Package ‘seewave’

March 2, 2018

Type Package

Title Sound Analysis and Synthesis

Version 2.1.0

Date 2018-03-02

Author Jerome Sueur <sueur@mnhn.fr> [cre, au], Thierry Aubin [au],
Caroline Simonis [au], Laurent Lellouch [main ctbr],
Ethan C. Brown [ctbr], Marion Depraetere [ctbr],
Camille Desjonqueres [ctbr], Francois Fabianek [ctbr],
Amandine Gasc [ctbr], Eric Kasten [ctbr], Stefanie LaZerte [ctbr], Jonathan Lees [ctbr],
Jean Marchal [ctbr], Sandrine Pavoine [ctbr], David Pinaud [ctbr], Alicia Stotz [ctbr],
Luis J. Villanueva-Rivera [ctbr], Zev Ross [ctbr],
Carl G. Witherott [ctbr], Hristo Zhivomirov [ctbr].

Maintainer Jerome Sueur <sueur@mnhn.fr>

Encoding UTF-8

SystemRequirements LIBSNDFILE

Imports graphics, grDevices, stats, utils, tuneR

Suggests audio, fftw, ggplot2, rgl, rpanel, phonTools, signal

ZipData no

Description Functions for analysing, manipulating, displaying, editing and synthesizing time waves (particularly sound). This package processes time analysis (oscillograms and envelopes), spectral content, resonance quality factor, entropy, cross correlation and autocorrelation, zero-crossing, dominant frequency, analytic signal, frequency coherence, 2D and 3D spectrograms and many other analyses.

License GPL (>= 2)

URL http://rug.mnhn.fr/seewave

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-03-02 16:49:05 UTC
R topics documented:

ACI .................................................. 4
acoustat ........................................... 6
addsilw ............................................. 8
afilter ............................................... 9
akamatsu .......................................... 10
ama .................................................. 12
AR .................................................... 13
attenuation ........................................ 15
autoc ............................................... 16
bwfilter .......................................... 17
ccoh ............................................... 19
ceps ............................................... 21
cepstro .......................................... 23
coh ............................................... 25
combfilter ....................................... 26
convSPL .......................................... 28
corenv .......................................... 29
corspec .......................................... 30
covspec .......................................... 33
crest ............................................ 35
csh ................................................ 36
cutspec .......................................... 37
cutw ............................................. 39
dBscale .......................................... 40
dBweight ......................................... 41
deletew .......................................... 43
dfreq ............................................ 44
diffcumspec .................................... 46
diffenv .......................................... 48
diffspee .......................................... 49
diffwave .......................................... 51
discrets .......................................... 53
drawnenv ........................................ 54
drawfilter ....................................... 56
duration .......................................... 57
dynoscello ....................................... 58
dynspec .......................................... 59
dynspectro ...................................... 61
echo ............................................... 64
env ............................................... 65
export ........................................... 67
fadew ............................................ 68
fbands .......................................... 69
fdoppler ........................................ 71
filter ............................................ 73
field ............................................. 74
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fir</td>
<td>76</td>
</tr>
<tr>
<td>fma</td>
<td>77</td>
</tr>
<tr>
<td>fpeaks</td>
<td>79</td>
</tr>
<tr>
<td>ftwindow</td>
<td>81</td>
</tr>
<tr>
<td>fund</td>
<td>83</td>
</tr>
<tr>
<td>ggspectro</td>
<td>84</td>
</tr>
<tr>
<td>H</td>
<td>86</td>
</tr>
<tr>
<td>hilbert</td>
<td>87</td>
</tr>
<tr>
<td>ifreq</td>
<td>88</td>
</tr>
<tr>
<td>istft</td>
<td>90</td>
</tr>
<tr>
<td>itakura.dist</td>
<td>92</td>
</tr>
<tr>
<td>kl.dist</td>
<td>93</td>
</tr>
<tr>
<td>ks.dist</td>
<td>94</td>
</tr>
<tr>
<td>lfs</td>
<td>96</td>
</tr>
<tr>
<td>listen</td>
<td>97</td>
</tr>
<tr>
<td>localpeaks</td>
<td>98</td>
</tr>
<tr>
<td>logspec.dist</td>
<td>100</td>
</tr>
<tr>
<td>M</td>
<td>101</td>
</tr>
<tr>
<td>meandB</td>
<td>103</td>
</tr>
<tr>
<td>meanspec</td>
<td>104</td>
</tr>
<tr>
<td>mel</td>
<td>106</td>
</tr>
<tr>
<td>melfilterbank</td>
<td>108</td>
</tr>
<tr>
<td>micsens</td>
<td>109</td>
</tr>
<tr>
<td>moredB</td>
<td>110</td>
</tr>
<tr>
<td>mutew</td>
<td>111</td>
</tr>
<tr>
<td>NDSI</td>
<td>112</td>
</tr>
<tr>
<td>noisew</td>
<td>113</td>
</tr>
<tr>
<td>notefreq</td>
<td>114</td>
</tr>
<tr>
<td>octaves</td>
<td>115</td>
</tr>
<tr>
<td>orni</td>
<td>116</td>
</tr>
<tr>
<td>oscillo</td>
<td>117</td>
</tr>
<tr>
<td>oscilloST</td>
<td>120</td>
</tr>
<tr>
<td>pastew</td>
<td>121</td>
</tr>
<tr>
<td>peewit</td>
<td>123</td>
</tr>
<tr>
<td>pellucens</td>
<td>123</td>
</tr>
<tr>
<td>phaseplot</td>
<td>124</td>
</tr>
<tr>
<td>phaseplot2</td>
<td>125</td>
</tr>
<tr>
<td>playlist</td>
<td>127</td>
</tr>
<tr>
<td>preemphasis</td>
<td>128</td>
</tr>
<tr>
<td>pulsew</td>
<td>129</td>
</tr>
<tr>
<td>Q</td>
<td>130</td>
</tr>
<tr>
<td>repw</td>
<td>132</td>
</tr>
<tr>
<td>resamp</td>
<td>133</td>
</tr>
<tr>
<td>revw</td>
<td>134</td>
</tr>
<tr>
<td>rmam</td>
<td>135</td>
</tr>
<tr>
<td>rrmnoise</td>
<td>136</td>
</tr>
<tr>
<td>rmoffset</td>
<td>137</td>
</tr>
<tr>
<td>rms</td>
<td>138</td>
</tr>
</tbody>
</table>
This function computes the Acoustic Complexity Index (ACI) as described in Pieretti et al. (2011)

Usage

ACI(wave, f, wl = 512, ovlp = 0, wn = "hamming", flim = NULL, nbwindows = 1)
Arguments

- **wave**: an R object.
- **f**: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- **wl**: window length for the analysis (even number of points) (by default = 512)
- **ovlp**: overlap between two successive windows (in %).
- **wn**: window name, see `ftwindow` (by default "hanning").
- **flim**: a numeric vector of length 2 to select a frequency band (in kHz).
- **nbwindows**: a numeric vector of length 1 specifying the number of windows (by default 1, ie a single window including the complete `wave` object).

Details

The function computes first a short-term Fourier transform and then the ACI index. The function returns only the ACI total, ACI tot in Pieretti *et al.* (2010). See the references for details on computation.

Value

A vector of length 1 returning the ACI total.

Note

Values returned were checked with the results provided by the add-on Soundscapemeter for the software Wavesurfer [http://www.speech.kth.se/wavesurfer/](http://www.speech.kth.se/wavesurfer/).

Author(s)

Laurent Lellouch, improved by Amandine Gasc and Morgane Paprin

References


See Also

- `spectro`

Examples

```r
data(tico)
ACI(tico)
## dividing the sound sample into 4 windows of equal duration
ACI(tico, nbwindows=4)
## selection of a frequency band
ACI(tico, flim=c(2,6))
```
acoustat  

Statistics on time and frequency STFT contours

Description

This function returns statistics based on STFT time and frequency contours.

Usage

```r
acoustat(wave, f, wl = 512, ovlp = 0, wn = "hanning", 
         tlim = NULL, flim = NULL, 
         aggregate = sum, fraction = 90, 
         plot = TRUE, type = "l", ...)
```

Arguments

- `wave` an R object.
- `f` sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl` window length for the analysis (even number of points) (by default = 512).
- `ovlp` overlap between two successive windows (in %).
- `wn` window name, see `ftwindow` (by default "hanning").
- `tlim` modifications of the time limits of the analysis (in s).
- `flim` modifications of the frequency limits of the analysis (in kHz).
- `aggregate` a character vector of length 1 specifying the function to be applied on the rows (time) and columns (frequency) of the STFT matrix. By default set to `sum`.
- `fraction` a numeric vector of length 1, specifying a particular fraction of the contours amplitude to be captured by the initial and terminal percentile values (in %). See details.
- `plot` a logical, if TRUE a two-frame plot is returned with the time and frequency contours and percentiles displayed.
- `type` if `plot` is TRUE, type of plot that should be drawn. See `plot` for details (by default "l" for lines).
- `...` other `plot` graphical parameters.

Details

The principle of acoustat is as follows:

1. Compute the short-term Fourier transform (STFT) with usual parameters (wl for window length, ovlp for overlap of successive windows, and wn for the name of window shape).
2. This results in a time * frequency matrix.
3. Compute an aggregation function (specified with the argument `aggregate` set by default to `sum`) accross rows and columns of time * frequency matrix.)
4. This results in two components: (i) the time contour, and (ii) the frequency contour.

5. Each contour is considered as a probability mass function (PMF) and transformed into a cumulated distribution function (CDF).

6. Measures are extracted from each CDF: median (M), initial percentile (P1) value, terminal percentile (P2) value, interpercentile range (IPR). P1, P2 and IPR are defined using a fraction parameter (fraction) that sets the percent of the contour amplitude to be captured by the initial and terminal percentile values. A fraction of 50% would result in the familiar quartiles and interquartile range. An energy fraction of 80% would return the 10th and 90th percentile values, and the width of the range in between.

**Value**

The function returns a list with 10 items:

- **time.contour** the time contour as a two-column matrix, the first column being time (s) and the second column being the amplitude probability mass function (no scale).
- **freq.contour** the frequency contour as a two-column matrix, the first column being frequency (kHz) and the second column being the amplitude probability mass function (no scale).
- **time.P1** the time initial percentile
- **time.M** the time median
- **time.P2** the time terminal percentile
- **time.IPR** the time interpercentile range
- **freq.P1** the frequency initial percentile
- **freq.M** the frequency median
- **freq.P2** the frequency terminal percentile
- **freq.IPR** the frequency interpercentile range

**Note**

`acoustat` was originally developed in Matlab language by Kurt Fristrup and XXXX Watkins (1992).

The R function was kindly checked by Kurt Fristrup.

**Author(s)**

Jerome Sueur

**References**


**See Also**

`meanspec`, `specprop`
Examples

data(tico)
note <- cutw(tico, from=0.5, to=0.9, output="Wave")
## default setting
acoustat(note)
## change the percentile fraction
acoustat(note, fraction=50)
## change the STFT parameters
acoustat(note, wl=1024, ovlp=80)
## change the function to compute the aggregate contours
## standard deviation instead of sum
acoustat(note, aggregate=sd)
## direct time and frequency selection
acoustat(tico, tlim=c(0.5,0.9), flim=c(3,6))
## some useless graphical changes
acoustat(note, type="o", col="blue")

addsilw  

Add or insert a silence section

Description

Add or insert a silence section to a time wave.

Usage

addsilw(wave, f, at = "end", choose = FALSE, d = NULL,
plot = FALSE, output = "matrix", ...)

Arguments

wave an R object.

f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.

at position where to add the silence section (in s). Can be also specified as "start", "middle" or "end".

choose logical, if TRUE the point where silence will be added into waveR (=at) can be graphically chosen with a cursor.

d duration of the silence section to add (in s).

plot logical, if TRUE returns an oscillographic plot of wave with the new silence section (by default TRUE).

output character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".

... other oscillo graphical parameters.
afilter

Description
This function deletes all signal which amplitude is below a selected threshold.

Usage
afilter(wave, f, threshold = 5, plot = TRUE, listen = FALSE, output = "matrix", ...)

Arguments
- wave: an R object.
- f: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
- threshold: amplitude threshold (in %).
- plot: logical, if TRUE plots the new oscillogram (by default TRUE).
- listen: if TRUE the new sound is played back.
- output: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
- ...: other oscillo graphical parameters.

Details
The threshold value is in % relative to the maximal value of wave. Signal inferior to this value is clipped.

Value
If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Author(s)
Jerome Sueur <sueur@mnhn.fr>

See Also
oscillo, cutw, deletew, fadew, pastew, mutew, revw, zapsilw
Note

This function is used as an argument (threshold) in the following functions: autoc, csh, dfreq, timer and zc.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

oscillo

Examples

data(orni)
op<-par(mfrow=c(2,1))
afilter(orni,f=22050)
title(main = "threshold level = 5")
afilter(orni,f=22050,threshold=0.5,colwave="blue")
title(main = "threshold level = 0.5")
par(op)

akamatsu Water tank minimum resonant and cutoff frequencies

Description

This function computes the resonant and cutoff frequencies when recording in a given aquarium according to the criteria explained in Akamatsu et al. (2002)

Usage

akamatsu(Lx, Ly, Lz, mode = c(1,1,1),
c = 148000, plot = FALSE, xlab = "Frequency (kHz)",
ylab = "Attenuation distance (cm)", ...)

Arguments

Lx watertank length (in cm).
Ly watertank width (in cm).
Lz watertank height (in cm).
mode mode, see details.
c sound velocity in cm/s (by default 148000 cm/s in water).
plot logical, if TRUE plots the attenuation distance in function of frequency.
xlab title of the x axis if plot is TRUE.
ylab title of the y axis if plot is TRUE.
... other plot graphical parameters.
Details

From Akamatsu et al. (2002):

1. Resonant frequency

The calculated resonant frequencies of a rectangular glass tank with the dimension of $L_x$, $L_y$, and $L_z$ (in centimeters) can be described by the following equation:

$$f_{\text{rectangular}}^{lmn} = \frac{c}{2} \times \sqrt{\left(\frac{l}{L_x}\right)^2 + \left(\frac{m}{L_y}\right)^2 + \left(\frac{n}{L_z}\right)^2}$$

where $c$ is the sound velocity (cm/s) and each $l$, $m$, $n$ represents an integer, and the combination of these parameters designates the ‘mode number’. The mode (1, 1, 1) represents the resonance wave of minimum frequency. The mode (2, 1, 1) represents one of the higher order of resonant component and has additional node of the soundpressure level at the middle of the X axis, i.e., $L_x/2$.

2. Cutoff frequency

The cutoff frequency can be calculated as follows:

$$f_{\text{rectangular}}^{\text{cutoff}} = \frac{c}{2} \times \sqrt{\left(\frac{1}{L_y}\right)^2 + \left(\frac{1}{L_z}\right)^2}$$

3. Attenuation distance

The theoretical attenuation distance $D$ can be expressed in function of the cutoff frequency and the projected frequency following:

$$D_{\text{rectangular}}(f) = 2 \times \log_{10} \left( \frac{c}{4\pi f_{\text{cutoff}}} \right) \times \frac{1}{\sqrt{1 - \left( \frac{f}{f_{\text{cutoff}}} \right)^2}}$$

Value

A list of two items:

res  Resonant frequency (in Hz). See Details

cut  Cut frequency (in Hz). See Details

Author(s)

Camille Desjonqueres
References


Examples

```r
akamatsu(60, 30, 40)
```

**ama**

*Amplitude modulation analysis of a time wave*

Description

This function computes the Fourier analysis of a time wave envelope. This allows to detect periodicity, in particular those generated by amplitude modulations.

Usage

```r
ama(wave, f, envt = "hil", wl = 512, plot = TRUE, type = "l", ...)
```

Arguments

- `wave` an R object.
- `f` sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `envt` the type of envelope to be used: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope.
- `wl` length of the window for the analysis (even number of points, by default = 512).
- `plot` logical, if `plot` is `TRUE` the spectrum of the envelope (by default `TRUE`).
- `type` if `plot` is `TRUE`, type of plot that should be drawn. See `plot` for details (by default "l" for lines).
- `...` other `meanspec` parameters.

Details

This function is based on `env` and `meanspec`. The envelope of `wave` is first computed and the spectrum of this envelope is then processed. All `env` and `meanspec` arguments can be set up. Be sure to set up `wl` large enough if you want to detect low amplitude modulation periodicity.

Value

If `plot` is `FALSE`, `ama` returns a numeric vector corresponding to the computed spectrum. If `peaks` is not `NULL`, `ama` returns a list with two elements:

- `spec` the spectrum computed
- `peaks` the peaks values (in kHz).
Author(s)
Jerome Sueur <sueur@mnhn.fr>

See Also
env, fma, meanspec

Examples
data(orni)
# detection of the main amplitude modulation in a cicada song:
# one with a 0.258 kHz frequency (due to pulses in the echemes)
# one with a 2.369 kHz frequency (fundamental frequency)
ama(orni,f=22050, wl=1024)
# these amplitude modulations can be identify with a cursor:
ama(orni,f=22050, wl=1024, identify=TRUE)

AR
Acoustic Richness index

Description
This function computes the Acoustic Richness index based on M and Ht indices

Usage
AR(..., datatype = "objects", envt = "hil",
msmooth = NULL, ksmooth = NULL, ssmooth = NULL,
pattern = "[.wav][.mp3]")

Arguments
...
wave, Wave, WaveMC, audioSample objects if datatype="objects", or a path as a
character string to a directory including .wav and/or .mp3 files if datatype="files".
datatype A character string to specify if inputs are either R objects (datatype="objects",
default) or files (datatype="files").
envt the type of envelope to be returned: either "abs" for absolute amplitude enve-
lope or "hil" for Hilbert (default) amplitude envelope. See env.
msmooth mean smooth. See env.
ksmooth kernel smooth via kernel. See env.
ssmooth sum smooth. See env.
pattern an optional regular expression. Only file names which match the regular expres-
sion will be returned when datatype="files". By default .wav or .mp3 files. See dir.
Details

AR is ranked index based on the rank of the M and Ht indices obtained with the functions `m` and `th` respectively following:

\[ AR = \frac{\text{rank}(M) \times \text{rank}(H_t)}{n^2} \]

with

\[ 0 \leq AR \leq 1 \]

Value

A `data.frame` with three columns (M, Ht, AR) and n columns, with n the number of objects (respectively files) used as input.

Note

As a ranked index, the results returned by AR strongly depends with the set of objects (respectively files) used as input. Comparaison between different data sets may be spurious. Computing AR on a set of a single object does not make any sense but is allowed.

