30, 30),
    main = "Spectrum"
)
summary(FT)

# 2D example with a propagating wave
#******************************************************************************

x <- seq(0, 1, length.out = 50)
y <- seq(0, 1, length.out = 50)

# calculate the data
m <- matrix(0, length(x), length(y))
for (i in 1:length(x))
    for (j in 1:length(y))
        m[i, j] <- sin(4 * 2 * pi * x[i] + 10 * 2 * pi * y[j])

# calculate the spectrum
FT <- spec.fft(x = x, y = y, z = m)

# plot
par(mfrow = c(2, 1))
rasterImage2(x = x,
    y = y,
    z = m,
    main = "Propagating Wave")
plot(
    FT,
    main = "2D Spectrum",
    palette = "wb",
    xlim = c(-20, 20),
    ylim = c(-20, 20),
    zlim = c(0, 0.51)
spec.fft

xlab = "fx",
ylab = "fy",
zlab = "A",
ndz = 3,
z.adj = c(0, 0.5)
,
z.cex = 1
)

summary(FT)

# 3D example with a propagating wave

# sampling vector
x <- list(x = seq(0,2,by = 0.1)[-1]
   , y = seq(0,1, by = 0.1)[-1]
   , z = seq(0,1, by = 0.1)[-1]

)

# initializing array
m <- array(data = 0,dim = sapply(x, length))

for(i in 1:length(x$x))
   for(j in 1:length(x$y))
      for(k in 1:length(x$z))
         m[i,j,k] <- cos(2*pi*(1*x$x[i] + 2*x$y[j] + 2*x$z[k])) + sin(2*pi*(1.5*x$x[i]))^2

FT <- spec.fft(x = x, y = m, center = c(TRUE,TRUE,FALSE))

par(mfrow = c(2,2))
# plotting m = 0
rasterImage2( x = FT$fx
   , y = FT$fy
   , z = abs(FT$A[,,1])
   , zlim = c(0,0.5)
   , main="m = 0"
)

# plotting m = 1
rasterImage2( x = FT$fx
   , y = FT$fy
   , z = abs(FT$A[,,2])
   , zlim = c(0,0.5)
   , main="m = 1"
)

# plotting m = 2
rasterImage2( x = FT$fx
   , y = FT$fy
   , z = abs(FT$A[,,3])
   , zlim = c(0,0.5)
   , main="m = 2"
rasterImage2( x = FT$fx
, y = FT$fy
, z = abs(FT$A[,4])
, zlim = c(0,0.5)
, main="m = 3"
}

summary(FT)

# calculating the derivative with the help of FFT
# Remember, a signal has to be band limited.
# !!! You must use a window function !!!
#
# preparing the data
x <- seq(-2, 2, length.out = 1e4)
dx <- mean(diff(x))
y <- win.tukey(x) * (-x ^ 3 + 3 * x)

# calculating spectrum
FT <- spec.fft(y = y, center = TRUE)
# calculating the first derivative
FT$A <- FT$A * 2 * pi * 1i * FT$fx
# back transform
dm <- spec.fft(FT)

# plot
par(mfrow=c(1,1))
plot(
x,
c(0, diff(y) / dx),
type = "l",
col = "grey",
lty = 2,
ylim = c(-4, 3)
)
# add some points to the line for the numerical result
points(approx(x, Re(dm$y) / dx, n = 100))
# analytical result
curve(-3 * x ^ 2 + 3,
    add = TRUE,
    lty = 3,
    n = length(x))

legend(
    "topright",
c("analytic", "numeric", "spectral"),
title = "diff",
lty = c(3, 2, NA),
spec.lomb

pch = c(NA, NA, 1),
col=c("black","grey","black")
)
title(expression(d / dx ~ (-x ^ 3 + 3 * x)))

---

**Lomb-Scargle Periodogram**

**Description**

The Lomb-Scargle periodogram represents a statistical estimator for the amplitude and phase at a given frequency. This function takes also multivariate (n-dimensional) input data.

