Package ‘spectral’

September 15, 2016

Type Package

Title Common Methods of Spectral Data Analysis

Version 1.0.1

Author Martin Seilmayer

Maintainer Martin Seilmayer <m.seilmayer@hzdr.de>

Description Fourier and Hilbert transforms are utilized to perform several types of spectral analysis on the supplied data. Also fragmented and irregularly spaced data can be processed. A user friendly interface helps to interpret the results.

License GPL-2

Depends rasterImage, R (>= 2.15.0)

LazyData TRUE

RoxygenNote 5.0.1

LinkingTo Rcpp

Imports Rcpp

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-09-15 15:46:39

R topics documented:

`analyticFunction` *Analytic function*

Description

Normaly 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. This is called analytic signal or analytic function. The result is a one sided spectrum.

Usage

```
analyticFunction(x)
```

Arguments

`x`	real valued data vector

Details

An analytic function xa is composed of the real valued signal representation y and its Hilber transform $H(y)$ as the complex complement

$$xa(t) = x(t) + iH(x(t)).$$

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 easy in the spectral space, compared to the standard FFT approach. Some examples are:

Filtering because the spectrum is one sided, the user must only modifiy 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 bandlimited 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 certered 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 and Fritzsche 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

R. Hoffmann, Signalanalyse und -erkennung: eine Einfuehrung fuer Informationstechniker. Berlin; Heidelberg: Springer, 1998.

H. Marko, Systemtheorie: Methoden und Anwendungen fuer ein- und mehrdimensionale Systeme. 3. Aufl., Berlin: Springer, 1995.

G. Fritzsche, Signale und Funktionaltransformationen - Informationselektronik. Berlin: VEB Verlag Technik, 1985

BP *Simple bandpass function*

Description

This function represents a simple weighting function for spectral filtering.

Usage

```
BP(f, fc, BW, n = 3)
```

Arguments

`f`	vector of frequencies
`fc`	center frequency
`BW`	bandwidth, with `w[f < (fc - BW) | f > (fc+BW)] == 0`
`n`	degree of the polynom, `n` can be real, e.g. `n = 2.5`

Details

The band pass is represented troughout a polynom in the form

$$w = 1 - a * (f - fc)^n$$

with the degree n. The parameter fc controlls the center frequency and a scales the required band width `BW`. outside the band width the result is forced to zero.

Value

This function returns a weight vector, which is to apply to the frequency vector `f` in a top level function 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 of the signal or the analytic signal.

Usage

```
envelope(y)
```

Arguments

y	numeric vector of the signal

Details

A modulated function $y(x) = A(x) * cos(\omega * x)$ 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

```
## 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 bandpass filter in the frequency domain.

Usage

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

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

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 applied bandpass filter function is a simple polynomial approach, which weights the frequencies. 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 = stop("No Lomb-Data"), newx = NULL, threshold = 6,
  filt = NULL, phase = "nextnb")
```

Arguments

`l`	lomb object
`newx`	vector of new values at which the restored function is to be evaluated
`threshold`	statistical threshold in terms of a standard deaviation of the amplidudes. It determines which frequencies are used. Lower values give more frequencies.
`filt`	vector or matix of frequencies (ranges) in which to select the frequencies
`phase`	set the method to determine the phase at a given frequency (moegliche werte???)

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

`H` *The Hilbert-transformation*

Description

The Hilbert-transform is a phase shifter, which represents the complex complement to a real vauled signal. It is calculated in the complex frequency space of the signal by using the Fourier transform. Finally, calculating $f = y + i * H(y)$ gives the analytic signal, with a one sided spectrum. (See `analyticFunction`)

Usage

```
H(x)
```

Arguments

`x`	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 way to interpolate data from a given spectrum. Frist, one can do zero padding to cover `n` new data points. Or, secound the complex amplitude with the associated frequency is taken and evaluated at given points `xout`. Doing that for all frequencies and amplitudes will give the interpolation.

Usage

```
interpolate.fft(y, x = NULL, n = NULL, xout = 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.

`lmb` *Lomb-Scargle estimation function*

Description

calculates the Lomb-Scargle estimation

Usage

```
lmb(x, y, omega)
```

Arguments

`x`	spatial vector
`y`	data vector
`omega`	frequency vector

Details

This wrapper function calls compiled C++ code, which does the job.

`plot.fft` *Plot `fft`-objects*

Description

Plot `fft`-objects

Usage

```
## 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

```
# See spec.fft
```

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

Description

This method plots a standard Lomb-Scargle periodogram, which contains the amplitude `A` and the false alarm probability `p`.