Author(s)

Jerome Sueur and Marion Depraetere

References


See Also

`m`, `th`, `env`

Examples

```r
## input as R objects
data(orni)
data(tico)
AR(orni, tico)
## give names to objects if you wish to have them as row names of the returned data.frame
AR(orni=orni, tico=tico)
## input as files stored in the working directory
## file names will be used as row names of the returned data.frame
## Not run:
require(tuneR)
AR(getwd(), datatype="files")
## End(Not run)
```
Generate sound intensity attenuation data

Description

This function generates dB data following theoretical spherical attenuation of sound.

Usage

attenuation(lref, dref = 1, dstop, n, plot = TRUE, xlab = "Distance (m)", ylab = "dB", type = "l", ...)

Arguments

lref reference intensity or pressure level (in dB).
dref reference distance corresponding to lref (in m.) (by default = 1).
dstop maximal distance of propagation (in m.).
n number of points generated between dref and dstop.
plot logical, if TRUE plots attenuation against distance of propagation (by default TRUE).
xlab title of the x axis.
ylab title of the y axis.
type if plot is TRUE, type of plot that should be drawn. See plot for details (by default "l" for lines).
... other plot graphical parameters.

Value

If plot is FALSE return a numeric vector with the data generated.

Note

Sound attenuation in a free, unbounded medium behaves in accordance with the inverse square law. attenuation generates data following this rule from a reference point where sound intensity level (SIL) or sound pressure level (SPL) is known. Such theoretical data can be compared with experimental data collected in a real environment.

Author(s)

Jerome Sueur

References

See Also

convSPL, moredB

Examples

# theoretical attenuation up to 150 m of a 100 dB/1m sound source
attenuation(lref=100, dref=1, dstop=150, n=200)

autoc

Short-term autocorrelation of a time wave

Description

This function returns the fundamental frequency of a harmonic time wave. This is achieved by computing a correlation of the signal with itself after a time delay.

Usage

autoc(wave, f, wl = 512, fmin, fmax, threshold = NULL, plot = TRUE, xlab = "Time (s)", ylab = "Frequency (kHz)", ylim = c(0, f/2000), pb = FALSE, ...)

Arguments

wave  an R object.
f  sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
w1  length of the window for the analysis (even number of points, by default = 512).
fmin  the minimum frequency to detect (in Hz). See details.
fmax  the maximum frequency to detect (in Hz). See details
threshold  amplitude threshold for signal detection (in %).
plot  logical, if TRUE plots the fundamental frequency against time (by default TRUE).
xlab  title of the x-axis.
ylab  title of the y-axis.
ylim  the range of y values.
pb  if TRUE returns a text progress bar in the console.
...  other plot graphical parameters.

Details

'fmin' and 'fmax' can help by reducing computing time but can also produce less accurate results.
Value

When plot is FALSE, autoc returns a two-column matrix, the first column corresponding to time in seconds (x-axis) and the second column corresponding to to fundamental frequency in kHz (y-axis). NA corresponds to pause sections in wave (see threshold).

Author(s)

Jerome Sueur <sueur@mnhn.fr> and Thierry Aubin <thierry.aubin@u-psud.fr>

References


See Also

ceps, acf

Examples

data(sheep)
# fundamental frequency of a sheep
res <- autoc(sheep, f=8000, threshold=5, fmin=100, fmax=700, plot=FALSE)
spectro(sheep, f=8000, ovlp=75, scale=FALSE)
points(res, pch=20)
legend(0.5, 3.6, "Fundamental frequency", pch=20, bty=0, cex=0.7)

bwfilter

Butterworth frequency filter

Description

This function is a Butterworth frequency filter that filters out a selected frequency section of of a time wave (low-pass, high-pass, low-stop, high-stop, bandpass or bandstop frequency filter).

Usage

bwfilter(wave, f, n = 1, from = NULL, to = NULL, bandpass = TRUE, listen = FALSE, output = "matrix")

Arguments

wave
an R object.
f
sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
n
Order of the filter. See details.
from
start frequency (in Hz) where to apply the filter.
to      end frequency (in Hz) where to apply the filter.
bandpass if TRUE a band-pass filter is applied between from and to, if not NULL a band-stop
          filter is applied between from and to (by default NULL).
listen   if TRUE the new sound is played back.
output   character string, the class of the object to return, either "matrix", "Wave",
          "Sample","audioSample" or "ts".

Details
The order of the filter determines the value of the roll-off value, that is the dB decrease per octave
of the transfer function. A filter of order \( n \) will have a transfer function with a roll-off value of \(-n\times6\)

Value
A new wave is returned. The class of the returned object is set with the argument output.

Note
This function mainly uses the functions filter() and filtfilt() from the package signal

Author(s)
Jerome Sueur, functions filter() and filtfilt() from the package signal.

References

See Also
ffilter,bwfilter,preemphasis,lfs,afilter

Examples
```r
require(signal)
f <- 8000
a <- noisew(f=f, d=1)
## low-pass
# 1st order filter
res <- bwfilter(a, f=f, n=1, to=1500)
# 8th order filter
res <- bwfilter(a, f=f, n=8, to=1500)
## high-pass
res <- bwfilter(a, f=f, from=2500)
## band-pass
res <- bwfilter(a, f=f, from=1000, to=2000)
## band-stop
res <- bwfilter(a, f=f, from=1000, to=2000,bandpass=FALSE)
```
ccoh

Continuous coherence function between two time waves

Description

This function returns a two-dimension coherence representation between two time waves. The function corresponds to a sliding coherence function along the two signals. This produces a 2-D density plot. An amplitude contour plot can be overlaid.

Usage

ccoh(wave1, wave2, f, wl = 512, ovlp = 0, plot = TRUE,
grid = TRUE, scale = TRUE, cont = FALSE,
collevels = seq(0, 1, 0.01), palette = reverse.heat.colors,
contlevels = seq(0, 1, 0.01), colcont = "black",
colbg="white", colgrid = "black",
colaxis = "black", collab="black",
xlab = "Time (s)", ylab = "Frequency (kHz)",
scalelab = "Coherence",
main = NULL,
scalefontlab = 1, scalecexlab =0.75, axisX = TRUE, axisY = TRUE,
flim = NULL, flimd = NULL,
...)

Arguments

wave1 a first R object
wave2 a second R object
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
wl window length for the analysis (even number of points, by default = 512).
ovlp overlap between two successive windows (in %).
plot logical, if TRUE plots the continuous coherence function (by default TRUE).
grid logical, if TRUE plots a y-axis grid (by default TRUE).
scale logical, if TRUE plots a dB colour scale on the right side of the plot (by default TRUE).
cont logical, if TRUE overplots contour lines on the plot (by default FALSE).
collevels a set of levels which are used to partition the amplitude range of the coherence (should be between 0 and 1).
palette a color palette function to be used to assign colors in the plot, see Details.
contlevels a set of levels which are used to partition the amplitude range for contour overplot (in dB).
colcont colour for cont plotting.
Coherence is a frequency domain function computed to show the degree of a relationship between two signals. The value of the coherence function ranges between zero and one, where a value of zero indicates there is no causal relationship between the signals. A value of one indicates the existence of linear frequency response between the two signals. This can be used, for instance, to compare the input and output signals of a system.

Any colour palette can be used. In particular, it is possible to use other palettes coming with `seewave`: `temp.colors`, `reverse.gray.colors.1`, `reverse.gray.colors.2`, `spectro.colors`, `reverse.terrain.colors`, `reverse.topo.colors`, `reverse.cm.colors` corresponding to the reverse of `terrain.colors`, `topo.colors`, `cm.colors`.

Use `locator` to identify points.

Details

Coherence is a frequency domain function computed to show the degree of a relationship between two signals. The value of the coherence function ranges between zero and one, where a value of zero indicates there is no causal relationship between the signals. A value of one indicates the existence of linear frequency response between the two signals. This can be used, for instance, to compare the input and output signals of a system.

Any colour palette can be used. In particular, it is possible to use other palettes coming with `seewave`: `temp.colors`, `reverse.gray.colors.1`, `reverse.gray.colors.2`, `spectro.colors`, `reverse.terrain.colors`, `reverse.topo.colors`, `reverse.cm.colors` corresponding to the reverse of `terrain.colors`, `topo.colors`, `cm.colors`.

Use `locator` to identify points.

Value

This function returns a list of three items:

- `time` a numeric vector corresponding to the time axis.
- `freq` a numeric vector corresponding to the frequency axis.
- `amp` a numeric matrix corresponding to the coherence. Each column corresponds to a coherence function of length `wl`.

Note

This function is based on `spec.pgram`, `contour` and `filled.contour`. See `spectro` for graphical changes.
Description

This function returns the cepstrum of a time wave allowing fundamental frequency detection.

Usage

```r
ceps(wave, f, phase = FALSE, wl = 512, at = NULL, from = NULL, to = NULL,
    tidentify = FALSE, fidentify = FALSE, col = "black", cex = 1, plot = TRUE,
    qlab = "Quefrcency (bottom: s, up: Hz)", alab = "Amplitude",
    qlim = NULL, alim = NULL, type = "l", ...)```

Arguments

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `phase`: if TRUE than the phase is taken into account in the computation of the cepstrum.
- `wl`: if `at` is not null, length of the window for the analysis (even number of points, by defaults = 512).
- `at`: position where to compute the cepstrum (in s).
- `from`: start position where to compute the cepstrum (in s).
- `to`: end position to compute the cepstrum (in s).
- `tidentify`: to identify time values on the plot with the help of a cursor.
- `fidentify`: to identify frequency values on the plot with the help of a cursor.
- `col`: colour of the cepstrum.
- `cex`: pitch size of the cepstrum.
- `plot`: logical, if TRUE plots the cepstrum.
The cepstrum of a time wave is the inverse Fourier transform of the logarithm of the Fourier transform. The cepstrum of a wave $s$ is then calculated as follows:

$$C(s) = \text{Re}[\text{FFT}^{-1} \log(|\text{FFT}(s)|)]$$

The independent variable of a cepstral graph is called the quefrency. The quefrency is a measure of time, though not in the sense of a signal in the time domain. A correspondence with the frequency domain is obtained by simply computing the reverse of the temporal $x$ coordinate. For instance if a peak appears at 0.005 s, this reveals a frequency peak at 200 Hz ($=1/0.005$). This explain the two scales plotted when plot is TRUE.

If at, from or to are FALSE then ceps computes the cepstrum of the whole signal.

When using tidentify or tidentify, press ‘stop’ tools bar button to return values in the console.

Value

When plot is FALSE, ceps returns the cesptral profile as a two-column matrix, the first column corresponding to quefrency ($x$-axis) and the second corresponding to amplitude ($y$-axis).

Warning

The argument peaks is no more available (version > 1.5.6). See the function fpeaks for peak(s) detection.

Note

Cepstral analysis is mainly used in speech processing. This analysis allows to extract the fundamental frequency, see the examples.

This function is based on fft.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

References

cepstro

See Also

cepstro, fund, autoc

Examples

data(sheep)
par(mfrow=c(2,1))
# phase not taken into account
ceps(sheep,f=8000,at=0.4,wl=1024)
# phase taken into account
ceps(sheep,f=8000,at=0.4,wl=1024, phase=TRUE)

cepstro 2D-cepstrogram of a time wave

Description

This function returns a two-dimension cepstrographic representation of a time wave. The function corresponds to a short-term cepstral transform. An amplitude contour plot can be overlaid.

Usage

cepstro(wave, f, wl = 512, ovlp = 0, plot = TRUE, grid = TRUE,
        scale = TRUE, cont = FALSE, collevels = seq(0, 1, 0.01),
        palette = reverse.heat.colors, contlevels = seq(0, 1, 0.01),
        colcont = "black", colbg = "white", colgrid = "black",
        colaxis = "black", collab = "black",
        xlab = "Time (s)", ylab = "Frequency (ms)",
        scalelab = "Amplitude", main = NULL, scalefontlab = 1, scalecexlab = 0.75,
        axisX = TRUE, axisY = TRUE, tlim = NULL, qlim = NULL, ...)

Arguments

wave  an R object.
f  sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
w1  if at is not null, length of the window for the analysis (even number of points, by defaults = 512).
ovlp  overlap between two successive windows (in %).
plot  logical, if TRUE plots the cepstrogram (by default TRUE).
grid  logical, if TRUE plots a y-axis grid (by default TRUE).
scale  logical, if TRUE plots a dB colour scale on the right side of the cepstrogram (by default TRUE).
cont  logical, if TRUE overplots contour lines on the cepstrogram (by default FALSE).
collevels  a set of levels which are used to partition the amplitude range of the cepstrogram (in dB).
palette    a color palette function to be used to assign colors in the plot.
contlevels a set of levels which are used to partition the amplitude range for contour over-plot (in dB).
colcont    colour for cont plotting.
colbg      background colour.
colgrid    colour for grid plotting.
colaxis    color of the axes.
collab     color of the labels.
xlab       label of the time axis.
ylab       label of the quefrency axis.
main       label of the main title.
scalelab   amplitude scale label.
scalefontlab font of the amplitude scale label.
scalecexlab cex of the amplitude scale label.
axisX      if TRUE plots time X-axis (by default TRUE).
axisY      if TRUE plots frequency Y-axis (by default TRUE).
tlim       modifications of the time X-axis limits.
qlim       modifications of the quefrency Y-axis limits (in ms).
...        other contour graphical parameters.

Details

It is unfortunately not possible to turn the y-axis to a frequency scale. See spectro for the use of the graphical arguments.

Value

This function returns a list of three items:
time       a numeric vector corresponding to the time axis.
freq       a numeric vector corresponding to the quefrency axis.
amp        a numeric matrix corresponding to the the successive cepstral profiles computed along time.

Note

This function is based on ceps.

Author(s)

Jerome Sueur <sueur@mnhn.fr>.
References


See Also

ceps, fund, autoc

Examples

data(sheep)
sheepc <- cutw(sheep, f=8000, from = 0.19, to = 2.3)
cepstro(sheepc,f=8000)

data(sheep)

coh          Coherence between two time waves

Description

This function returns the frequency coherence between two time waves.

Usage

coh(wave1, wave2, f, plot =TRUE, xlab = "Frequency (kHz)",
ylab = "Coherence", xlim = c(0,f/2000), type = "l",...)

Arguments

wave1        a first R object.
wave2        a second R object.
f            sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
plot         logical, if TRUE plots the continuous coherence function (by default TRUE).
xlab         title of the frequency X-axis.
ylab         title of the coherence Y-axis.
xlim         range of frequency X-axis.
type         if plot is TRUE, type of plot that should be drawn. See plot for details (by default "l" for lines).
...           other plot graphical parameters.

Details

Coherence is a frequency domain function computed to show the degree of a relationship between two signals. The value of the coherence function ranges between zero and one, where a value of zero indicates there is no causal relationship between the signals. A value of one indicates the existence of linear frequency response between the two signals. This can be used, for instance, to compare the input and output signals of a system.
Value

When plot is FALSE, this coh returns a two-column matrix, the first column being the frequency axis in kHz (x-axis) and the second column being the coherence (y-axis).

Note

This function is based on spec.pgram.

Author(s)

Jerome Sueur <sueur@mnhn.fr> but this function is based on spec.pgram by Martyn Plummer, Adrian Trapletti and B.D. Ripley.

See Also

ccoh, spectro, spec.pgram.

Examples

```r
wave1<-synth(d=1,f=4000,cf=500)
wave2<-synth(d=1,f=4000,cf=800)
coh(wave1,wave2,f=4000)
```

`combfilter`  

**Comb filter**

Description

This function processes a feedforward comb filter and plots a spectrogram of the filtered wave associated with the frequency response of the filter.

Usage

```r
combfilter(wave, f, alpha, K, units = c("samples", "seconds"), plot = FALSE, output = "matrix", ...)
```

Arguments

- `wave`: an R object
- `f`: sampling frequency (in Hz). Does not need to be specified if embedded in wave
- `alpha`: a numeric vector of length 1 for the scaling factor. See Details.
- `K`: a numeric vector of length 1 for the delay length, in units. See Details.
- `units`: units in which K is given, the default is 'samples' but can be set to 'seconds'.
- `plot`: a logical, if TRUE plots the spectrogram of the filtered wave and the frequency response of the comb filter.
combfilter

output character string, the class of the object to return, either 'matrix', 'Wave', 'Sample', 'audioSample' or 'ts'.

... other arguments to be passed to spectro except scale and osc that are set by default to FALSE.

Details

A comb filter consists in adding a delayed version of a signal to itself resulting in constructive and destructive interference. The feedforward version of a comb filter can be written following:

\[ y(n) = x(n) + \alpha \times x(n - K) \]

where alpha is the scaling factor and K the delay length. The frequency response of the filter is obtained with:

\[ H(f) = \sqrt{(1 + \alpha^2) + 2 \times \cos(\omega K)} \]

The frequency response is periodic. The depth of the cycles is controlled with alpha and the number of cycles with K.

Value

A new wave is returned. The class of the returned object is set with the argument output.

Note

Setting K to high values may generate unwanted results. The feedback form of the combfilter is not implemented yet.

Author(s)

Jerome Sueur

See Also

combfilter, fir, squarefilter, drawfilter, ffilter, bwfilter

Examples

```r
## Not run:
f <- 44100
## chirp
s1 <- synth(f=f, cf=1, d=2, fm=c(0,0,f/2,0,0), out="Wave")
combfilter(s1, alpha=1, K=50, plot=TRUE)
## harmonic sound
s2 <- synth(f=f, d=2, cf=600, harmonics=rep(1, 35), output="Wave")
combfilter(s2, alpha=1, K=10, plot=TRUE)
## noise, units in seconds
s3 <- noisew(d=2, f=44100, out="Wave")
combfilter(s3, alpha=0.5, K=1e-4, units="seconds", plot=TRUE)
## End(Not run)
```
convSPL  

Convert sound pressure level in other units

Description

This function converts sound pressure level (in dB) in sound power (Watt), intensity (Watt/m2) and pressure (Pa). By default, these conversions are applied to air-borne sound.

Usage

```r
convSPL(x, d = 1, Iref = 10^-12, pref = 2*10^-5)
```

Arguments

- `x`: a numeric vector or a matrix describing SPL values (in dB).
- `d`: the distance from the sound source where SPL values have been measured (in meter) (by default = 1m)
- `Iref`: reference intensity (in Watt/m2) (by default = 10^-12)
- `pref`: reference pressure (in Pa) (by default = 2*10^-5)

Value

convSPL returns a list containing three components:

- `P`: data converted in sound power (in Watt).
- `I`: data converted in sound intensity (in Watt/m2).
- `p`: data converted in sound pressure (in Pa).

Note

`Iref` and `pref` correspond to a 1 kHz sound in air.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

References


See Also

- `moredB`, `dBweight`, `attenuation`

Examples