**Usage**

```r
spec.lomb(
  x = NULL,
  y = stop("Missing y-Value"),
  f = NULL,
  ofac = 1,
  w = NULL,
  mode = "normal",
  maxMem = 8,
  cl = NULL
)
```

**Arguments**

- `x` sampling vector or data frame `data.frame(x1, x2, x3, ...)`
- `y` input data vector or data frame `data.frame(x1, x2, ..., val)`
- `f` optional frequency vector / data frame. If not supplied `f` is calculated.
- `ofac` in case `f=NULL` this value controls the amount of frequency oversampling.
- `w` weights for data. It must be a 1D vector.
- `mode` "normal" calculates the normal Lomb-Scargle periodogram; "generalized" calculates the generalized Lomb-Scargle periodogram including floating average and weights.
- `maxMem` sets the amount of memory (in MB) to utilize, as a rough approximate.
- `cl` if numeric, it defines the number of workers to use, or provides a cluster definition of class `cluster` or `SocketCluster` from `parallel` package.
Details

Since the given time series does not need to be evenly sampled, the data mainly consists of data pairs \(x_1, x_2, x_3, \ldots\) (sampling points) and (one) corresponding value \(y\), which stores the realisation/measurement data. As can be seen from the data definition above, multivariate (n-dimensional) input data is allowed and properly processed.

Two different methods are implemented: the standard Lomb-Scargle method with

\[ y(t) = a \ast \cos(\omega(t - \tau)) + b \ast \sin(\omega(t - \tau)) \]

as model function and the generalized Lomb-Scargle (after Zechmeister 2009) method with

\[ y(t) = a \ast \cos(\omega t) + b \ast \sin(\omega t) + c \]

as model function, which investigates a floating average parameter \(c\) as well.

Both methods can be supplied by an artificial dense frequency vector \(f\). In conjunction with the resulting phase information the user might be able to build a "Fourier"-like spectrum to reconstruct or interpolate the timeseries in equally spaced sampling. Remind the band limitation which must be fulfilled for this.

\( f\) The frequencies should be stored in a 1D vector or – in case of multivariate analysis – in a data.frame structure to preserve variable names

\( \text{ofac}\) If the user does not provide a corresponding frequency vector, the \(\text{ofac}\) parameter causes the function to estimate

\[ nf = \text{ofac} \ast \text{length}(x)/2 \]

equidistant frequencies.

\(\text{p-value}\) The p-value (aka false alarm probability FAP) gives the probability, whether the estimated amplitude is NOT significant. However, if \(p\) tends to zero the amplitude is significant. The user must decide which maximum value is acceptable, until an amplitude is not valid.

If missing values \(\text{NA}\) or \(\text{NaN}\) appear in any column, the corresponding row is excluded from calculation.

Value

The \texttt{spec.lomb} function returns an object of the class \texttt{lomb}, which is a list containing the following information:

\( A\) A vector with amplitude spectrum
\( f\) corresponding frequency vector
\( \phi\) phase vector
\( \text{PSD}\) power spectral density normalized to the sample variance
\( \text{floatAvg}\) floating average value only in case of \(\text{mode} == \text{"generalized"}\)
\( w\) if \(\text{mode} == \text{"generalized"}\) contains the weighting vector
\( x, y\) original data
\( p\) p-value False Alarm Probability
**Speed Up**

In general the function calculates everything in a vectorized manner, which speeds up the procedure. If the memory requirement is more than `maxMem`, the calculation is split into chunks which fit in the memory (cache). Depending on the problem size (number of frequencies and data size) a tuning of this value enhances speed.

Please consider to replace the BLAS library by a multithreaded version. For example [https://prs.ism.ac.jp/~nakama/SurviveGotoBLAS2/binary/windows/x64/](https://prs.ism.ac.jp/~nakama/SurviveGotoBLAS2/binary/windows/x64/) is hosting some Windows RBlas.dll files. Refer to [https://mattstats.wordpress.com/2016/02/07/r-with-gotoblas-on-windows-10/](https://mattstats.wordpress.com/2016/02/07/r-with-gotoblas-on-windows-10/) for further information.