Usage

```
## 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 Propability"), legend.lwd = NULL, legend.lty = NULL,
  legend.col = NULL, xlab = "Frequency", ylab = "Amplitude", 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
`FAPlab`	label of the right vertical axis
`legend.pos`	position of the legend
`legend.cex`	cex value for the legend
`legend.on`	logical, wheater to draw a legend or not
`legend.text`	legend text
`legend.lwd`	line width

`legend.lty`	line type
`legend.col`	color vector of the legend elements
`xlab`	a label for the x axis, defaults to a description of `x`.
`ylab`	a label for the y axis, defaults to a description of `y`.
`main`	setting the title of the plot
`...`	further parameters to the plot function

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.

See Also

`spec.lomb`

Examples

```
# See spec.lomb
```

`spec.fft` *1D/2D spectrum of the Fourier transform*

Description

This function calculates the Fourier spectrum of a given 1D or 2D data object. It returns a user friendly object, which contains one or two frequency vectors to map the complex amplitudes (vector or matirx) to the corresponding frequencies. The output is already normalized and the frequencies can be seen in terms of $1/\Delta x$-units.

Usage

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

Arguments

`y`	1D data vector, y coordinate of a 2D matrix or object of class `fft`
`x`	x-coordinate of the data in y vector or z matrix
`z`	optional 2D matrix
`center`	logical parameter, if the spectrum should be centered or not
`inverse`	logical parameter, if the back transformation should be performed

Value

An object of the type `fft` is returned. This contains the original dataset and the corresponding spectrum, with "reasonable" frequency vectors.

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"
)

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

x <- seq(0, 1, length.out = 1e2)
y <- seq(0, 1, length.out = 1e2)

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


# 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)

# calcualting 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, inverse = TRUE)

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

`spec.lomb` *Lomb-Scargle Periodigram*

Description

The Lomb-Scargle periodigram represents an statistical estimator for the amplitude and phase for a given frequency.

Usage

```
spec.lomb(y = stop("Missing y-Value"), x = stop("Missing x-Value"),
  f = NULL, ofac = 4)
```

Arguments

`y`	data vector
`x`	sampling vector
`f`	frequeny vector
`ofac`	in case `f=NULL` this value controlls the amount of frequency oversampling.

Details

A given time series does not need to be evenly sampled. This means a time series mainly consists of data pairs `x` and `y`, which store the data and the sampling position (e.g. in time). Additionally, this method enables the user to analyse the data with respect to a given frequency vector, which can be artifical dense.

`ofac` If the user does not provide a corresponding frequency vector, the `ofac` parameter causes the function to estimate

$$nf = ofac * length(x)/2$$

equidistant frequencies.

`p`-value The `p`-value gives the probability, wheater the estimated amplitude is NOT significant. However, if `p` tends to zero the amplidutde is significant. The user must decide which maximum value is acceptable, until an amplitude is not valid.

Value

The `spec.lomb` function returns an object of the type `lomb`, which is a `list` containing the following parameters:

`A` A vector with amplitude spectrum

`f` corresponding frequency vector

`phi` phase vector

`x,y` original data

p p-value as statistical measure

References

A. Mathias, F. Grond, R. Guardans, D. Seese, M. Canela, H. H. Diebner, und G. Baiocchi, "Algorithms for spectral analysis of irregularly sampled time series", Journal of Statistical Software, 11(2), pp. 1–30, 2004.

J. D. Scargle, "Studies in astronomical time series analysis. II - Statistical aspects of spectral analysis of unevenly spaced data", The Astrophysical Journal, 263, pp. 835–853, 1982.

See Also

`filter.lomb`

Examples

```
# create two sin-functions
x_orig <- seq(0,1,by=1e-2)
y_orig <- sin(10*2*pi*x_orig) + 0.3*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)
# 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")
```

waterfall *Estimate the local frequencies*

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

```
waterfall(y = stop("y value is missing"), x = NULL, nf = 3, width = 10)
```

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.
`width`	normalized (to df) maximum width of the bandpass.

Details

Each frequency is evaluated by calculating the demodulation. This is equivalent to the envelope function of the bandpass filtered signal. The frequency of interest defines automatically the center frequency of the applied bandpass with the bandwidth BW:

$$BW = f0/4, BW < 4df- > BW = 4df, BW > width * df- > BW = width * df$$

The minimal frequency is df and $f0$ denotes the center frequency of the bandpass. With increasing frequency the bandwidth becomes wider, which lead to a variable resolution in space and frequency. This is comparable to the wavelet transform, which scales the wavelet 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.

Examples

```
## noisy signal with amplitude modulation
x <- seq(0,1, length.out=1000)
# original data
# extended example from envelope function
y <- 2*(abs(x-0.5))*sin(10*2*pi*x) + ifelse(x > 0.5,sin(10*(1+2*(x - 0.5))*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)
plot(wf,ylim=c(0,40),main="Waterfall")


## uncertainty principle
#
# 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.
#
x <- seq(0,1, length.out=500)
y <- sin(100*x*x)

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="")

par(mar=c(3.5,3.5,2,0))
wf <- waterfall(y,x)
rasterImage2(x = wf$x, y= wf$fx,z=wf$A,ylim=c(0,40),main="Waterfall")
```

`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.tukey` *Tukey window function*

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`

`Windowfunctions` *Windowfunctions*

Description

Some typical windowfunctions are defined below:

Details

`win.cos()` cosine window

`win.tukey()` the tukey 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 Fourier transform.

Value

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

Examples

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

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

Index