```r
# conversion of two SPL measurements taken at 0.5 m from the source
convSPL(c(80, 85), d=0.5)
```
Cross-correlation between two time wave envelopes

Description
This function tests the similarity between two time wave envelopes by returning their maximal correlation and the time shift related to it.

Usage
corenv(wave1, wave2, f, envt="hil", msmooth = NULL, ksmooth = NULL, ssmooth = NULL, plot = TRUE, plotval = TRUE, method = "spearman", col = "black", colval = "red", cexval = 1, fontval = 1, xlab = "Time (s)", ylab = "Coefficient of correlation (r)", type = "l", pb = FALSE, ...)

Arguments
- wave1: a first R object.
- wave2: a second R object.
- f: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
- envt: the type of envelope to be used: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope. See env.
- msmooth: a vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %). See env.
- ksmooth: kernel smooth via kernel. See env.
- ssmooth: sum smooth. See env.
- plot: logical, if TRUE plots r values against frequency shift (by default TRUE).
- plotval: logical, if TRUE adds to the plot maximum r value and frequency offset (by default TRUE).
- method: a character string indicating which correlation coefficient is to be computed ("pearson", "spearman", or "kendall") (see cor).
- col: colour of r values.
- colval: colour of r max and frequency offset values.
- cexval: character size of r max and frequency offset values.
- fontval: font of r max and frequency offset values.
- xlab: title of the frequency axis.
- ylab: title of the r axis.
- type: if plot is TRUE, type of plot that should be drawn. See plot for details (by default "l" for lines).
- pb: if TRUE returns a text progress bar in the console.
- ... other plot graphical parameters.
Details

Successive correlations between the envelopes of `wave1` and `wave2` are computed when regularly sliding forward and backward `wave2` along `wave1`. The maximal correlation is obtained at a particular shift (time offset). This shift may be positive or negative. The higher `smooth` is set up, the faster will be the computation but less precise the results will be. The corresponding `p` value, obtained with `cor.test`, is plotted. Inverting `wave1` and `wave2` may give slight different results.

Value

If `plot` is `FALSE`, `corenv` returns a list containing four components:

- `r` a two-column matrix, the first column corresponding to the time shift (frequency x-axis) and the second column corresponding to the successive r correlation values between `env1` and `env2` (correlation y-axis).
- `rmax` the maximum correlation value between x and y.
- `p` the `p` value corresponding to `rmax`.
- `t` the time offset corresponding to `rmax`.

Author(s)

Jerome Sueur

See Also

`env`, `corspec`, `cvspectro`, `cor.cor.test`.

Examples

```r
## Not run:
data(orni)
# cross-correlation between two schemes of a cicada song
wave1<-cutw(orni,f=22050,from=0.3,to=0.4,plot=FALSE)
wave2<-cutw(orni,f=22050,from=0.58,to=0.68,plot=FALSE)
corenv(wave1,wave2,f=22050)
## End(Not run)
```

corspec

Cross-correlation between two frequency spectra

Description

This function tests the similarity between two frequency spectra by returning their maximal correlation and the frequency shift related to it.
Usage

corspec(spec1, spec2, f = NULL, mel = FALSE, plot = TRUE, plotval = TRUE,
method = "spearman", col = "black", colval = "red",
cexval = 1, fontval = 1, xlab = NULL,
ylab = "Coefficient of correlation (r)", type="l",...)

Arguments

spec1 a first data set resulting of a spectral analysis obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

spec2 a first data set resulting of a spectral analysis obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

f sampling frequency of waves used to obtain spec1 and spec2 (in Hz). Not necessary if spec1 and/or spec2 is a two columns matrix obtained with spec or meanspec.

mel a logical, if TRUE the (htk-)mel scale is used.

plot logical, if TRUE plots r values against frequency shift (by default TRUE).

plotval logical, if TRUE adds to the plot maximum r value and frequency offset (by default TRUE).

method a character string indicating which correlation coefficient is to be computed ("pearson", "spearman", or "kendall") (see cor).

col colour of r values.

colval colour of r max and frequency offset values.

cexval character size of r max and frequency offset values.

fontval font of r max and frequency offset values.

xlab title of the frequency axis.

ylab title of the r axis.

type if plot is TRUE, type of plot that should be drawn. See plot for details (by default "l" for lines).

... other plot graphical parameters.

Details

It is important not to have data in dB.
Successive correlations between spec1 and spec2 are computed when regularly shifting spec2 towards lower or higher frequencies.
The maximal correlation is obtained at a particular shift (frequency offset). This shift may be positive or negative.
The corresponding p value, obtained with cor.test, is plotted.
Inverting spec1 and spec2 may give slight different results, see examples.
Value

If plot is FALSE, corspec returns a list containing four components:

- \( r \) a two-column matrix, the first column corresponding to the frequency shift (frequency x-axis) and the second column corresponding to the successive \( r \) correlation values between \( \text{spec1} \) and \( \text{spec2} \) (correlation y-axis).
- \( r_{\text{max}} \) the maximum correlation value between \( \text{spec1} \) and \( \text{spec2} \).
- \( p \) the \( p \) value corresponding to \( r_{\text{max}} \).
- \( f \) the frequency offset corresponding to \( r_{\text{max}} \).

Author(s)

Jerome Sueur, improved by Laurent Lellouch

References


See Also

\( \text{spec}, \text{meanspec}, \text{corspec}, \text{covspectro}, \text{cor}, \text{cor.test} \).

Examples

## Not run: data(tico)
## compare the two first notes spectra
a<-\text{spec(tico, f=22050, wl=512, at=0.2, plot=FALSE)}
c<-\text{spec(tico, f=22050, wl=512, at=1.1, plot=FALSE)}
\text{op<par(mfrow=c(2,1), mar=c(4.5,4,3,1))}
\text{spec(tico, f=22050, at=0.2, col="blue")}
\text{par(new=TRUE)}
\text{spec(tico, f=22050, at=1.1, col="green")}
\text{legend(x=8, y=0.5, c("Note A", "Note C"), lty=1, col=c("blue", "green"), bty="o")}
\text{par(mar=c(5,4,2,1))}
\text{corspec(a, c, ylim=c(-0.25,0.8), xaxs="i", yaxs="i", las=1)}
\text{par(op)}
## different correlation methods give different results...
\text{op<par(mfrow=c(3,1))}
\text{corspec(a, c, xaxs="i", las=1, ylim=c(-0.25,0.8))}
\text{title("spearmann correlation (by default)")}
\text{corspec(a, c, xaxs="i", las=1, ylim=c(0,1), method="pearson")}
\text{title("pearson correlation")}
\text{corspec(a, c, xaxs="i", las=1, ylim=c(-0.23,0.5), method="kendall")}
\text{title("kendall correlation")}
\text{par(op)}
## inverting x and y does not give exactly similar results
\text{op<par(mfrow=c(2,1), mar=c(2,4,3,1))}
\text{corspec(a, c)}
\text{corspec(c, a)}
covspectro

Covariance between two spectrograms

Description

This function tests the similarity between two spectrograms by returning their maximal covariance and the time shift related to it.

Usage

covspectro(wave1, wave2, f, wl = 512, wn = "hanning", n,
plot = TRUE, plotval = TRUE,
method = "spearman", col = "black", colval = "red", cexval = 1,
fontval = 1, xlab = "Time (s)",
 ylab = "Normalised covariance (cov)", type = "l", pb = FALSE, ...)

Arguments

wave1 a first R object.
wave2 a second R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
wl length of the window for the analysis (even number of points, by default = 512).
wn window name, see ftwindow (by default "hanning").
n number of covariances computed between wave1 and wave2 when sliding wave2 along wave1.
plot logical, if TRUE plots r values against frequency shift (by default TRUE).
plotval logical, if TRUE adds to the plot maximum R value and frequency offset (by default TRUE).
method a character string indicating which correlation coefficient is to be computed ("pearson", "spearman", or "kendall") (see cor).
col colour of r values.
colval colour of r max and frequency offset values.
covspectro

cexval character size of r max and frequency offset values.
fontval font of r max and frequency offset values.
xlab title of the frequency axis.
ylab title of the r axis.
type if plot is TRUE, type of plot that should be drawn. See plot for details (by default "l" for lines).
pb if TRUE returns a text progress bar in the console.
... other plot graphical parameters.

Details
Successive covariances between the spectrogram of waveQ and the spectrogram of waveR are computed when regularly sliding forward and backward waveR along waveQ. The maximal covariance is obtained at a particular shift (time offset). This shift may be positive or negative.
n sets in how many steps waveR will be slided along waveQ. Time process can be then decreased by setting low n value. Inverting waveQ and waveR may give slight different results.

Value
If plot is FALSE, covspectro returns a list containing three components:
cov the successive covariance values between waveQ and waveR.
covmax the maximum covariance between waveQ and waveR.
t the time offset corresponding to cov.

Author(s)
Jerome Sueur <sueur@mnhn.fr>

References

See Also
corspec, corenv, spectro, cor.

Examples
# covariance between two notes of a birdsong
## Not run:
data(tico)
note1<-cutw(tico, f=22050, from=0.5, to=0.9)
note2<-cutw(tico, f=22050, from=0.9, to=1.3)
covspectro(note1,note2,f=22050,n=37)
## End(Not run)
crest

Crest factor and visualization

Description

This function returns the crest factor and localizes the different crest(s).

Usage

crest(wave, f, plot = FALSE, col = 2, cex = 3, symbol = "*", ...)

Arguments

wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
plot if TRUE plots the oscillogram of wave and indicates the location of the crest(s)
col color of the symbol indicating the localisation of the crest(s)
cex symbol magnification
symbol symbol indicating the localisation of the crest(s)
... other

Details

The crest factor of a time series s is calculated according to:

\[ C = \frac{\text{max}(s)}{\text{rms}(s)} \]

with rms the root-mean-square (see \texttt{rms}).

Value

The function returns a list of three items

C crest factor
val value of the crest(s)
loc location of the crest(s)

Note

There might be several crests (maxima) along the time wave but there is a single crest factor.

Author(s)

Jerome Sueur <sueur@mnhn.fr>
References


See Also

oscillo, rms

Examples

data(tico)
crest(tico, f=22050)
  # see the crest location and change the default graphical parameters
crest(tico, f=22050, plot=TRUE, sym="-")

csh  
Continuous spectral entropy

Description

This function computes the continuous spectral entropy (H) of a time wave.

Usage

csh(wave, f, wl = 512, wn = "hanning", ovlp = 0, fftw = FALSE, threshold = NULL,
    plot = TRUE, xlab = "Times (s)", ylab = "Spectral Entropy",
    ylim = c(0, 1.1), type = "l", ...)

Arguments

wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded
    in wave.
wl if at is not null, length of the window for the analysis (even number of points,
    by default = 512).
wn window name, see ftwindow (by default "hanning").
ovlp overlap between two successive windows (in %).
fftw if TRUE calls the function FFT of the library fftw. See Notes of the spectro.
threshold amplitude threshold for signal detection (in %).
plot logical, if TRUE plots the spectral entropy against time (by default TRUE).
xlab title of the x axis.
ylab title of the y axis.
ylim the range of y values.
type if plot is TRUE, type of plot that should be drawn. See plot for details (by
    default "l" for lines).
... other plot graphical parameters.
Details

See `sh` for computing method.

Value

When `plot` is `FALSE`, `csh` returns a two-column matrix, the first column being time in seconds (x-axis) and the second column being the spectral entropy (y-axis) computed along time. NA corresponds to pause sections in `wave` (see `threshold`).

Note

The spectral entropy of a noisy signal will tend towards 1 whereas the spectral entropy of a pure tone signal will tend towards 0.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

References


See Also

`sh`, `th`

Examples

data(orni)
csh(orni,f=22050,wl=512,ovlp=50)
# using the threshold argument can lead to some edge effects
# here `sh=1` at the end of schemes
csh(orni,f=22050,wl=512,ovlp=50,threshold=5)

---

cutspec  
*Cut a frequency spectrum*

Description

This function can be used to select (cut) a specific part of a frequency spectrum.

Usage

cutspec(spec, f = NULL, flim, mel = FALSE, norm = FALSE, PMF = FALSE)
Arguments

spec a vector or a two-column matrix set resulting of a spectral analysis. This can be the value obtained with `spec` or `meanspec`.

f sampling frequency of `spec` (in Hz).

flim a vector of length 2 to specify the new frequency range (in kHz).

mel a logical, if TRUE the (htk-)mel scale is used.

norm a logical, if TRUE the spectrum returned is normalised between 0 and 1.

PMF a logical, if TRUE the spectrum returned is a probability mass function.

Value

A new spectrum is returned. The class of the returned object is the one of the input object (`spec`).

Note

The sampling frequency `f` is not necessary if `spec` has been obtained with either `spec` or `meanspec`. This function can be used before calling analysis function like `sh` or `sfm`. See examples.

Author(s)

Jerome Sueur, improved by Laurent Lellouch

See Also

`spec`, `meanspec`

Examples

data(orni)
a <- meanspec(orni, f=22050, plot=FALSE)
b <- cutspec(a, flim=c(4,8))
## quick check with a plot
plot(b, type="l")
## effects on spectral entropy
sfm(a)
sfm(b)
## mel scale
require(tuneR)
mel <- melfcc(orni, nbands = 256, dcttype = "t3", fbtype = "htkmel", spec_out=TRUE)
melspec.mean <- apply(mel$aspectrum, MARGIN=2, FUN=mean)
c <- cutspec(melspec.mean, f=22050, flim=c(4000,8000), mel=TRUE)
cutw

Cut a section of a time wave

Description
This function selects and cuts a section of data describing a time wave. Original and cut sections can be plotted as oscillograms for comparison.

Usage
cutw(wave, f, from = NULL, to = NULL, choose = FALSE, plot = FALSE, marks = TRUE, output="matrix", ...)

Arguments
- wave: an R object.
- f: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
- from: start mark (in s).
- to: end mark (in s).
- choose: logical, if TRUE start (=from) and end (=to) points can be graphically chosen with a cursor on the oscillogram.
- plot: logical, if TRUE returns an oscillographic plot of original and cut sections (by default FALSE).
- marks: logical, if TRUE shows the start and end mark on the plot (by default TRUE).
- output: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
- ...: other oscillo graphical parameters.

Details
If plot is TRUE returns a two-frame plot with both original and cut sections.

Value
If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Author(s)
Jerome Sueur

See Also
oscillo, addsilw, deletew, fadew, mutew, pastew, revw, zapsilw
Examples

# a 0.4 s section in a bird song
data(tico)
a<-cut(tico,f=22050,from=0.5,to=0.9)
oscillo(a,22050)
# a direct way to see what has been cut
cut(tico,f=22050,from=0.5,to=0.9,plot=TRUE)

dBscale dB colour scale for a spectrogram display

Description

This function displays a vertical or horizontal dB colour scale to be used with spectro plots.

Usage

dBscale(collevels, palette = spectro.colors, side = 4,
textlab = "Amplitude\n(dB)", cexlab = 0.75,
fontlab = 1, collab = "black", colaxis = "black",...)

Arguments

collevels a set of levels which are used to partition the amplitude range of the spectrogram (in dB).
palette a color palette function to be used to assign colors in the plot, see note.
side side of the axis.
textlab text of the label.
cexlab character size of the label.
fontlab font of the label.
collab colour of the label.
colaxis colour of the axis.
... other axis arguments.

Note

This function, based on filled.contour by Ross Ihaka, is not supposed to be used by itself but as a legend of spectro. Any colour palette can be used. In particular, it is possible to use other palettes coming with seewave: rev.gray.colors.1, rev.gray.colors.2, rev.heat.colors, rev.terrain.colors, rev.topo.colors, rev.cm.colors corresponding to the reverse of heat.colors, terrain.colors, topo.colors, cm.colors.

Author(s)

Jerome Sueur <sueur@mnhn.fr> and Caroline Simonis <csimonis@mnhn.fr>.
**dBweight**

See Also

* spectro.*

Examples

```r
data(pellucens)
# place the scale on the left and not on the right as spectro() does
def.par <- par(no.readonly = TRUE)
layout(matrix(c(1, 2), nc = 2), widths = c(1, 5))
par(mar=c(5,3,4,2))
dbscale(collevels=seq(-30,0,1),side=2)
par(mar=c(5,4,4,2))
spectro(pellucens, f=22050, wl=512, scale=FALSE)
par(def.par)
# place the scale on the top and not on the right as spectro() does
def.par <- par(no.readonly = TRUE)
layout(matrix(c(0,1,2,2), nc = 2, byrow=TRUE),widths=c(1,2),heights=(c(1,5.5)))
par(mar=c(0,5,3,4,2))
dbscale(collevels=seq(-30,0,1),textlab="",side=3)
mtext("Amplitude (dB)",side=2,line = 1,at=0.6,cex=0.75)
par(mar=c(5,4,0.5,2))
spectro(pellucens, f=22050, wl=512, scale=FALSE)
par(def.par)
```

---

<table>
<thead>
<tr>
<th>dBweight</th>
<th>dB weightings</th>
</tr>
</thead>
</table>

**Description**

This function returns the four most common dB weightings.

**Usage**

```r
dbWeight(f, dBRef = NULL)
```

**Arguments**

- `f` frequency (in Hz).
- `dBRef` dB reference level (by default NULL).

**Details**

By default, the function returns four weightings. When `dBRef` is not NULL then the function returns the conversion from a dB reference level to four dB weighting levels.
Value

dBweight returns a list of four items corresponding to four dB weightings.

A    dB (A)
B    dB (B)
C    dB (C)
D    dB (D)

Note

The transfer equations used here come from Wikipedia but they were originally coming from the appendix of an international standard on the design performance of sound level meters IEC 651:1979 (Neil Glenister, pers. com.).

Author(s)

Jerome Sueur <sueur@mnhn.fr> and Zev Ross

References


See Also

convSPL, moredB

Examples

# weight for a 50 Hz frequency
dBweight(f=50)
# A weight for the 1/3 Octave centre frequencies.
dBweight(f=c(20,25,31.5,40,50,63,80,100,125,160,200,250,
            315,400,500,630,800,1000,1500,
            2000,2500,3150,4000,5000,
            6300,8000,10000,12500,16000,20000))
# correction for a 50 Hz sound emitted at 100 dB
dBweight(f=50, dB=100)
# weighting curves plot
f <- seq(10,20000,by=10)
par(las=1)
plot(f, dBweight(f)$A, type="n", log="x",
xlim=c(10,10^5), ylim=c(-80,20), xlab="",ylab="",xaxt="n",yaxt="n")
abline(v=c(seq(10,100,by=10), seq(100,1000,by=100),
            seq(1000,10000,by=1000), seq(10000,100000,by=10000),
            c(100,1000,10000,100000)),col="lightgrey",lty=2)
abline(v=c(100,1000,10000,100000),col="grey")
abline(h=seq(-80, 20,20),col="grey")
par(new=TRUE)
plot(f, dBweight(f)$A, type="l", log="x",
xlab="Frequency (Hz)", ylab="dB", lwd=2, col="blue", xlim=c(10,10^5), ylim=c(-80,20))
**deletew**

`deletew(main="Acoustic weighting curves (10 Hz - 20 kHz)")`