The parameter `cl` controls a possible cluster, which can be invoked. It takes an integer number of workers (i.e. `cl = 4`), a list with node names `c("localhost",...)` or an object of class `"cluster"` or similar. The first two options cause the function to create the cluster internally. This takes time due to the initialization. The faster way is to provide an already initialized cluster to the function.

**References**


**See Also**

`filter.lomb`

**Examples**

```r
# create two sin-functions
x_orig <- seq(0,1,by=1e-2)
y_orig <- 2*sin(10*2*pi*x_orig) + 1.5*sin(2*2*pi*x_orig)

# make a 10% gap
i <- round(length(x_orig)*0.2) : round(length(x_orig)*0.3)
x <- x_orig
y <- y_orig
x[i] <- NA
y[i] <- NA

# calculating the lomb periodogram
l <- spec.lomb(x = x, y = y, ofac = 20, mode = "normal")

# select a frequency range
m <- rbind(c(9,11))
# select and reconstruct the most significant component
```
l2 = filter.lomb(l, x_orig, filt = m)

# plot everything
par(mfrow=c(2,1),mar = c(4,4,2,4))
plot(x,y,"l", main = "Gapped signal")
lines(l2$x, l2$y,lty=2)
legend("bottomleft",c("gapped","10Hz component"),lty=c(1,2))

plot(l,main = "Spectrum")

summary(l)

### Multivariate -- 3D Example ###
require(lattice)
fx <- 8.1
fy <- 5
fz <- 2

# creating frequency space
f <- expand.grid( fx = seq(-10,10,by = 0.5)
                ,fy = seq(-10,10,by = 0.5)
                ,fz = 0:3)

# creating spatial space
pts <- expand.grid( x = seq(0,1,by = 0.02)
                    ,y = seq(0,1,by = 0.02)
                    ,z = seq(0,1,by = 0.02))

# gapping 30%
i <- sample(1:dim(pts)[1],0.7*dim(pts)[1])
pts <- pts[i,]

# caluculating function
pts$val <- cos(2*pi*( fx*pts$x
                     + fy*pts$y
                     + fz*pts$z
                     ) + pi/4
                  ) +
0.5 * cos(2*pi*( - 0.5 * fx*pts$x
                     + 0.5*fy*pts$y
                     + 1 * pts$z
                     ) + pi/4
                  )

# display with lattice
levelplot(val~x*y,pts,subset = z == 0,main = "with z = 0")

# calculating lomb takes a while
# or we sample only a few points
# which enlarges the noise but accelerates the calculation
l <- spec.lomb(y = pts[sample(1:dim(pts)[1],2e3),]
# name the stripes
l$fz_lev <- factor(x = paste("fz =",l$fz))

# display output
levelplot(PSD~fx+fy|fz_lev,l)

# the result is an oversampled spectrum of a non equidistant
# sampled function. We recognize a 3D analysis in all provided
# spatial directions x, y, z.

summary(l)

---

**summary.fft**  
*Summarize FFT objects*

**Description**

The function summarizes properties from the class(fft) object.

**Usage**

```r
## S3 method for class 'fft'
summary(object, p0 = 0.01, ...)
```

**Arguments**

- `object` lomb object
- `p0` False Alarm Probability (FAP) threshold, default 1%
- `...` not used

**Details**

The false alarm probability threshold p0 value can be changed to modify the amount of significant peaks.

**Value**

a list of significant values of the spectral analysis

**Examples**

# see spec.fft() example
**summary.lomb**

### Summarize Lomb objects

**Description**

The function summarizes properties from the Lomb object.

**Usage**

```r
## S3 method for class 'lomb'
summary(object, p0 = 0.01, ...)
```

**Arguments**

- `object`: lomb object
- `p0`: False Alarm Probability threshold, default 1%
- `...`: not used

**Details**

The false alarm probability threshold p0 value will adjust the number of peaks.

The `effectiveBandWidth` describes the coverage of processed frequencies by the `spec.lomb` function. If the ratio to `averageSampling` is almost 2, then the Nyquist criterion can be assumed to be fulfilled. If the ratio is much less than 2 then only a fraction of information is analysed.