```r
lines(x=f, y=dbweight(f)$B, col="green", lwd=2)
lines(x=f, y=dbweight(f)$C, col="red", lwd=2)
lines(x=f, y=dbweight(f)$D, col="black", lwd=2)
legend("bottomright", legend=c("dB(A)","dB(B)","dB(C)","dB(D)"),
lwd=2, col=c("blue", "green", "red", "black"), bty="o", bg="white")
```
Author(s)
Jerome Sueur <sueur@mnhn.fr>

See Also
oscillo, addsilw, cutw, fadew, mutew, pastew, revw, zapsilw

Examples

```
# deletion a 0.4 s section in a bird song
data(tico)
a<-deletew(tico,f=22050,from=0.5, to=0.9)
oscillo(a,22050)
# a direct way to see what has been cut
deletew(tico,f=22050,from=0.5, to=0.9,plot=TRUE)
```

### dfreq

**Dominant frequency of a time wave**

**Description**

This function gives the dominant frequency (i.e. the frequency of highest amplitude) of a time wave.

**Usage**

```
dfreq(wave, f, wl = 512, wn = "hanning", ovlp = 0, fftw= FALSE, at = NULL, tlim = NULL, threshold = NULL, bandpass = NULL, clip = NULL, plot = TRUE, xlab = "Times (s)" , ylab = "Frequency (kHz)" , ylim = c(0, f/2000), ...)
```

**Arguments**

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl`: length of the window for the analysis (even number of points, by default = 512).
- `wn`: window name, see `ftwindow` (by default "hanning").
- `ovlp`: overlap between two successive analysis windows (in %).
- `fftw`: if TRUE calls the function FFT of the library `fftw`. See Notes of the `spectro`.
- `at`: time position where the dominant frequency has to be computed (in s.).
- `tlim`: modifications of the time X-axis limits.
- `threshold`: amplitude threshold for signal detection (in %).
- `bandpass`: a numeric vector of length two, giving the lower and upper limits of a frequency bandpass filter (in Hz).
dfreq

clip a numeric value to select dominant frequency values according to their amplitude in reference to a maximal value of 1 for the whole signal (has to be >0 & < 1).

plot logical, if TRUE plots the dominant frequency against time (by default TRUE).

xlab title of the x axis.

ylab title of the y axis.

ylim the range of y values.

... other plot graphical parameters.

Value

When plot is FALSE, dfreq returns a two-column matrix, the first column corresponding to time in seconds (x-axis) and the second column corresponding to dominant frequency in kHz (y-axis). NA corresponds to pause sections in wave (see threshold).

Note

This function is based on fft.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

spec, meanspec, spectro.

Examples

data(tico)
f <- 22050
# default
dfreq(tico, f)
# using the amplitude threshold and changing the graphical output
dfreq(tico, f, ovlp=50, threshold=5, type="1", col=2)
# using 'at' argument for specific positions along the time axis
dfreq(tico, f, at=c(0.25, 0.75, 1.2, 1.6))
dfreq(tico, f, at=seq(0.5, 1.4, by=0.005), threshold=5)
# a specific number of measures on a single note
dfreq(tico, f, at=seq(0.5, 0.9, len=100), threshold=5, xlim=c(0.5,0.9))
# overlap on spectrogram
# and use of 'clip' argument to better track the dominant frequency
# in noisy conditions
op <- par()
ticon <- tico@left/max(tico@left) + noises(d=length(tico@left))/f, f)
spectro(ticon, f)
res <- dfreq(ticon, f, clip=0.3, plot=FALSE)
points(res, col=2, pch =13)
par(op)
**diffcumspec**  

*Difference between two cumulative frequency spectra*

**Description**

This function compares two distributions (e.g. two frequency spectra) by computing the difference between two cumulative frequency spectra.

**Usage**

```r
diffcumspec(spec1, spec2, f = NULL, mel = FALSE,  
plot = FALSE, type = "l", lty = c(1, 2), col = c(2, 4, 8),  
flab = NULL, alab = "Cumulated amplitude",  
flim = NULL, alim = NULL,  
title = TRUE, legend = TRUE, ...)  
```

**Arguments**

- `spec1` any distribution, especially a spectrum obtained with `spec` or `meanspec` (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).
- `spec2` any distribution, especially a spectrum obtained with `spec` or `meanspec` (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).
- `f` sampling frequency of waves used to obtain `spec1` and `spec2` (in Hz). Not necessary if `spec1` and/or `spec2` is a two columns matrix obtained with `spec` or `meanspec`.
- `mel` a logical, if TRUE the (htk-)mel scale is used.
- `plot` logical, if TRUE plots both cumulative spectra and their distance.
- `type` if `plot` is TRUE, type of plot that should be drawn. See `plot` for details (by default "l" for lines).
- `col` a vector of length 3 for the colour of `spec1`, `spec2`, and the difference between each of them.
- `lty` a vector of length 2 for the line type of `spec1` and `spec2` if `type="l"`.
- `flab` title of the frequency axis.
- `alab` title of the amplitude axis.
- `flim` the range of frequency values.
- `alim` range of amplitude axis.
- `title` logical, if TRUE, adds a title with D and F values.
- `legend` logical, if TRUE adds a legend to the plot.
- `...` other `plot` graphical parameters.
**diffcumspec**

**Details**

Both spectra are transformed into cumulative distribution functions (CDF). Spectral difference is then computed according to:

\[ D_{df}(x, y) = \frac{\sum_{i=1}^{n} |X_i - Y_i|}{n}, \text{with} X \text{ and } Y \text{ the spectrum CDFs, and } D \in [0, 1]. \]

**Value**

A numeric vector of length 1 returning the difference between the two spectra. No unit.

**Note**

This metric is sensitive not only to the spectral overlap between but also to the mean frequential distance between the different frequency peaks.

**Author(s)**

Laurent Lellouch, Jerome Sueur

**References**


**See Also**

kl.dist, ks.dist, simspec, diffspec, logspec.dist, itakura.dist

**Examples**

```r
## Hz scale
data(tico)
data(orni)
orni.hz <- meanspec(orni, plot=FALSE)
tico.hz <- meanspec(tico, plot=FALSE)
diffcumspec(orni.hz, tico.hz, plot=TRUE)
## mel scale
require(tuner)
orni.mel <- melfcc(orni, nbands = 256, dcttype = "t3", fbtype = "htkmel", spec_out=TRUE)
orni.mel.mean <- apply(orni.mel$spectrum, MARGIN=2, FUN=mean)
tico.mel <- melfcc(tico, nbands = 256, dcttype = "t3", fbtype = "htkmel", spec_out=TRUE)
tico.mel.mean <- apply(tico.mel$spectrum, MARGIN=2, FUN=mean)
diffcumspec(orni.mel.mean, tico.mel.mean, f=22050, mel=TRUE, plot=TRUE)
```
**diffenv**

**Difference between two amplitude envelopes**

**Description**

This function estimates the surface difference between two amplitude envelopes.

**Usage**

```r
diffenv(wave1, wave2, f, envt = "hil", msmooth = NULL, ksmooth = NULL, plot = FALSE, lty1 = 1, lty2 = 2, col1 = 2, col2 = 4, cold = 8, xlab = "Time (s)", ylab = "Amplitude", ylim = NULL, legend = TRUE, ...)```

**Arguments**

- `wave1`: a first R object.
- `wave2`: a second R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `envt`: the type of envelope to be used: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope. See `env`.
- `msmooth`: a vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %). See `env`.
- `ksmooth`: kernel smooth via `kernel`. See `env`.
- `plot`: logical, if TRUE plots both envelopes and their surface difference (by default FALSE).
- `lty1`: line type of the first envelope (envelope of `wave1`).
- `lty2`: line type of the second envelope (envelope of `wave2`).
- `col1`: colour of the first envelope (envelope of `wave1`).
- `col2`: colour of the second envelope (envelope of `wave2`).
- `cold`: colour of the surface difference.
- `xlab`: title of the time axis.
- `ylab`: title of the amplitude axis.
- `ylim`: range of amplitude axis.
- `legend`: logical, if TRUE adds a legend to the plot.
- `...`: other `plot` graphical parameters.

**Details**

Envelopes of both waves are first transformed as probability mass functions (PMF). Envelope difference is then computed according to:

$$D = \sum \frac{|env1 - env2|}{2}, \text{with } D \in [0, 1].$$
**Value**

The difference is returned. This value is without unit. When `plot` is `TRUE`, both envelopes and their difference surface are plotted on the same graph.

**Note**

This method can be used as a relative distance estimation between different envelopes.

**Author(s)**

Jerome Sueur <sueur@mnhn.fr>.

**References**


**See Also**

`env`, `corenv`, `diffspec`, `diffwave`

**Examples**

```r
data(tico); tico <- tico@left
data(orni); orni <- orni@left
# selection in tico of two waves with similar duration
tico2<-tico[1:length(orni)]
diffenv(tico2,orni,f=22050,plot=TRUE)
# smoothing the envelope gives a better graph but slightly changes the result
diffenv(tico2,orni,f=22050,msmooth=c(20,0),plot=TRUE)
```

### diffspec

**Difference between two frequency spectra**

**Description**

This function estimates the surface difference between two frequency spectra.

**Usage**

```r
diffspec(spec1, spec2, f = NULL, mel = FALSE, plot = FALSE, type="l",
lty=c(1, 2), col =c(2, 4, 8),
flab = NULL, alab = "Amplitude",
flim = NULL, alim = NULL, title = TRUE, legend = TRUE, ...)
```
Arguments

spec1  a first data set resulting of a spectral analysis obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

spec2  a first data set resulting of a spectral analysis obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

f  sampling frequency of waves used to obtain spec1 and spec2 (in Hz). Not necessary if spec1 and/or spec2 is a two-column matrix obtained with spec or meanspec.

mel  a logical, if TRUE the (htk-)mel scale is used.

plot  logical, if TRUE plots both spectra and their surface difference (by default FALSE).

type  if plot is TRUE, type of plot that should be drawn. See plot for details (by default "l" for lines).

lty  a vector of length 2 for the line type of spec1 and spec2 if type="l".

col  a vector of length 3 for the colour of spec1, spec2, and the surface difference between each of them.

flab  title of the frequency axis.

alab  title of the amplitude axis.

flim  the range of frequency values.

alim  range of amplitude axis.

title  logical, if TRUE, adds a title with D value.

legend  logical, if TRUE adds a legend to the plot.

...  other plot graphical parameters.

Details

Both spectra are first transformed as probability mass functions (PMF). Spectral difference is then computed according to:

\[ D = \sum \frac{|spec1 - spec2|}{2}, \quad with \quad D \in [0, 1]. \]

Value

The difference is returned. This value is without unit. When plot is TRUE, both spectra and their difference surface are plotted on the same graph.

Note

This method can be used as a relative distance estimation between different spectra. The dB value obtained can be very different from the one visually estimated when looking at the graph (plot=TRUE).
**diffwave**

**Difference between two time waves**

**Description**

This function estimates the difference between two waves by computing the product between envelope surface difference and frequency surface difference.

**Author(s)**

Jerome Sueur, Sandrine Pavoine and Laurent Lellouch

**References**


**See Also**

spec, meanspec, corspec, simspec, diffcumspec, diffenv, kl.dist, ks.dist, logspec.dist, itakura.dist

**Examples**

```r
a <- noisew(f=8000, d=1)
b <- synth(f=8000, d=1, cf=2000)
c <- synth(f=8000, d=1, cf=1000)
d <- noisew(f=8000, d=1)
specia <- spec(a, f=8000, wl=512, at=0.5, plot=FALSE)
specb <- spec(b, f=8000, wl=512, at=0.5, plot=FALSE)
specc <- spec(c, f=8000, wl=512, at=0.5, plot=FALSE)
specd <- spec(d, f=8000, wl=512, at=0.5, plot=FALSE)
diffspec(specia, specb) #[1] 0 => similar spectra of course!
diffspec(specta, specb)
diffspec(speca, specb, plot=TRUE)
diffspec(specb, specc, plot=TRUE)
diffspec(specia, specd, plot=TRUE)
```

---

**diffwave**

**Difference between two time waves**

**Description**

This function estimates the difference between two waves by computing the product between envelope surface difference and frequency surface difference.
Usage

diffwave(wave1, wave2, f, wl = 512, envt = "hil", 
msmooth = NULL, ksmooth = NULL)

Arguments

wave1  a first R object.
wave2  a second R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
wl window length for spectral analysis (even number of points).
envt the type of envelope to be used: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope. See env.
msmooth a vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %). See env.
ksmooth kernel smooth via kernel. See env.

Details

This function computes the product between the values obtained with diffspec and diffenv functions. This then gives a global (time and frequency) estimation of dissimilarity. The frequency mean spectrum and the amplitude envelope needed for computing respectively diffspec and diffenv are automatically generated. They can be controlled through wl, smooth and ksmooth arguments respectively. See examples below and examples in diffspec and diffenv for implications on the results.

Value

A single value varying between 0 and 1 is returned. The value has no unit.

Note

This method can be used as a relative distance estimation between different waves.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

References


See Also

diffspec, diffenv
discrets

Examples

data(tico) ; tico <- tico@left
data(orni) ; orni <- orni@left
# selection in tico to have two waves of similar duration (length)
tico <- tico[1:length(orni)]
diffwave(tico,orni,f=22050)
# changing the frequency parameter (w1)
diffwave(tico,orni,f=22050,w1=1024)
# changing the temporal parameter (msmooth)
diffwave(tico,orni,f=22050,msmooth=c(20,0))

Time series discretisation

Description

This function transforms a numeric (time) series into a sequence of symbols

Usage

discrets(x, symb = 5, collapse = TRUE, plateau=1)

Arguments

- **x**: a vector, a matrix (first column), an object of class `ts`, `Sample` (left channel), or `Wave` (left channel).
- **symb**: the number of symbols used for the discretisation, can be set to 3 or 5 only.
- **collapse**: logical, if `TRUE`, the symbols are pasted in a character string of length 1.
- **plateau**: a numeric vector of length 1 taking the values 1 or 2 only. See details.

Details

The function partitions the numeric (time) series into a sequence of finite number of symbols. These symbols result of the comparaison of each series value with its temporal neighbours.

They are two discretisations available:

- when `symb` is set to 3, each value will be replaced by either:
  - *I* if the series is *Increasing*,
  - *D* if the series is *Decreasing*,
  - *F* if the series remains *Flat*,
- when `symb` is set to 5, each value will be replaced by either:
  - *I* if the series is *Increasing*,
  - *D* if the series is *Decreasing*,
  - *F* if the series remains *Flat*,
  - *P* if the series shows a *Peak*,
  - *T* if the series shows a *Trough*. 
The argument plateau can be used to control the way a plateau is encoded. A plateau is an elevated flat region that can be either considered a 'flat peak' encoded as \textit{PF...FP} (plateau = 1) or as an increase, a flat region and a decrease encoded as \textit{IF...FD} (plateau = 1). The default value (plateau = 1) refers to Cazelles et al. (2004).

**Value**

A character string of length 1 if collapse is TRUE. Otherwise, a character string of length \(n-2\) if symbol=5 (the first and last values cannot be replaced with a symbol) or \(n-1\) if symbol=3 (the first value cannot be replaced with a symbol.)

**Author(s)**

Jerome Sueur, improved by Laurent Lellouch

**References**


**See Also**

\texttt{symba}

**Examples**

\begin{verbatim}
# a random variable
discrets(rnorm(30))
discrets(rnorm(30),symb=3)
# a frequency spectrum
data(tico)
spec1<-spec(tico,f=22050,at=0.2,plot=FALSE)
discrets(spec1[,2])
\end{verbatim}

**drawenv**

\textit{Draw the amplitude envelope of a time wave}

**Description**

This function lets the user modifying the amplitude envelope of a time wave by drawing it with the graphics device

**Usage**

\texttt{drawenv(wave, f, n = 20, plot = FALSE, listen = FALSE, output = "matrix")}
Arguments

- **wave**: an R object.
- **f**: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- **n**: the maximum number of points to draw the new envelope. Valid values start at 1.
- **plot**: if TRUE returns the oscillogram of the new time wave (by default FALSE).
- **listen**: if TRUE the new sound is played back.
- **output**: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".

Details

The function first plots an oscillogram view of `wave`. The user has then to choose points on the positive side of the y-axis (amplitude). The junction of these points will draw a new amplitude envelope. The order of points along the x-axis (time) is not important but points cannot be cancelled. When this process is finished the new time wave is returned in the console or as an oscillogram in a second graphics device if `plot` is TRUE. The function uses `locator`.

Value

If `plot` is FALSE, a new wave is returned. The class of the returned object is set with the argument `output`.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

- `setenv`, `env`, `synth`

Examples

```
## Not run:
a<-synth(d=1,f=22050,cf=1000)
# drawenv(a,f=22050,plot=TRUE)
# choose points on the oscillogram view to draw a new envelope
# stop (ESC on Windows; right mouse button on Linux)
# check the result on the second graphics device opened thanks to plot=TRUE

## End(Not run)
```
drawfilter

Draw the amplitude profile of a frequency filter

Description

This function lets the user drawing the amplitude profile of a frequency filter.

Usage

drawfilter(f, n = 256, continuous = TRUE, discrete = TRUE)

Arguments

- **f**: a numeric vector of length 1 for the sampling frequency of the object to be filtered (in Hz).
- **n**: a numeric vector of length 1 for the length (i.e. number of points) of the filter. By default = 256 to fit with a FIR with \( \text{wl} = 512 \).
- **continuous**: a logical (TRUE by default) to draw a continuous filter.
- **discrete**: a logical (TRUE by default) to draw a discrete filter.

Details

If the same frequency of a discrete filter is selected twice then the sum of the amplitudes of the two selections is used. If both arguments `continuous` and `discrete` are set to `TRUE` and if frequencies selected overlap between the two filters then only the frequencies of the discrete filter are considered.

Value

The function returns a two-column matrix, the first column is the frequency in kHz and the second column is the amplitude of the filter.

Note

This function can be used to prepare bandpass or bandstop custom filters to be used with `fir` and `ffilter`. See examples.

Author(s)

Laurent Lellouch

See Also

`fir`, `squarefilter`, `combfilter`, `ffilter`, `drawenv`
### Examples

```r
## Not run:
f <- 8000
a <- noisew(f=f, d=1)
## bandpass continuous and discrete
cont.disc <- drawfilter(f=f/2)
a.cont.disc <- fir(a, f=f, custom=cont.disc)
spectro(a.cont.disc, f=f)
## bandpass continuous only
cont <- drawfilter(f=f/2, discrete=FALSE)
a.cont <- fir(a, f=f, custom=cont)
spectro(a.cont, f=f)
## bandstop continuous only
cont.stop <- drawfilter(f=f/2, discrete=FALSE)
a.cont.stop <- fir(a, f=f, custom=cont.stop, bandpass=FALSE)
spectro(a.cont.stop, f=f)
## bandpass discrete only
disc <- drawfilter(f=f/2, continuous=FALSE)
a.disc <- fir(a, f=f, custom=disc, bandpass=FALSE)
spectro(a.disc, f=f)
## End(Not run)
```

---

### duration

Duration of a time wave

#### Description

Returns the duration (in second) of a time wave

#### Usage

```r
duration(wave, f)
```

#### Arguments

- **wave**: an R object.
- **f**: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.

#### Value

A numeric vector of length 1 returning the duration in second.

#### Author(s)

Jerome Sueur
Examples

data(tico)
duration(tico)

dynoscillo  Dynamic oscillogram

Description

This graphical function displays a time wave as an windowed oscillogram.

Usage

dynoscillo(wave, f, wd = NULL, wl = NULL, wnb = NULL, title = TRUE, ...)

Arguments

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wd`: a numerical vector, duration of the window (in seconds).
- `wl`: a numerical vector, length of the window (in number of points).
- `wnb`: a numerical vector, number of windows (no unit).
- `title`: a logical, if TRUE displays a title with information regarding window size and number.
- `...`: other `plot` graphical parameters.

Details

The arguments `wd`, `wl` and `wn` have to be used isolated, not in conjunction. They basically do the same, i.e., they set the duration of the zooming window that is sliding along the signal. For instance, for a 5 seconds sound with a sampling rate (`f`) at 44.1 kHz, `wl = 4096` is equivalent to `wd = 4096 / 44100 = 0.093 s` and equivalent to `wnb = 5*4096 / 44100 = 53`.

Note

This function requires the package `rpanel`.

Author(s)

Jerome Sueur

See Also

oscillo, oscilloST, dynspec.
**dynspec**

**Examples**

```r
## Not run:
require(rrpanel)
data(tico)
dynoscillo(tico, wn=4)
## End(Not run)
```

---

**Dynamic sliding spectrum**

**Description**

This function plots dynamically a sliding spectrum along a time wave. This basically corresponds to a short-term Fourier transform.

**Usage**

```r
dynspec(wave, f, wl = 512, wn = "hanning", zp = 0,
ovlp = 0, fftw = FALSE, norm = FALSE, dB = NULL, dBref = NULL, plot = TRUE,
title = TRUE, osc = FALSE,
tlab = "Time (s)", flab = "Frequency (kHz)",
alab = "Amplitude", alim = NULL, flim = c(0, f/2000),
type = "l", from = NULL, to = NULL, envt = NULL,
msmooth = NULL, ksmooth = NULL, colspec = "black",
coltitle = "black", colbg = "white", colline = "black",
colaxis = "black", collab = "black", cexlab = 1,
fontlab = 1, colwave = "black",
coly0 = "lightgrey", colcursor = "red", bty = "l")
```

**Arguments**

- `wave` an R object.
- `f` sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl` if at is not null, length of the window for the analysis (even number of points, by defaults = 512).
- `wn` window name, see `ftwindow` (by default "hanning").
- `zp` zero-padding (even number of points), see Details.
- `ovlp` overlap between two successive windows (in %).
- `fftw` if TRUE calls the function FFT of the library `fftw`. See Notes of the spectro.
- `norm` logical, if TRUE compute a normalised sliding spectrum.
- `dB` a character string specifying the type dB to return: "max0" for a maximum dB value at 0, "A", "B", "C" and "D" for common dB weights.
- `dBref` a dB reference value when dB is not NULL. NULL by default but should be set to 2*10e-5 for a 20 microPa reference (SPL).
plot logical, if TRUE plots in an ew graphics device the successive spectra sliding along the time wave (by default TRUE).

title logical, if TRUE adds a title with the time position of the current spectrum along the time wave.

osc logical, if TRUE plots an oscillogram beneath the sliding spectrum with a cursor showing the position of the current spectrum (by default FALSE).

tlab title of the time axis.

flab title of the frequency axis.
alab title of the amplitude axis.

flim range of frequency axis.
alim range of amplitude axis.
type type of plot that should be drawn for the sliding spectrum. See plot for details (by default "l" for lines).

from start mark where to compute the sliding spectrum (in s).
to end mark where to compute the sliding spectrum (in s).

envt the type of envelope to be plotted: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope. See env.

msmooth when env is not NULL, a vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %). See env.

ksmooth when env is not NULL, kernel smooth via kernel. See env.

colspec colour of the sliding spectrum.
coltitle if title is TRUE, colour of the title.
colbg background colour.
colline colour of axes line.
colaxis colour of the axes.
collab colour of axes title.
cexlab character size for axes title.
fontlab font for axes title.
colwave colour of the oscillogram or of the envelope (only when osc is TRUE).
coly0 colour of the y=0 line (only when osc is TRUE).
colcursor colour of oscillogram cursor (only when osc is TRUE).
bty the type of box to be drawn around the oscillogram (only when osc is TRUE).

Details

Use the slider panel to move along the time wave. Use the argument norm if you wish to have each spectrum normalised, i.e. with values between 0 and 1 or maximised to 0 dB when dB is TRUE. The function requires the package rpanel that is based on the package tcltk.
Value

This function returns a list of three items:

- **time**: a numeric vector corresponding to the time axis.
- **freq**: a numeric vector corresponding to the frequency axis.
- **amp**: a numeric matrix corresponding to the amplitude values. Each column is a Fourier transform of length $wl/2$.

Note

This function is very similar to a spectrogram. See the Details of `spectro` for some information regarding the short term Fourier transform.

Author(s)

Jerome Sueur and Caroline Simonis

See Also

`spectro`, `spectro3D`, `wf`, `spec`, `dynspectro`, `fft`, `oscillo`, `env`.

Examples

```r
## Not run:
data(sheep)
require(rpanel)
dynspec(sheep, f=8000, wl=1024, ovlp=50, osc=TRUE)
## End(Not run)
```

dynspectro  Dynamic sliding spectrogram

Description

This function plots dynamically a sliding spectrogram along a time wave.

Usage

dynspectro(wave, f, slidframe = 10,
wl = 512, wn = "hanning", zp = 0, ovlp = 75,
fftw = FALSE, dB = TRUE, plot = TRUE,
title = TRUE, osc = FALSE,
tlab = "Time (s)", flab = "Frequency (kHz)", alab = "Amplitude",
from = NULL, to = NULL,
collevels = NULL, palette = spectro.colors,
envt = NULL, msmooth = NULL, ksmooth = NULL)
Arguments

`wave` an R object.

`f` sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.

`slidframe` size of the sliding frame (in percent of the wave duration).

`wl` if `at` is not null, length of the window for the analysis (even number of points, by defaults = 512).

`wn` window name, see `ftwindow` (by default “hanning”).

`zp` zero-padding (even number of points), see Details.

`ovlp` overlap between two successive windows (in %).

`fftw` if TRUE calls the function FFT of the library `fftw`. See Notes of the `spectro`.

`dB` a logical, if TRUE then uses dB values

`plot` logical, if TRUE plots in an new graphics device the successive spectrograms sliding along the time wave (by default TRUE).

`title` logical, if TRUE adds a title with the time position of the current spectrogram along the time wave.

`osc` logical, if TRUE plots an oscillogram beneath the sliding spectrogram with a cursor showing the position of the current spectrum (by default FALSE).

`tlab` title of the time axis.

`flab` title of the frequency axis.

`alab` title of the amplitude axis.

`from` start mark where to compute the sliding spectrogram (in s).

`to` end mark where to compute the sliding spectrogram (in s).

`collevels` a set of levels which are used to partition the amplitude range of the spectrogram.

`palette` a color palette function to be used to assign colors in the plot.

`envt` the type of envelope to be plotted: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope. See env.

`msmooth` when env is not NULL, a vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %). See env.

`ksmooth` when env is not NULL, kernel smooth via `kernel`. See env.

`coltitle` if `title` is TRUE, colour of the title.

`colbg` background colour.

`colline` colour of axes line.
colaxis  colour of the axes.
collab   colour of axes title.
cexlab   character size for axes title.
fontlab  font for axes title.
colwave  colour of the oscillogram or of the envelope (only when osc is TRUE).
coly0    colour of the y=0 line (only when osc is TRUE).
colcursor colour of oscillogram cursor (only when osc is TRUE).
bty      the type of box to be drawn around the oscillogram (only when osc is TRUE).

Details

Use the slider panel to move along the time wave.
The function requires the package rpanel that is based on the package tcltk.
The function is mainly written for inspecting long sounds.
The function is based on image for fast display when spectro is based on filled.contour. Displaying the amplitude envelope with the argument envt can slow down significantly the display.

Value

This function returns a list of three items:

- time  a numeric vector corresponding to the time axis.
- freq  a numeric vector corresponding to the frequency axis.
- amp   a numeric matrix corresponding to the amplitude values. Each column is a Fourier transform of length wl/2.

Note

This function is very similar to a spectrogram. See the Details of spectro for some information regarding the short term Fourier transform.

Author(s)

David Pinaud and Jerome Sueur

See Also

spectro, spectro3D, wf, spec, dynspec, fft, oscillo, env.

Examples

```r
## Not run:
data(sheep)require(rpanel)
dynspectro(sheep, ovlp=95, osc=TRUE)

## End(Not run)
```
Description

This function generates echoes of a time wave.

Usage

```r
echo(wave, f, amp, delay, plot = FALSE,
     listen = FALSE, output = "matrix", ...)
```

Arguments

- **wave**: an R object.
- **f**: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- **amp**: a vector describing the relative amplitude of the successive echoes. Each value of the vector should be in [0,1]
- **delay**: a vector describing the time delays of the successive echoes from the beginning of `wave` (in s.)
- **plot**: logical, if TRUE returns an oscillographic plot of the wave modified (by default FALSE).
- **listen**: if TRUE the new sound is played back.
- **output**: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
- **...**: other oscillo graphical parameters.

Details

- `amp` and `delay` should strictly have the same length corresponding to the number of desired echoes.

Value

If `plot` is FALSE, a new wave is returned. The class of the returned object is set with the argument `output`.

Note

This function is based on a convolution (`convolve`) between the input wave and a pulse echo filter.

Author(s)

Jerome Sueur <sueur@mnhn.fr>
References


See Also

synth

Examples

# generation of the input wave
a <- synth(f=11025,d=1,cf=2000,shape="tria",am=c(50,10),fm=c(1000,10,1000,0,0))
# generation of three echoes
# with respectively a relative amplitude of 0.8, 0.4, and 0.2
# and with a delay of 1s, 2s, and 3s from the beginning of the input wave
echo <- echo(a,f=11025,amp=c(0.8,0.4,0.2),delay=c(1,2,3))
# another echo with time delays overlapping with the input wave
other <- echo(a,f=11025,amp=c(0.4,0.2,0.4),delay=c(0.6,0.8,1.5))

env

Amplitude envelope of a time wave

Description

This function returns the absolute or Hilbert amplitude envelope of a time wave.

Usage

env(wave, f, envt = "hil",
msmooth = NULL, ksmooth = NULL, ssmooth = NULL,
fftw = FALSE, norm = FALSE,
plot = TRUE, k = 1, j = 1, ...)

Arguments

wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
envt the type of envelope to be returned: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope. See Details section.
msmooth a vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %). See examples.
ksmooth kernel smooth via kernel. See examples.
sssmooth sum smooth.
fftw if TRUE calls the function FFT of the library fftw for faster computation for the Hilbert amplitude envelope (envt="hil") and/or for kernel smoothing (ksmooth).
norm a logical, if TRUE the amplitude of the envelope is normalised between 0 and 1.
plot logical, if TRUE returns a plot of wave envelope (by default TRUE).
k number of horizontal sections when plot is TRUE (by default =1).
 j number of vertical sections when plot is TRUE (by default =1).
... other oscillo graphical parameters.

Details
When envt is set as "abs", the amplitude envelope returned is the absolute value of wave.
When envt is set as "hil", the amplitude envelope returned is the modulus (Mod) of the analytical signal of wave obtained through the Hilbert transform (hilbert).

Value
Data are returned as one-column matrix when plot is FALSE.

Note
Be aware that smoothing with either msmooth or ksmooth changes the original number of points describing wave.

Author(s)
Jerome Sueur. Implementation of ’fftw’ argument by Jean Marchal and Francois Fabianek.

See Also
oscillo.hilbert

Examples

data(tico)
# Hilbert amplitude envelope
env(tico)
# absolute amplitude envelope
env(tico, envt="abs")
# smoothing with a 10 points and 50% overlapping mean sliding window
env(tico, msmooth=c(10,50))
# smoothing kernel
env(tico, ksmooth=kernel("daniell",10))
# sum smooth
env(tico, ssmooth=50)
# overplot of oscillographic and envelope representations
oscillo(tico)
par(new=TRUE)
env(tico, colwave=2)
Export sound data

Description

Export sound data as a text file that can be read by a sound player like 'Goldwave'

Usage

```r
export(wave, f = NULL, filename = NULL, header=TRUE, ...)
```

Arguments

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `filename`: name of the new file. (by default the name of `wave`).
- `header`: either a logical or a character vector, if `TRUE` add a header to be read by Goldwave, if `FALSE` does not add any header, if a character vector add the character vector as a header.
- `...`: other `write.table` parameters.

Details

Creates a new text file with a header describing the main features of the sound (`wave`). For instance, for a 2 s sound with a sampling frequency of 8000 Hz, the header will be: [ASCII 8000Hz, Channels: 1, Samples: 160000, Flags: 0]. This type of file can be read by sound players like Goldwave (http://www.goldwave.com/).

Author(s)

Jerome Sueur <sueur@mnhn.fr>

Examples

```r
a<-synth(f=8000,d=2,cf=2000,plot=FALSE)
export(a,f=8000)
unlink("a.txt")
```
fadew

Fade in and fade out of a time wave

Description

This function applies a “fade in” and/or a “fade out” to a time wave following a linear, exponential or cosinus-like shape.

Usage

fadew(wave, f, din = 0, dout = 0, shape = "linear", plot = FALSE, listen = FALSE, output = "matrix", ...)

Arguments

wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
din fade in duration.
dout fade out duration.
shape fade shape, "linear", "exp" for exponential, "cos" for cosinus-like, (by default "linear").
plot logical, if TRUE returns an oscillographic plot of the wave modified (by default FALSE).
listen if TRUE the new sound is played back.
output character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
... other oscillo graphical parameters.

Value

If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

oscillo, addsilw, cutw, deletew, mutew, pastew, revw, zapsilw
Examples

```r
a<-noise(d=5,f=4000)
op<-par(mfrow=c(3,1))
fadew(a,f=4000,din=1,dout=2,plot=TRUE,title="Linear",cexlab=0.8)
fadew(a,f=4000,din=1,dout=2,shape="exp",plot=TRUE,title="Exponential shape",colwave="blue",coltitle="blue",cexlab=0.8)
fadew(a,f=4000,din=1,dout=2,shape="cos",plot=TRUE,title="Cosinus-like shape",colwave="red",coltitle="red",cexlab=0.8)
par(op)
```

Description

This graphical function returns a frequency spectrum as a bar plot.

Usage

```r
fbands(spec, f = NULL, bands = 10, width = FALSE, mel = FALSE, plot = TRUE,
xlab = NULL, ylab = "Relative amplitude", ...)
```

Arguments

- `spec`: a data set resulting of a spectral analysis obtained with `spec` or `meanspec`. Can be in dB.
- `f`: sampling frequency of `spec` (in Hz). Not requested if the first column of `spec` contains the frequency axis.
- `bands`: a numeric vector. If vector of length 1, then sets the number of bands dividing in equal parts the spectrum. If of length > 1, then takes the values as kHz limits of the bands dividing the spectrum. These bands can be of different size. See details and examples.
- `width`: logical, if TRUE and that `bands` is an irregular series of values, then the width of the bands will be proportional to the frequency limits defined in `bands`.
- `mel`: a logical, if TRUE the (htk-)mel scale is used.
- `plot`: logical, if TRUE, a plot showing the peaks is returned.
- `xlab`: label of the x-axis.
- `ylab`: label of the y-axis.
- `...`: other `plot` graphical parameters.
Details

The function proceeds as follows

• divides the spectrum in bands. The limits of the bands are set with the argument bands. There are two options:
  – you set a number of bands with equal size by giving a single value to bands. For instance, setting bands to a value of 10 will slice the spectrum in 10 equal parts and return 10 local peaks.
  – you set the limits of the bands. This is achieve by giving a numeric vector to bands. The limits can follow a regular or irregular series. For instance attributing the vector c(0,2,4,8) will generate the following bands [0,2[, [2,4[, [4,8] kHz. Be aware that the last value should not exceed half the sampling frequency used to obtain the spectrum spec.

• uses the function barplot.

Value

A two-column matrix, the first column corresponding to the frequency values (x-axis, mean of the bars limits) and the second column corresponding to height values (y-axis) of the bars.

Note

The value below bars is the mean between the corresponding frequency limits.

Author(s)

Jerome Sueur, improved by Laurent Lellouch

See Also

meanspec, spec, barplot.

Examples

data(sheep)
spec <- meanspec(sheep, f=8000, plot=FALSE)
# default plot
fbands(spec)
# setting a specific number of bands
fbands(spec, bands=6)
#setting specific regular bands limits
fbands(spec, bands=seq(0,4,by=0.25))
# some plot tuning
op <- par(las=1)
fbands(spec, bands=seq(0,4,by=0.1),
   horiz=TRUE, col=heat.colors(41),
   xlab="", ylab="",
   cex.axis=0.75, cex.names = 0.75,
   axes=FALSE)
par(op)
# showing or not the width of the bands
The function computes the altered frequency of a moving source due to the Doppler effect.

**Usage**

```r
fdoppler(f, c = 340, vs, vo = 0, movs = "toward", movo = "toward")
```

**Arguments**

- `f`: original frequency produced by the source (in Hz or kHz)
- `c`: speed of sound in meters/second.
- `vs`: speed of the source in meters/second.
- `vo`: speed of the observer in meters/second. The observer is static by default *i.e.* `vo = 0`
- `movs`: movement direction of the source in relation with observer position, either "toward" (by default) or "away".
- `movo`: movement direction of the observer in relation with the source position, either "toward" (by default, but be aware that the observer is static by default) or "away".
Details

The altered frequency \( f' \) is computed according to:

\[
  f' = f \times \frac{c \pm v_o}{c \pm v_s}
\]

with \( f \) = original frequency produced by the source (in Hz or kHz),
\( vs \) = speed of the source,
\( vo \) = speed of the observer.

Value

The altered frequency is returned in a vector.