The `minFreqStep` is an estimate of the minimum frequency step determined from the Lomb-Object.

Average sampling is calculated from the median distance between two spatial points.

The possible frequency resolution originates also from the spatial (temporal) input data by `1/(diff(range(x)))`, if `x` is the spatial (temporal) coordinate.

**Value**

A list of significant values of the spectral analysis

**Examples**

```r
# see spec.lomb() example
```
Description

A waterfall-diagramm displays the local frequency in dependence of or spatial vector. One can then locate an event in time or space.

Usage

```r
waterfall(
  y = stop("y value is missing"),
  x = NULL,
  nf = 3,
  type = "b",
  width = 7
)
```

Arguments

- **y**: numeric real valued data vector
- **x**: numeric real valued spatial vector. (time or space)
- **nf**: steepness of the bandpass filter, degree of the polynomial.
- **type**: type of weightening function: "poly", "sinc", "bi-cubic", "gauss", can be abbreviated
- **width**: normalized maximum "inverse" width of the bandpass $bw = fc/width$.

Details

Each frequency is evaluated by calculating the amplitude demodulation, which is equivalent to the envelope function of the band pass filtered signal. The frequency of interest defines automatically the center frequency $fc$ of the applied band pass with the bandwidth $BW$:

$$BW = fc/width, BW < width \rightarrow BW = width, BW > width \rightarrow BW = fc/width$$

The frequency is normalized so the minimal frequency is 1. With increasing frequency the bandwidth becomes wider, which lead to a variable resolution in space and frequency. This is comparable to the wavelet (or Gabor) transform, which scales the wavelet (window) according to the frequency. However, the necessary bandwidth is changed by frequency to take the uncertainty principle into account. Slow oscillating events are measured precisely in frequency and fast changing processes can be determined more exact in space. This means for a signal with steady increasing frequency the waterfall function will produce a diagonally stripe. See the examples below.

Value

a special `fft`-object is returned. It has mode "waterfall" and x and fx present, so it is only plotable.
### Missing values

Given a regular grid \( x_i = \delta x \cdot i \) there might be missing values marked with NA, which are treated by the function as 0’s. This “zero-padding” leads to a loss of signal energy being roughly proportional to the number of missing values. The correction factor is then \( 1 - N_{na}/N \) as long as \( N_{na}/N < 0.2 \). As long as the locations of missing values are randomly distributed the implemented procedure works quite robust. If, in any case, the distribution becomes correlated the proposed correction is faulty and projects the wrong energies.

The amplitudes and PSD values are compensated to show up an estimate of the "correct" value. Therefore this method is experimental.

### Examples

#### noisy signal with amplitude modulation ####

```r
x <- seq(0,3, length.out = 1000)
# original data
# extended example from envelope function
y <- 1*(abs(x-1.5))*sin(10*2*pi*x) + ifelse(x > 1.5,sin(15*(1+0.25*(x - 1.5))*2*pi*x),0)
ye <- base::Re(envelope(y))
par(mfrow=c(2,1),mar=c(1,3.5,3,3),mgp=c(2.5,1,0))
# plot results
plot(x,y,type="l",lwd=1,col="darkgrey",lty=2,ylab="y",main="Original Data",xaxt="n",xlab="")
lines(x,ye)
legend("bottomright",c("modulated","envelope"),col=c("grey","black"),lty=c(2,1))
par(mar=c(3.5,3.5,2,0))
wf <- waterfall(y,x,nf = 3)
# rasterImage2(x = wf$x, y = wf$fx, z = wf$A
# ,ylim = c(0,60))
plot(wf,ylim=c(0,40),main="Waterfall")
```

#### uncertainty principle ####

```r
# take a look at the side effects
# at [0,30] and [1,0]
# With a large steepness e.g. n = 50 you will gain
# artefacts.
# if frequency is not stationary
# PSD becomes > 1 depending on the type of band filter.
#
# x <- seq(0,1, length.out=1500)
y <- sin(100*x*x)
FT <- spec.fft(x = x, y = y)
wf <- waterfall(y,x)
```
win.cos

Cosine window function

Description
This window function returns a vector of weights with means of a cosine window.
Usage

win.cos(n)

Arguments

n  data vector to be windowed

See Also

Windowfunctions

---

win.hann  Hanning window function

Description

This window function returns a vector of weights with means of a generalized Hann-window.