Note

You can use `wasp` to have exact values of \( c \). See examples.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

`wasp`

Examples

```r
# a 400 Hz source moving toward or away from the observer at 85 m/s
fdoppler(f=400,vs=85)
# [1] 533.3333
fdoppler(f=400,vs=85,movs="away")
# [1] 320
# use wasp() if you wish to have exact sound speed at a specific temperature
fdoppler(f=wasp(f=400,t=25)&c, vs=85)
# [1] 461.8667
# Doppler effect at different source speeds
f<-seq(1,10,by=1); lf<-length(f)
v<-seq(10,300,by=20); lv<-length(v)
res<-matrix(numeric(lf+lv),ncol=lv)
for(i in 1:lv) res[,i]<-fdoppler(f=f,vs=v[i])
op<-par(bg="lightgrey")
matplot(x=f,y=res,type="l",lty=1,las=1,col= spectro.colors(lv),
  xlab="Source frequency (kHz)", ylab="Altered frequency (kHz)"
  legend("topleft",legend=paste(as.character(v),"m/s"),
  lty=1,col= spectro.colors(lv))
title(main="Doppler effect at different source speeds")
par(op)
```
**ffilter**

*Frequency filter*

**Description**

This function filters out a selected frequency section of a time wave (low-pass, high-pass, low-stop, high-stop, bandpass or bandstop frequency filter).

**Usage**

```r
ffilter(wave, f, from = NULL, to = NULL, bandpass = TRUE, custom = NULL, wl = 1024, ovlp = 75, wn = "hanning", fftw = FALSE, rescale=FALSE, listen=FALSE, output="matrix")
```

**Arguments**

- `wave` an R object.
- `f` sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `from` start frequency (in Hz) where to apply the filter.
- `to` end frequency (in Hz) where to apply the filter.
- `bandpass` if TRUE a band-pass filter is applied between `from` and `to`, if FALSE a band-stop filter is applied between `from` and `to` (by default TRUE).
- `custom` a vector describing the frequency response of a custom filter. This can be manually generated or obtained with `spec` and `meanspec`. The length of the vector should be half the length of `wl`. See examples.
- `wl` window length for the analysis (even number of points).
- `ovlp` overlap between successive FFT windows (in %).
- `wn` window name, see `ftwindow` (by default "hanning").
- `fftw` if TRUE calls the function FFT of the library `fftw`. See Notes of the `spectro`.
- `rescale` a logical, if TRUE then the sample values of new wave (output) are rescaled according to the sample values of `wave` (input).
- `listen` a logical, if TRUE the new sound is played back.
- `output` character string, the class of the object to return, either "matrix", "Wave", "audioSample" or "ts".

**Details**

A short-term Fourier transform is first applied to the signal (see `spectro`), then the frequency filter is applied and the new signal is eventually generated using the reverse of the Fourier Transform (`istft`).

There is therefore neither temporal modifications nor amplitude modifications.
Value

If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Author(s)

Jerome Sueur

See Also

afilter, lfs, fir, preemphasis, combfilter, bwfilter

Examples

```r
a<-noisew(f=8000, d=1)
# low-pass
b<-ffilter(a, f=8000, to=1500)
spectro(b, f=8000, wl=512)
# high-pass
c<-ffilter(a, f=8000, from=2500)
spectro(c, f=8000, wl=512)
# band-pass
d<-ffilter(a, f=8000, from=1000, to=2000)
spectro(d, f=8000, wl=512)
# band-stop
e<-ffilter(a, f=8000, from=1500, to=2500, bandpass=FALSE)
spectro(e, f=8000, wl=512)
# custom
myfilter!<-rep(c(rep(0,64),rep(1,64)),4)
g<-ffilter(a, f=8000, custom=myfilter!)
spectro(g, f=8000)
```

---

**field**

*Near field and far field limits*

Description

This function helps in knowing whether you are working in the near or far field.

Usage

`field(f, d)`

Arguments

- `f` frequency (Hz)
- `d` distance from the sound source (m)
Details

Areas very close to the sound source are in the near-field where the contribution of particle velocity to sound energy is greater than that of sound pressure and where these components are not in phase. Sound propagation properties are also different near or far from the source. It is therefore important to know where the microphone was from the source.

To know this, the product k*d is computed according to:

\[ k \times d = \frac{f}{c} \times d \]

with \( d \) = distance from the source (m), \( f \) = frequency (Hz) and \( c \) = sound celerity (m/s).

If k*d is greatly inferior 1 then the microphone is in the near field.

The decision help returned by the function follows the rule:

- far field: \( k \times d > 1 \)
- between near and far field limits: \( 0.1 \leq k \times d \leq 1 \)
- near field: \( k \times d < 0.1 \)

Value

A list of two values is returned:

- \( kd \) the numeric value k*d used to take a decision
- \( d \) a character string giving the help decision.

Note

This function works for air-borne sound only.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

Examples

```r
# 1 kHz near field at 1 cm from the source
field(f=1000,d=0.01)
# playing with distance from source and sound frequency
op<-par(bg="lightgrey")
D<-seq(0.01,0.5,by=0.01); nD<-length(D)
F<-seq(100,1000,by=25); nF<-length(F)
a<-matrix(numeric(nD*nF),nrow=nD)
for(i in 1:nF) a[,]<-field(f=F[i],d=D)$kd
matplot(x=D,y=a,type="l",lty=1,col= spectro.colors(nF),
       xlab="Distance from the source (m)", ylab="k*d")
title("Variation of the product k*d with distance and frequency")
```
Description

This function is a FIR filter that filters out a selected frequency section of a time wave (low-pass, high-pass, low-stop, high-stop, bandpass or bandstop frequency filter).

Usage

```r
fir(wave, f, from = NULL, to = NULL, bandpass = TRUE, custom = NULL, 
    wl = 512, wn = "hanning", rescale = FALSE, listen = FALSE, output = "matrix")
```

Arguments

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `from`: start frequency (in Hz) where to apply the filter.
- `to`: end frequency (in Hz) where to apply the filter.
- `bandpass`: if TRUE a band-pass filter is applied between `from` and `to`, if not NULL a band-stop filter is applied between `from` and `to` (by default NULL).
- `custom`: a vector describing the frequency response of a custom filter. This can be manually generated or obtained with `spec` and `meanspec`. `wl` is no more required. See examples.
- `wl`: window length of the impulse filter (even number of points).
- `wn`: window name, see `ftwindow` (by default "hanning").
- `rescale`: a logical, if TRUE then the sample values of new wave (output) are rescaled according to the sample values of `wave` (input).
- `listen`: a logical, if TRUE the new sound is played back.
- `output`: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".

Details

This function is based on the reverse of the Fourier Transform (`fft`) and on a convolution (`convolve`) between the wave to be filtered and the impulse filter.
Value

A new wave is returned. The class of the returned object is set with the argument output.

Author(s)

Jerome Sueur

References


See Also

ffilter, bwfilter, preemphasis, lfs, afilter

Examples

```r
a <- noise(f=8000, d=1)
# low-pass
b <- fir(a, f=8000, to=1500)
spectro(b, f=8000)
# high-pass
c <- fir(a, f=8000, from=2500)
spectro(c, f=8000)
# band-pass
d <- fir(a, f=8000, from=1000, to=2000)
spectro(d, f=8000)
# band-stop
e <- fir(a, f=8000, from=1500, to=2500, bandpass=FALSE)
spectro(e, f=8000)
# custom filter manually generated
myfilter1 <- rep(c(rep(0.32), rep(1, 32)), 4)
g <- fir(a, f=8000, custom=myfilter1)
spectro(g, f=8000)
# custom filter generated using spec()
data(tico)
myfilter2 <- spec(tico, f=22050, at=0.7, wl=512, plot=FALSE)
b <- noise(d=1, f=22050)
h <- fir(b, f=22050, custom=myfilter2)
spectro(h, f=22050)
```
Usage

fma(wave, f, threshold = NULL, plot = TRUE, ...)

Arguments

wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
threshold amplitude threshold for signal detection (in %).
plot logical, if TRUE the spectrum of the instantaneous frequency (by default TRUE).
... other spec parameters.

Details

This function is based on ifreq and spec. The instantaneous frequency of wave is first computed and the spectrum of this frequency modulation is then processed. All env and spec arguments can be set up.

Value

If plot is FALSE, fma returns a numeric vector corresponding to the computed spectrum. If peaks is not NULL, fma returns a list with two elements:

spec the spectrum computed
peaks the peaks values (in kHz).

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

ifreq, hilbert, spec, ama

Examples

# a sound with a 1 kHz sinusoid FM
a<-synth(d=1, f=8000, cf=1500, fm=c(1000,1000,0,0,0), output="Wave")
fma(a)
Frequency peak detection

Description
This function searches for peaks of a frequency spectrum.

Usage
```
fpeaks(spec, f = NULL,
nmax = NULL, amp = NULL, freq = NULL, threshold = NULL,
mel = FALSE,
plot = TRUE, title = TRUE,
xlab = NULL, ylab = "Amplitude",
labels = TRUE, digits = 2,
legend = TRUE, collab = "red", ...)```

Arguments
- `spec`: a data set resulting of a spectral analysis obtained with `spec` or `meanspec`. Can be in dB.
- `f`: sampling frequency of `spec` (in Hz). Not requested if the first column of `spec` contains the frequency axis.
- `nmax`: maximal number of peaks detected. Overrides `amp` and `freq`. See details.
- `amp`: amplitude slope parameter, a numeric vector of length 2. Refers to the amplitude slopes of the peak. The first value is the left slope and the second value is the right slope. Only peaks with higher slopes than threshold values will be kept. See details.
- `freq`: frequency threshold parameter (in Hz). If the frequency difference of two successive peaks is less than this threshold, then the peak of highest amplitude will be kept only. See details.
- `threshold`: amplitude threshold parameter. Only peaks above this threshold will be considered. See details.
- `mel`: a logical, if TRUE the (htk-)mel scale is used.
- `plot`: logical, if TRUE, a plot showing the peaks is returned.
- `title`: logical, if TRUE add the number of peaks detected as a plot title.
- `xlab`: label of the x-axis.
- `ylab`: label of the y-axis.
- `labels`: logical, if TRUE peak labels are plotted.
- `digits`: if `labels` is TRUE, the number of decimal places (`round`) for the peak labels.
- `legend`: logical, if TRUE a legend returning the different selection parameters (nmax, amp, freq, threshold, threshold) is added to the plot.
- `collab`: labels color.
- `...`: other `plot` graphical parameters.
Details

Here are some details regarding the different selection parameters:

- *nmax*: this parameter is to be used if you wish to get a specific number of peaks. The peaks selected are those with the highest slopes. It then does not work in conjunction with the other parameters.

- *freq*: this parameter allows to remove from the selection successive peaks with a small frequency difference. Imagine you have two successive peaks at 1200 Hz and 1210 Hz and at 0.5 and 0.25 in amplitude. If you set *freq* to 50 Hz, then only the first peak will be kept.

- *amp*: this parameter allows to remove from the selection peaks with low slopes. You can make the selection on both slopes or on a single one. Imagine you have an asymmetric peak with a 0.01 left slope and a 0.02 right slope. The peak will be discarded for the following settings: both values higher than 0.02 (e.g. amp = c(0.03, 0.04)), the first value higher than 0.01 (e.g. amp = c(0.02, 0.001)), the second value higher than 0.02 (e.g. amp = c(0.001, 0.03)). If you do not want to apply the selection on one of the slopes use 0. For instance, a selection on the left slope only will be achieved with: amp = c(0.02, 0).

- *threshold*: this parameter can be used to do a rough selection on the spectrum. Peaks with an amplitude value (not a slope) lower than this threshold will be automatically discarded. This can be useful when you want to remove peaks of a low-amplitude background noise.

Value

A two-column matrix, the first column corresponding to the frequency values (x-axis) and the second column corresponding to the amplitude values (y-axis) of the peaks.

Note

You can also use `fpeaks` with other kind of spectrum, for instance a cepstral spectrum. See examples.

Author(s)

Jerome Sueur and Amandine Gasc

See Also

`localpeaks`, `meanspec`, `spec`

Examples

data(tico)
spec <- meanspec(tico, f=22050, plot=FALSE)
specdB <- meanspec(tico, dB="max0", plot=FALSE)
# all peaks
fpeaks(spec)
# 10 highest peaks
fpeaks(spec, nmax=10)
# highest peak (ie dominant frequency)
ftwindow

**Fourier transform windows**

**Description**

Generates different Fourier Transform windows.
Usage

```r
ftwindow(wl, wn = "hamming", 
correction = c("none", "amplitude", "energy"))
```

Arguments

- `wl`: window length
- `wn`: window name: bartlett, blackman, flattop, hamming, hanning, or rectangle (by default hamming).
- `correction`: a character vector of length 1 to apply an amplitude ("amplitude") or an energy ("energy") correction to the FT window. By default no correction is applied ("none").

Value

A vector of length `wl`.

Note

Try the example to see windows shape.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

References


See Also

covspectro, dfreq, meanspec, spec, spectro, spectro3D

Examples

```r
a<-ftwindow(512)
b<-ftwindow(512,wn="bartlett")
c<-ftwindow(512,wn="blackman")
d<-ftwindow(512,wn="flattop")
e<-ftwindow(512,wn="hanning")
f<-ftwindow(512,wn="rectangle")
all<-cbind(a,b,c,d,e,f)
matplot(all,type="l",col=1:6,lty=1:6)
legend(legend=c("hamming","bartlett","blackman","flattop","hanning","rectangle"), x=380,y=0.95,col=1:6,lty=1:6,cex=0.75)
```
**fund**

**Fundamental frequency track**

**Description**

This function estimates the fundamental frequency through a short-term cepstral transform.

**Usage**

```
fund(wave, f, wl = 512, ovlp = 0, fmax = f/2, threshold = NULL,
at = NULL, from = NULL, to = NULL,
plot = TRUE, xlab = "Time (s)", ylab = "Frequency (kHz)",
ylim = c(0, f/2000), pb = FALSE, ...)
```

**Arguments**

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl`: if `at` is not null, length of the window for the analysis (even number of points, by defaults = 512).
- `ovlp`: overlap between two successive windows (in %).
- `fmax`: the maximum frequency to detect (in Hz).
- `threshold`: amplitude threshold for signal detection (in %).
- `at`: position where the estimate the fundamental frequency (in s).
- `from`: start position where to compute the fundamental frequency (in s).
- `to`: end position to compute the fundamental frequency (in s).
- `plot`: logical, if TRUE plots the fundamental frequency modulations against time (by default TRUE).
- `xlab`: title of the time axis (s).
- `ylab`: title of the frequency axis (Hz).
- `ylim`: the range of frequency values.
- `pb`: if TRUE returns a text progress bar in the console.
- `...`: other `plot` graphical parameters.

**Value**

When `plot` is FALSE, `fund` returns a two-column matrix, the first column corresponding to time in seconds (x-axis) and the second column corresponding to to fundamental frequency in kHz (y-axis). NA corresponds to pause sections in `wave` (see `threshold`). No plot is produced when using `at`. 
Note

This function is based on ceps.

Author(s)

Jerome Sueur <sueur@mnhn.fr>.

References


See Also

ceps.autoc, cestro, ceps

Examples

data(sheep)
# estimate the fundamental frequency at a single position
fund(sheep, f=8000, fmax=300, at=1, plot=FALSE)
# track the fundamental frequency along time
fund(sheep, f=8000, fmax=300, type="1")
# with 50% overlap between successive sliding windows, time zoom and
# amplitude filter (threshold)
fund(sheep, f=8000, fmax=300, type="b", ovlp=50, threshold=5, ylim=c(0,1), cex=0.5)
# overlaid on a spectrogram
spectro(sheep, f=8000, ovlp=75, zp=16, scale=FALSE, palette=reverse.gray.colors.2)
par(new=TRUE)
fund(sheep, f=8000, fmax=300, type="p", pch=24, ann=FALSE,
     xaxs="i", yaxs="i", col="black", bg="red", threshold=6)

---

**ggspectro**

2D-spectrogram of a time wave using ggplot2

Description

This function returns a ggplot object to draw a spectrogram with the package ggplot2. This is an alternative to spectro.

Usage

ggspectro(wave, f, tlab = "Time (s)",
flab = "Frequency (kHz)", alab = "Amplitude\n(dB)\n", ...)

Arguments

- wave: an R object.
- f: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
- tlab: label of the time axis.
- flab: label of the frequency axis.
- alab: label of the amplitude axis.
- ...: other non-graphical arguments to be passed to spectro (wl, ovlp etc).

Details

This function return the first layer (data and aesthetic mapping) of a ggplot2 plot. See the example section to understand how to build a spectrogram and consult ggplot2 help to get what you exactly need. There is no way to plot the oscillogram as spectro does.

Value

A ggplot layer.

Note

This function requires ggplot2 package.

Author(s)

Jerome Sueur

References


See Also

spectro, spectro3D, dynspec

Examples

```R
## Not run:
require(ggplot2)
## first layer
v <- ggspectro(tico, ovlp=50)
summary(v)
## using geom_tile ##
v + geom_tile(aes(fill = amplitude)) + stat_contour()
## coordinates flip (interest?)
v + geom_tile(aes(fill = amplitude)) + stat_contour() + coord_flip()
## using stat_contour ##
# default (not nice at all)
```
v + stat_contour(geom="polygon", aes(fill=..level..))
# set up to 30 color levels with the argument bins
(vv <- v + stat_contour(geom="polygon", aes(fill=..level..), bins=30))
# change the limits of amplitude and NA values as transparent
vv + scale_fill_continuous(name="Amplitude\n(dB)\n", limits=c(-30,0), na.value="transparent")
# Black-and-white theme
(vv + scale_fill_continuous(name="Amplitude\n(dB)\n", limits=c(-30,0),
  na.value="transparent", low="white", high="black") + theme_bw())
# Other colour scale (close to spectro() default output)
v + stat_contour(geom="polygon", aes(fill=..level..), bins=30)
  + scale_fill_gradientn(name="Amplitude\n(dB)\n", limits=c(-30,0),
    na.value="transparent", colours = spectro.colors(30))

## End(Not run)

---

**Total entropy**

**Description**

This function estimates the total entropy of a time wave.

**Usage**

```r
H(wave, f, wl = 512, envt="hil", msmooth = NULL, ksmooth = NULL)
```

**Arguments**

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl`: window length for spectral entropy analysis (even number of points). See `sh`.
- `envt`: the type of envelope to be used: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope. See `env`.
- `msmooth`: a vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %). See `env`.
- `ksmooth`: kernel smooth via `kernel`. See `env`.

**Details**

This function computes the product between the values obtained with `sh` and `th` functions.

This then gives a global (time and frequency) estimation of signal entropy.

The frequency mean spectrum and the amplitude envelope needed for computing respectively `sh` and `th` are automatically generated. They can be controlled through `wl` and `smooth` arguments respectively. See examples below and examples in `sh` and `th` for implications on the results.
Value
A single value varying between 0 and 1 is returned. The value has no unit.

Note
The entropy of a noisy signal will tend towards 1 whereas the entropy of a pure tone signal will tend towards 0.

Author(s)
Jerome Sueur <sueur@mnhn.fr>

References

See Also
sh, th, csh

Examples
```r
data(orni)
H(orni,f=22050)
  # changing the spectral parameter (wl)
H(orni,f=22050,wl=1024)
  # changing the temporal parameter (msmooth)
H(orni,f=22050,msmooth=c(20,0))
```

Description
This function returns the analytic signal of a time wave through Hilbert transform.

Usage
```r
hilbert(wave, f, fftw = FALSE)
```

Arguments
```r
wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
fftw if TRUE calls the function FFT of the library fftw for faster computation. See Notes of the function spectro.
```
Details

The analytic signal is useful to get the amplitude envelope (see argument henv of oscillo) and the instantaneous phase or frequency (see ifreq) of a time wave.

Value

hilbert returns the analytic signal as a complex matrix. The imaginary part of this matrix is the Hilbert transform.

Note

To get the Hilbert component only, use Im(Hilbert(wave)).

Author(s)

Jonathan Lees <jonathan.lees@unc.edu>. Implementation of 'fftw' argument by Jean Marchal and Francois Fabianek.

References


See Also

ifreq

Examples