Usage

win.hann(n, a = 2)

Arguments

n  data vector to be windowed
a  order of the window, default a = 2

See Also

Windowfunctions

---

win.nutt  Nuttall window function

Description

This window function returns a vector of weights with means of a Nuttall-window.

Usage

win.nutt(n, a = c(0.355768, 0.487396, 0.144232, 0.012604, 0))

Arguments

n  data vector to be windowed
a  coefficients default a = c(0.355768, 0.487396, 0.144232, 0.012604, 0)
**Details**

This window function provides a continuous first derivative everywhere, like the Hann window. Adopted from the idea of Hann this window consists of up to 5 trigonometric polynomial terms, i.e.

\[ w_n = a_1 - a_2 \cos(2\pi n / M) + a_3 \cos(4\pi n / M) - a_4 \cos(6\pi n / M) + a_5 \cos(8\pi n / M) \]

Different sets of coefficients:

- **Nuttall (Default)**: \( c(0.355768, 0.487396, 0.144232, 0.012604, 0) \)
- **Blackman-Nuttall**: \( c(0.3635819, 0.4891775, 0.1365995, 0.0106411, 0) \)
- **Blackman-Harris**: \( c(0.35875, 0.48829, 0.14128, 0.01168, 0) \)
- **Flat-Top**: \( c(0.211557895, 0.41663158, 0.277263158, 0.083578947, 0.006947368) \)

**See Also**

- Windowfunctions

---

**Description**

This window function returns a vector of weights with means of a Tukey-window. In contrast to a cosine window this function is more steep at the beginning and the end. And it is 1 in the middle.

**Usage**

```
win.tukey(n, a = 0.5)
```

**Arguments**

- `n`: data vector to be windowed
- `a`: width of the rising and falling edge as ratio of the total data length

**See Also**

- Windowfunctions
Description

Some typical window functions are defined below:

Details

- `win.cos()` cosine window
- `win.tukey()` Tukey window
- `win.hann()` Hann window
- `win.nutt()` Nutt window

A window function weights a given dataset in a way, that the new data set is coerced to be periodic. This method reduces the leakage effects of the discrete Fourier transform.

Value

All window functions return a weighting vector with the same length as the provided data vector.

Examples

```r
y <- 1:100
y_cos <- y * win.cos(y)
y_tuk <- y * win.tukey(y)
y_han <- y * win.hann(y)

# Plot the original data
plot(y, main="Effect of window functions")
legend("topleft", c("original", "cos", "tukey", "han"), pch=c(1, 16, 17, 18))
points(y_cos, pch=16)
points(y_tuk, pch=17)
points(y_han, pch=18)
```
Index

* Fourier Transform
  spec.fft, 19
* Lomb Scargle
  spec.lomb, 23
* Periodogram
  spec.lomb, 23
* convolution
  deconvolve, 7
* deconvolution
  deconvolve, 7
* power spectral density
  deconvolve, 7
  .onAttach, 2
  .onDetach, 3
  amax, 3
  analyticFunction, 4, 14

BP, 5, 11
  deconvolve, 7
  envelope, 4, 9
  filter.fft, 4, 11
  filter.lomb, 12, 25
  gLmb, 13
  H, 14
  interpolate.fft, 14
  lmb, 15
  plot.fft, 15, 20
  plot.lomb, 16
  print.fft, 17
  print.lomb, 18
  spec.fft, 4, 16, 19
  spec.lomb, 17, 19, 23

summary.fft, 27
summary.lomb, 28
waterfall, 29
win.cos, 31
win.hann, 32
win.nutt, 32
win.tukey, 33
Windowfunctions, 32, 33, 34