```r
a <- synth(f=8000, d=1, cf=1000)
aa <- hilbert(a, f=8000)
```

```r
ifreq

Instantaneous frequency
```

Description

This function returns the instantaneous frequency (and/or phase) of a time wave through the computation of the analytic signal (Hilbert transform).

Usage

```r
ifreq(wave, f, phase = FALSE, threshold = NULL, plot = TRUE, xlab = "Time (s)", ylab = NULL, ylim = NULL, type = "l", ...)
```
Arguments

- **wave**: an R object.
- **f**: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- **phase**: if TRUE and `plot` is also TRUE plots the instantaneous phase instead of the instantaneous frequency.
- **threshold**: amplitude threshold for signal detection (in %).
- **plot**: logical, if TRUE plots the instantaneous frequency or phase against time (by default TRUE).
- **xlab**: title of the x axis.
- **ylab**: title of the y axis.
- **ylim**: the range of y values.
- **type**: if `plot` is TRUE, type of plot that should be drawn. See `plot` for details (by default "l" for lines).
- **...**: other `plot` graphical parameters.

Details

The instantaneous phase is the argument of the analytic signal obtained through the Hilbert transform. The instantaneous phase is then unwrapped and derived against time to get the instantaneous frequency. There may be some edge effects at both start and end of the time wave.

Value

If `plot` is FALSE, `ifreq` returns a list of two components:

- **f**: a two-column matrix, the first column corresponding to time in seconds (x-axis) and the second column corresponding to instantaneous frequency in kHz (y-axis).
- **p**: a two-column matrix, the first column corresponding to time in seconds (x-axis) and the second column corresponding to wrapped instantaneous phase in radians (y-axis).

Note

This function is based on the analytic signal obtained with the Hilbert transform (see `hilbert`). The function requires the package `signal`. The matrix describing the instantaneous phase has one more row than the one describing the instantaneous frequency.

Author(s)

Jerome Sueur <sueur@mnhn.fr>
References


See Also

hilbert, zc

Examples

# generate a sound with sine and linear frequency modulations
a<-synth(d=1, f=8000, cf=1500, fm=c(200,10,1000,0,0))
# plot on a single graphical device the instantaneous frequency and phase
op<par(mfrow=c(2,1))
ifreq(a,f=8000,main="Instantaneous frequency")
ifreq(a,f=8000,phase=TRUE,main="Instantaneous phase")
par(op)

---

istft

*Inverse of the short-term Fourier transform*

Description

This function returns a wave object from a complex STFT matrix by computing the inverse of the short-term Fourier transform (STFT)

Usage

`istft(stft, f, wl, ovlp=75, wn="hanning", output = "matrix")`

Arguments

- `f`: sampling frequency of the original wave object (in Hz)
- `wl`: FFT window length for the analysis (even number of points).
- `ovlp`: overlap between successive FFT windows (in %, by default 75%, see the Details section).
- `wn`: character string specifying the FFT window name, see `ftwindow` (by default "hanning").
- `output`: character string, the class of the object to return, either "matrix", "Wave", "audioSample" or "ts".
Details

The function is based on the inverse of the FFT (see `fft`) and on the overlap add (OLA) method. The overlap percentage must satisfy the Perfect Reconstruction OLA-constraint. For the most windows, this constraint is:

\[ ovlp = 100 \times \left(1 - \frac{1}{4 \times n}\right) , \]

with \( n \) being a positive integer. A default value is set to 75%. We suggest not to change it.

Value

A new wave is returned. The class of the returned object is set with the argument `output`.

Note

The `stft` input data must be complex. This function is used by `ffilter`, `lfs` to respectively filter in frequency and shift in frequency a sound. The function can be used to reconstruct or modify a sound. See examples.

Author(s)

Original Matlab code by Hristo Zhivomirov (Technical University of Varna, Bulgaria), translated and adapted to R by Jerome Sueur

See Also

`spectro`, `ffilter`, `lfs`

Examples

```r
## Not run:
# STFT and iSTFT parameters
wl <- 1024
ovlp <- 75
# reconstruction of the tico sound from the stft complex data matrix
data(tico)
data <- spectro(tico, wl=wl, ovlp=ovlp, plot=FALSE, norm=FALSE, dB=NULL, complex=TRUE) &
res <- istft(data, ovlp=ovlp, wn="hanning", wl=wl, f=22050, out="Wave")
spectro(res)
# a strange frequency filter
n <- noise(d=1, f=44100)
data <- spectro(n, f=44100, wl=wl, ovlp=ovlp, plot=FALSE, norm=FALSE, dB=NULL, complex=TRUE) &
data[64:192, 6:24] <- 0
nfilt <- istft(data, f=8000, wl=wl, ovlp=ovlp, output="Wave")
spectro(nfilt, wl=wl, ovlp=ovlp)
## End(Not run)
```
Description

Compare two distributions (e.g. two frequency spectra) by computing the Itakuro-Saito distance

Usage

```
itakura.dist(spec1, spec2, scale=FALSE)
```

Arguments

- `spec1`: any distribution, especially a spectrum obtained with `spec` or `meanspec` (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).
- `spec2`: any distribution, especially a spectrum obtained with `spec` or `meanspec` (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).
- `scale`: a logical, if TRUE the distance is scaled by dividing the distance by the length of spec1 (or spec2).

Details

The Itakura-Saito (I-S) distance is a non-symmetric measure of the difference between two probability distributions. It is here adapted for frequency spectra. The distance is asymmetric, i.e. computing the I-S distance between spec1 and spec2 is not the same as computing it between spec2 and spec1. A symmetry can be obtained by calculating the mean between the two directions. The distance is obtained following:

\[
D_{I-S}(spec1∥spec2) = \sum \frac{spec1}{spec2} - \log(\frac{spec1}{spec2}) - 1
\]

Value

The function returns a list of three items:

- `D1`: The I-S distance of 'spec2' with respect to 'spec1' (i.e. D(spec1 || spec2))
- `D2`: The I-S distance of 'spec1' with respect to 'spec2' (i.e. D(spec2 || spec1))
- `D`: The symmetric distance (i.e. D = 0.5*(D1+D2))

If `scale = TRUE` the distance is divided by the length of spec1 (or spec2).

Note

The function works for both Hz and (htk-)mel scales.
Author(s)
Jerome Sueur, improved by Laurent Lellouch

See Also
kl.dist, ks.dist, logspec.dist, simspec, diffspec

Examples
# Comparison of two spectra
data(tico)
tico1 <- spec(tico, at=0.65, plot=FALSE)
tico2 <- spec(tico, at=.11, plot=FALSE)
itakura.dist(tico1, tico2)
itakura.dist(tico1, tico2, scale=TRUE)

kl.dist  Kullback-Leibler distance

Description
Compare two distributions (e.g. two frequency spectra) by computing the Kullback-Leibler distance

Usage
kl.dist(spec1, spec2, base = 2)

Arguments
spec1 any distribution, especially a spectrum obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

spec2 any distribution, especially a spectrum obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

base the logarithm base used to compute the distance. See log.

Details
The Kullback-Leibler distance or relative entropy is a non-symmetric measure of the difference between two probability distributions. It is here adapted for frequency spectra. The distance is asymmetric, i.e. computing the K-L distance between spec1 and spec2 is not the same as computing it between spec2 and spec1. A symmetry can be obtained by calculating the mean between the two directions. The distance is obtained following:

\[ D_{K-L}(spec1||spec2) = \sum spec1 \times \log\left(\frac{spec1}{spec2}\right) \]
Value

The function returns a list of three items:

- \( D_1 \) The K-L distance of 'spec2' with respect to 'spec1' \( (i.e. \ D(spec1 \| spec2)) \)
- \( D_2 \) The K-L distance of 'spec1' with respect to 'spec2' \( (i.e. \ D(spec2 \| spec1)) \)
- \( D \) The symmetric K-L distance \( (i.e. \ D = 0.5*(D1+D2)) \)

Note

The base of the logarithm can be changed using the argument \( \text{base} \). When set to base 2, the information is measured in units of bits. When set to base \( e \), the information is measured in nats. The function works for both Hz and (htk-)mel scales.

Author(s)

Jerome Sueur, improved by Laurent Lellouch

References


See Also

*ks.dist*, *logspec.dist*, *simspec*, *difspect*

Examples

```r
# Comparison of two spectra
data(tico)
tico1 <- spec(tico, at=0.65, plot=FALSE)
tico2 <- spec(tico, at=1.1, plot=FALSE)
kl.dist(tico1, tico2)  # log2 (binary logarithm)
kldist(tico1, tico2, base=exp(1))  # ln (natural logarithm)
```

Description

This function compares two distributions (e.g. two frequency spectra) by computing the Kolmogorov-Smirnov distance

\[ KS.DIST(Spec1, Spec2) = D(Spec1 || Spec2) = D(Spec2 || Spec1) \]

\[ KS.DIST = 0.5 \times (D(Spec1 || Spec2) + D(Spec2 || Spec1)) \]

The base of the logarithm can be changed using the argument \( \text{base} \). When set to base 2, the information is measured in units of bits. When set to base \( e \), the information is measured in nats.
ks.dist

Usage
ks.dist(spec1, spec2, f = NULL, mel = FALSE,
plot = FALSE, type = "l",
lt = c(1, 2), col = c(2, 4),
flab = NULL, alab = "Cumulated amplitude",
flim = NULL, alim = NULL,
title = TRUE, legend = TRUE, ...)

Arguments

spec1 any distribution, especially a spectrum obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

spec2 any distribution, especially a spectrum obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

f sampling frequency of waves used to obtain spec1 and spec2 (in Hz). Not necessary if spec1 and/or spec2 is a two columns matrix obtained with spec or meanspec.

mel a logical, if TRUE the (htk-)mel scale is used.

plot logical, if TRUE plots both cumulated spectra and their maximal distance (i.e. the K-S distance.)

type if plot is TRUE, type of plot that should be drawn. See plot for details (by default "l" for lines).

lt a vector of length 2 for the line type of spec1 and spec2 if type="l".

col a vector of length 2 for the colour of spec1 and spec2.

flab title of the frequency axis.
alab title of the amplitude axis.

flim the range of frequency values.
alim range of amplitude axis.
title logical, if TRUE, adds a title with D and F values.

legend logical, if TRUE adds a legend to the plot.

... other plot graphical parameters.

Details
The Kolmogorov distance is the maximal distance between the cumulated spectra. The function returns this distance and the corresponding frequency. This is an adaptation of the statistic computed by the non-parametric Kolmogorov-Smirnov test (see ks.test).

Value
The function returns a list of two items

D the Kolomogorov-Smirnov distance

F the frequency (in KHz) where the Kolmogorov-Smirnov distance was found
Note

There is no p-value associated to the K-S distance.
If no frequency is provided, only the distance D.

Author(s)
Jerome Sueur, improved by Laurent Lellouch

See Also

kl.dist, simspec, diffspec, logspec.dist, diffcumspec, itakura.dist

Examples

## Comparison of two spectra and plot of the cumulated spectra with the K-S distance
data(tico)
tico1 <- spec(tico, at=0.65, plot=FALSE)
tico2 <- spec(tico, at=1.1, plot=FALSE)
ks.dist(tico1, tico2, plot=TRUE)
#### mel scale
require(tuneR)
data(orni)
orni.mel <- melfcc(orni, nbands = 256, dcttype = "t3", fbtype = "htkmel", spec_out=TRUE)
orni.mel.mean <- apply(orni.mel$aspectrum, MARGIN=2, FUN=mean)
tico.mel <- melfcc(tico, nbands = 256, dcttype = "t3", fbtype = "htkmel", spec_out=TRUE)
tico.mel.mean <- apply(tico.mel$aspectrum, MARGIN=2, FUN=mean)
ks.dist(orni.mel.mean, tico.mel.mean, f=22050, mel=TRUE, plot=TRUE)

\section*{lfs}

\textit{Linear Frequency Shift}

Description

This function linearly shifts all the frequency content of a time wave.

Usage

lfs(wave, f, shift, wl = 1024, ovlp = 75, wn = "hanning", fftw = FALSE, output = "matrix")

Arguments

\begin{itemize}
  \item wave: an R object.
  \item f: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
  \item shift: positive or negative frequency shift to apply (in Hz).
  \item wl: window length for the analysis (even number of points, by default = 1024).
  \item ovlp: overlap between successive FFT windows (in %, by default 75%).
\end{itemize}
window name, see `ftwindow` (by default "hanning").

if TRUE calls the function FFT of the library `fftw`. See Notes of the `spectro`.

character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".

Details

A short-term Fourier transform is first applied to the signal (see `spectro`), then the frequency shift is applied and the new signal is eventually generated using the reverse of the Fourier Transform (`istft`).

There is therefore neither temporal modifications nor amplitude modifications.

Value

If `plot` is `FALSE`, a new wave is returned. The class of the returned object is set with the argument `output`.

Author(s)

Jerome Sueur <sueur@mnhn.fr> and Thierry Aubin <thierry.aubin@u-psud.fr>

References


See Also

`fffilter`, `spectro`

Examples

data(orni)
a<-lfs(orni,f=22050,shift=1000)
spectro(a,f=22050)
  # to be compared with the original signal
spectro(orni,f=22050)

---

**listen**

*Play a sound wave*

---

Description

Play a sound wave

Usage

`listen(wave, f, from = NULL, to = NULL, choose = FALSE)`
Arguments

wave     an R object.
f          sampling frequency of wave (in Hz). Does not need to be specified in wave.
from      start of play (in s).
to        end of play (in s).
choose   logical, if TRUE start (=from) and end (=to) points can be graphically chosen with a cursor on the oscillogram.

Note

This function is based on `play` but allows to read one-column matrix, data.frame, time-series and Sample objects.

Author(s)

Jerome Sueur <sueur@mnhn.fr> but the original `play` function is by Uwe Ligges (package `tuneR`).

See Also

`play`

Examples

```r
## NOT RUN
# data(tico)
# listen(tico,f=22050)
# listen(tico,f=22050,from=0.5,to=1.5)
# listen(noise(d=1,f=8000,Wave=TRUE))
## change f to play the sound a different speed
# data(sheep)
## normal
# listen(sheep,f=8000)
## two times faster
# listen(sheep,f=8000*2)
## two times slower
# listen(sheep,f=8000/2)
```

---

**localpeaks**

*Local maximum frequency peak detection*

Description

This function searches for local peaks of a frequency spectrum.
### Usage

```r
localpeaks(spec, f = NULL, bands = 10, mel = FALSE, plot = TRUE,
            xlab = NULL, ylab = "Amplitude", labels = TRUE, ...)
```

### Arguments

- `spec`: a data set resulting of a spectral analysis obtained with `spec` or `meanspec`. Can be in dB.
- `f`: sampling frequency of `spec` (in Hz). Not requested if the first column of `spec` contains the frequency axis.
- `bands`: a numeric vector. If vector of length 1, then sets the number of bands dividing in equal parts the spectrum. If of length > 1, then takes the values as kHz limits of the bands dividing the spectrum. These bands can be of different size. See details and examples.
- `mel`: a logical, if TRUE the (htk-)-mel scale is used.
- `plot`: logical, if TRUE, a plot showing the peaks is returned.
- `xlab`: label of the x-axis.
- `ylab`: label of the y-axis.
- `labels`: logical, if TRUE peak labels are plotted.
- `...`: other `plot` graphical parameters.

### Details

The function proceed as follows

- divides the spectrum in bands. The limits of the bands are set with the argument `bands`. There are two options:
  - you set a number of bands with equal size by giving a single value to `bands`. For instance, setting `bands` to a value of 10 will slice the spectrum in 10 equal parts and return 10 local peaks.
  - you set the limits of the bands. This is achieve by giving a numeric vector to `bands`. The limits can follow a regular or irregular series. For instance attributing the vector `c(0,2,4,8)` will generate the following bands `[0,2[, [2,4[, [4,8] kHz. Be aware that the last value should not exceed half the sampling frequency used to obtain the spectrum `spec`.
- uses the function `fpeaks` with the argument `nmax` set to 1.

### Value

A two-column matrix, the first column corresponding to the frequency values (x-axis) and the second column corresponding to the amplitude values (y-axis) of the peaks.

### Author(s)

Jerome Sueur
See Also

fpeaks, meanspec, spec

Examples

data(sheep)
spec <- meanspec(sheep, f=8000)
# a specific number of bands with all the same size
localpeaks(spec, bands=5)
# bands directly specified with a regular sequence
localpeaks(spec, bands=seq(0,8/2,by=0.5))
# bands directly specified with an irregular sequence
localpeaks(spec, bands=c(0,0.5,1,1.5,3,4))
# Amaj octave bands, note that there is no peak detection
# in the higher part of the spectrum as sequence stops at 3520 Hz
localpeaks(spec, bands=octaves(440, below=3, above=3)/1000)
# melscale
require(tuneR)
mel <- melfcc(sheep, nbands = 256, dcttype = "t3", fbtype = "htkmel", spec_out=TRUE)
mlspec.mean <- apply(mel$aspectrum, MARGIN=2, FUN=mean)
mlspec.mean <- melspec.mean/max(melspec.mean) # [0,1] scaling
localpeaks(melspec.mean, f=8000, bands=8)

logspec.dist

Log-spectral distance

Description

Compare two distributions (e.g. two frequency spectra) by computing the log-spectral distance

Usage

logspec.dist(spec1, spec2, scale=FALSE)

Arguments

spec1 any distribution, especially a spectrum obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

spec2 any distribution, especially a spectrum obtained with spec or meanspec (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).

scale a logical, if TRUE the distance is scaled by dividing by the square-root of the length of spec1 (or spec2).
Details

The distance is computed according to:

\[ D_{LS}(\text{spec1}\|\text{spec2}) = D_{LS}(\text{spec2}\|\text{spec1}) = \sqrt{\sum_{i=1}^{10} 10 \times \log_{10}\left(\frac{\text{spec1}_{i}}{\text{spec2}_{i}}\right)^2} \]

If scale = TRUE the distance is divided by the length of spec1 (or spec2).

Value

A numeric vector of length 1 returning the D distance.

Note

The function works for both Hz and (htk-)mel scales.

Note

The distance is symmetric.

Author(s)

Jerome Sueur, improved by Laurent Lellouch

See Also

ks.dist, kl.dist, itakura.dist, simspec, diffspec

Examples

\[
\begin{align*}
\text{# Comparison of two spectra} \\
data(tico) \\
tico1 &\leftarrow \text{spec}(tico, \text{at}=0.65, \text{plot}=\text{FALSE}) \\
tico2 &\leftarrow \text{spec}(tico, \text{at}=1.1, \text{plot}=\text{FALSE}) \\
\text{logspec.dist}(tico1, tico2) \\
\text{logspec.dist}(tico1, tico2, \text{scale}=\text{TRUE}) \\
\end{align*}
\]

\[
\text{M} \\
\text{Median of the amplitude envelope}
\]

Description

This function computes an acoustic index based on the median of the amplitude envelope.

Usage

\[
\text{M(wave, f, envt = "hil", plot = FALSE, ...)}
\]
Arguments

- wave: an R object.
- f: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
- envt: the type of envelope to be used: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope. See env.
- plot: logical, if TRUE returns a plot of the amplitude envelope of wave (by default FALSE).
- other env parameters, in particular smoothing parameters. See env.

Details

This amplitude index $M$ is computed according to:

$$M = \bar{A}(t) \times 2^{1-depth}$$

with

$$0 \leq M \leq 1$$

where $A(t)$ is the amplitude envelope and depth is the signal digitization depth in number of bits.

Value

A numeric vector of length 1 between 0 and 1, without unit.

Author(s)

Jerome Sueur and Marion Depraetere

References


See Also

env, AR

Examples

data(tico)
M(tico)
# smoothing the amplitude may change slightly the result
M(tico, msmooth=c(500,50), plot=TRUE)
**meandB**

*Mean of dB values*

**Description**

This function calculates the mean of dB values.

**Usage**

```r
meandB(x, level="IL")
```

**Arguments**

- `x`: a numeric vector or a numeric matrix.
- `level`: intensity level ("IL") or sound pressure level ("SPL")

**Details**

The mean of dB values is not linear. See examples.

**Value**

A numeric vector of length 1 is returned.

**Author(s)**

Jerome Sueur and Zev Ross

**References**


**See Also**

`sddb, moredB, convSPL, dBweight`

**Examples**

```r
meandB(c(89, 90, 95))
```
Description

This function returns the mean frequency spectrum (i.e. the mean relative amplitude of the frequency distribution) of a time wave. Results can be expressed either in absolute or dB data.

Usage

```r
meanzspec(wave, f, wl = 512, wn = "hanning", ovlp = 0, fftw = FALSE, 
norm = TRUE, PSD = FALSE, PMF = FALSE, FUN = mean, correction = "none", dB = NULL, 
dBref = NULL, from = NULL, to = NULL, identify = FALSE, 
col = "black", cex = 1, plot = 1, flab = "Frequency (kHz)", 
alab = "Amplitude", flim = NULL, alim = NULL, type = "l", ...)
```

Arguments

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl`: length of the window for the analysis (even number of points, by default = 512).
- `wn`: window name, see `ftwindow` (by default "hanning").
- `ovlp`: overlap between two successive analysis windows (in %).
- `fftw`: if `TRUE` calls the function FFT of the library `fftw`. See Notes of `spectro`.
- `norm`: if `TRUE` the mean spectrum is normalised (i.e. scaled) by its maximum.
- `PSD`: if `TRUE` return Power Spectra Density, i.e. the square of the spectra.
- `PMF`: if `TRUE` return Probability Mass Function, i.e. the probability distribution of frequency.
- `FUN`: The function to apply on the rows of the STFT matrix, by default `mean` for a mean spectrum but could be other as `median` or `var` for a median spectrum or variance spectrum.
- `correction`: a character vector of length 1 to apply an amplitude ("amplitude") or an energy ("energy") correction to the FT window. This argument is useful only when one wish to obtain absolute values that is when `norm=FALSE` and `PMF=FALSE`. By default no correction is applied ("none").
- `dB`: a character string specifying the type dB to return: "max0" for a maximum dB value at 0, "A", "B", "C" and "D" for common dB weights.
- `dBref`: a dB reference value when dB is not NULL. NULL by default but should be set to 2*10e-5 for a 20 microPa reference (SPL).
- `from`: start mark where to compute the spectrum (in s).
- `to`: end mark where to compute the spectrum (in s).
**meanspec**

identify to identify frequency and amplitude values on the plot with the help of a cursor.
col colour of the spectrum.
cex pitch size.
plot if 1 returns frequency on x-axis, if 2 returns frequency on y-axis, (by default 1).
flab title of the frequency axis.
alab title of the amplitude axis.
flim range of frequency axis (in kHz).
alim range of amplitude axis.
type if plot is TRUE, type of plot that should be drawn. See plot for details (by default "l" for lines).
... other plot graphical parameters.

**Details**

See examples of spec. This function is based on fft.

**Value**

If plot is FALSE, meanspec returns a two columns matrix, the first column corresponding to the frequency axis, the second column corresponding to the amplitude axis.
If identify is TRUE, spec returns a list with two elements:

freq the frequency of the points chosen on the spectrum

amp the relative amplitude of the points chosen on the spectrum

**Warning**

The argument peaks is no more available (version > 1.5.6). See the function fpeaks for peak(s) detection.

**Note**

The argument fftw can be used to try to speed up process time. When set to TRUE, the Fourier transform is computed through the function FFT of the package fftw. This package is a wrapper around the fastest Fourier transform of the free C subroutine library FFTW (http://www.fftw.org/). FFT should be then installed on your OS.

**Author(s)**

Jerome Sueur <sueur@mnhn.fr>

**See Also**

spec, fpeaks, localpeaks, dynspec, corspec, diffspec, simspec, fft.
Examples

data(orni)
# compute the mean spectrum of the whole time wave
meanspec(orni,f=22050)
# compute the mean spectrum of a time wave section (from 0.32 s to 0.39 s)
meanspec(orni,f=22050,from=0.32,to=0.39)
# different window lengths
op<-par(mfrow=c(3,1))
meanspec(orni,f=22050,wl=256)
title("wl=256")
meanspec(orni,f=22050,wl=1024)
title("wl=1024")
meanspec(orni,f=22050,wl=4096)
title("wl=4096")
par(op)
# different overlap values (almost no effects here...)
op<-par(mfrow=c(3,1))
meanspec(orni,f=22050)
title("ovlp=0")
meanspec(orni,f=22050,ovlp=50)
title("ovlp=50")
meanspec(orni,f=22050,ovlp=95)
title("ovlp=95")
par(op)
# use of flim to zoom in
op<-par(mfrow=c(2,1))
meanspec(orni,f=22050)
title("zoom in")
meanspec(orni,f=22050,wl=512,flim=c(4,6))
par(op)
# comparaison of spectrum and mean spectrum
op<-par(mfrow=c(2,1))
spec(orni,f=22050)
title("spec()")
meanspec(orni,f=22050)
title("meanspec()")
par(op)
# log scale on frequency axis
meanspec(orni, f=22050, log="x")
# median spectrum
meanspec(orni,f=22050, FUN=median)
# variance spectrum
meanspec(orni,f=22050, FUN=var)

---

mel

**Hertz / Mel conversion**

Description

This function converts Hertz data in Mel data.
Usage

```
mel(x, inverse = FALSE)
```

Arguments

- **x**: a value in Hertz (or in Mel if `inverse` is TRUE)
- **inverse**: logical, if TRUE converts the Mel data in Hertz data.

Details

Hertz to Mel conversion is computed according to:

\[
m = 1127.01048 \times \log (1 + \left(\frac{f}{700}\right))
\]

with \(m\) in Mel and \(f\) in Hertz.

Mel to Hertz conversion (when `inverse` is TRUE) is therefore computed according to:

\[
f = 700 \times \left( e^{-\frac{m}{1127.01048}} - 1 \right)
\]

with \(f\) in Hertz and \(m\) in Mel.

Value

A corresponding R object is returned.

Note

The Mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another. The name Mel comes from the word melody to indicate that the scale is based on pitch comparisons. The reference point between this scale and normal frequency measurement is defined by equating a 1000 Hz tone, 40 dB above the listener’s threshold, with a pitch of 1000 mels.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

References


See Also

- `melfilterbank`

Examples

```r
x <- seq(0, 10000, by=50)
y <- mel(x)
plot(x, y, type="l", xlab = "f (hertz)", ylab = "f (mel)",
     main = "Mel scale", col="red")
```
melfilterbank  

Mel-filter bank for MFCC computation

Description

This function returns graphically and numerically the Mel-filters used to compute MFCC.

Usage

melfilterbank(f = 44100, wl = 1024,
minfreq = 0, maxfreq = f/2, m = 20,
palette, alpha = 0.5, plot = FALSE)

Arguments

f  
the Fourier window length (in number of samples).
wl  
the minimum (or lower) frequency of the filter bank (in Hz).
minfreq  
the maximum (or upper) frequency of the filter bank (in Hz).
maxfreq  
the total number of filters.
m  
an optional colour palette if plot is TRUE.
palette  
alpha-transparency when a colour palette is used.
alpha  
if TRUE all filters are displayed in a single plot.
plot

Value

A list of 3 items:

central.freq  
the kHz central frequencies of the filters,
freq  
the kHz frequency scale,
amp  
the amplitude of the filters, scaled between 0 and 1.

Note

These triangular filters are used for computing MFCCs.

Author(s)

Jerome Sueur

References

**micsens**

**Microphone sensitivity and conversion**

**Description**

This function converts microphone sensitivity from mV/Pa to dB.

**Usage**

```r
micsens(x, sref = 1, inverse = FALSE)
```

**Arguments**

- `x`: a measured sensitivity in mV/Pa (or in dB if `inverse` is TRUE)
- `sref`: the sensitivity reference (by default equals to 1 V/Pa)
- `inverse`: logical, if TRUE, the inverse conversion from dB to mV/Pa is computed.

**Details**

The sensitivity $S$ in dB is calculated according to:

$$ S_{dB} = 20 \times \log_{10} \left( \frac{s}{s_{ref}} \right) $$

with $s$ the measured sensitivity in mv/Pa and $s_{ref}$ the reference sensitivity (by default 1 mV/Pa).

**Value**

A numeric value in dB re 1V/Pa with default settings, in mV/Pa if `inverse` is set to FALSE.
Description
This function calculates the sum of dB values.

Usage
moredB(x, level="IL")

Arguments
x  a numeric vector or numeric matrix.
level  intensity level ("IL") or sound pressure level ("SPL")

Details
The addition of dB values is not linear. See examples.

Value
A numeric vector of length 1.

Author(s)
Jerome Sueur

References

See Also
meandB, sddB, convSPL, dBweight
Examples

```r
# two sources of 60 dB give an intensity level of 63 dB
moredB(c(60,60))
# addition of three sources
moredB(c(89,90,95))
```

**mutew**  
*Replace time wave data by 0 values*

### Description

This function replaces a time wave or a section of a time wave by 0 values. For a time wave describing a sound, this corresponds to muting the sound or a section of it.

### Usage

```r
mutew(wave, f, from = NULL, to = NULL, choose = FALSE, plot = TRUE,
output = "matrix", ...)
```

### Arguments

- **wave**: an R object.
- **f**: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- **from**: start of the silence section (in s).
- **to**: end of the silence section (in s).
- **choose**: logical, if `TRUE` start (= `from`) and end (= `to`) points can be graphically chosen with a cursor on the oscillogram.
- **plot**: logical, if `TRUE` returns an oscillographic plot of `wave` with the new silence section (by default `TRUE`).
- **output**: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
- **...**: other oscillo graphical parameters.

### Details

By default, `from` and `to` are `NULL`, this results in completely muting `wave`.

### Value

If `plot` is `FALSE`, a new wave is returned. The class of the returned object is set with the argument `output`.

### Author(s)

Jerome Sueur <sueur@mnhn.fr>
See Also

oscillo, addsilw, cutw, deletew, fadew, pastew, revw, zapsilw

Examples

data(tico)
mutew(tico,f=22050,from=0.5,to=0.9)

---

NDSI

Normalized Difference Soundscape Index

Description

This function computes the Normalized Difference Soundscape Index as described by Kasten et al. (2012).

Usage

NDSI(x, anthropophony = 1, biophony = 2:8)

Arguments

- **x**: a two-column numeric matrix computed with `soundscapespec`.
- **anthropophony**: a numeric vector defining the frequency band(s) of the anthropophony (in kHz).
- **biophony**: a numeric vector defining the frequency band(s) of the biophony (in kHz).

Details

NDSI aims at estimating the level of anthropogenic disturbance on the soundscape by computing the ratio of human-generated (anthropophony) to biological (biophony) acoustic components found in field collected sound samples. In terms of frequency, the anthropophony is defined as the [1-2] kHz frequency bin and the biophony as the [2-8] kHz frequency bins of a soundscape frequency spectrum (see `soundscapespec`).

NDSI is computed according to:

\[
NDSI = \frac{\text{biophony} - \text{anthropophony}}{\text{biophony} + \text{anthropophony}}
\]

NDSI varies between -1 and +1, where +1 indicates a signal containing no anthropophony.

Value

A numeric vector of length 1 giving the NDSI value.
noisew

Author(s)
Jerome Sueur

References

See Also
soundscapespec, SAX, NDSI

Examples
```r
## Note that 'tico' is not a soundscape recording...
data(tico)
spec <- soundscapespec(tico, plot=FALSE)
NDSI(spec)
```

noisew Generate noise

Description
This function generates noise.

Usage
```r
noisew(f, d, type="unif", listen = FALSE, output = "matrix")
```

Arguments
- `f`: sampling frequency of the signal to be generated (in Hz)
- `d`: duration of the signal to be generated.
- `type`: a character string to specify the type of noise, either "unif" or "gaussian".
- `listen`: if TRUE the new sound is played back.
- `output`: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".

Details
Uniform noise is generated using `runif` and gaussian noise is based on `rnorm`
Value

A new wave is returned. The class of the returned object is set with the argument output.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

synth, pulse

Examples

```r
# add noise to a synthetic signal
a <- noise(d=1, f=8000)
b <- synth(f=8000, d=1, cf=2000, plot=FALSE)
c <- a + b
spectro(c, f=8000)
```

notefreq

**Frequency of a musical note**

Description

This function computes the frequency of a musical note (Equal temperament)

Usage

```r
notefreq(note, ref = 440, octave = 3)
```

Arguments

- **note**: a numerical or a character vector. See Note.
- **ref**: a numerical vector of length 1 for the reference frequency.
- **octave**: a numerical vector of length for the octave number.

Details

The frequency is computed according to:

\[ f = ref \times 2^{\frac{octave-3+\frac{note-1}{12}}{12}} \]

with:
- **ref** = reference frequency,
- **octave** = octave number, and
- **note** = rank of the note along the scale.
Value

The frequency in Hz is returned.

Note

The note can be given in two ways. The first solution is to give the rank of the note along the scale (e.g. rank 10 for A) or to give its names in characters with the following notation: C, D, E, F, G, A, B.

Author(s)

Jerome Sueur

See Also

octaves

Examples

```r
# Some notes frequency (use apply-like functions when dealing with character strings)
sapply(c("C", "A", "Gb"), notefreq)

# C major scale plot
n <- 1:12
freq <- notefreq(n)
plot(n, freq, pch=19, cex=1.5,
   xlab = "Note name",
   ylab = "Frequency (Hz)",
   xaxt="n", las=1, main="Third octave")
axis(side=1, at=n, labels=names)
abline(h=freq, col="lightgrey")

# C major scale sound
f <- 2000 # sampling rate
s <- NULL
for (i in 1:length(freq))
  { tmp <- synth(d=0.5, f=f, cf=freq[i])
    s <- pastew(s, tmp, at="start", f)
  }
spectro(s, f, ovlp=75)
```

Description

This function returns the frequency values of the octaves below and above a specific frequency
Usage

octaves(x, below = 3, above = 3)

Arguments

x a numeric vector, frequency of the note in Hz or kHz.
below the number of octaves below x.
above the number of octaves above x.

Value

A numeric vector with the octave series in frequency (Hz or kHz depending on x unit).

Author(s)

Jerome Sueur

See Also

notefreq

Examples

names <- c("C", "D", "E", "F", "G", "A", "B")
values <- c(261.63, 293.66, 329.64, 349.23, 392, 440, 493.88)
res <- sapply(values, FUN=octaves)/1000
op <- par(las=1,mfrow=c(2,1))
par(mar=c(0,4,1,1))
matplot(x=1:7, y=res, t="o", pch=names, xlab="",
ylab="Frequency (kHz) [linear scale]", col=rainbow(7), xaxt="n")
par(mar=c(4.5,4,0,1))
matplot(x=1:7, y=res, t="o", pch=names, xlab="Octave",
ylab="Frequency (kHz) [log scale]", col=rainbow(7), ylog=TRUE, log="y")
par(op)

---

orni Song of the cicada Cicada orni

Description

Recording of a calling song section of the Mediterranean cicada Cicada orni.

Usage

data(orni)
oscillo

Format

A Wave object.

Details

Duration = 0.719 s. Sampling frequency = 22050 Hz.

Source

Recording by Jerome Sueur.

Examples

data(orni)
oscillo(orni,f=22050)

---

oscillo  

Show a time wave as an oscillogram

Description

This graphical function displays a time wave as an oscillogram in a single or multi-frame plot. The envelope of the wave can also be shown.

Usage

oscillo(wave, f, from = NULL, to = NULL, scroll = NULL, zoom = FALSE, k=1, j=1, cex, labels = TRUE, tlab = "Time (s)", alab = "Amplitude", byrow = TRUE, identify = FALSE, nidentify = NULL, plot = TRUE, colwave = "black", coltitle = "black", cextitle = 1.2, fonttitle = 2, collab = "black", cexlab = 1, fontlab = 1, colline = "black", colaxis = "black", cexaxis = 1, fontaxis = 1, coly = "lightgrey", tcl = 0.5, title = FALSE, xaxt="s", yaxt="n", type="l", bty = "l")

Arguments

wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
from start of the oscillogram (in s).
to end of the oscillogram (in s).
scroll  a numeric of length 1 allowing to move along the time wave using a slider panel. This numeric corresponds to the number of successive windows dividing the time wave.

zoom  time zoom in with start and end points chosen on the oscillogram with a cursor.

k  number of horizontal sections (by default =1).

j  number of vertical sections (by default =1).

cex  Pitch size if type = "p".

labels  if TRUE plots time and amplitude labels (by default TRUE).

tlab  Label of time axis.

alab  Label of amplitude axis.

byrow  logical, if TRUE, the sections are filled by rows, otherwise the sections are filled by colmuns (by default TRUE).

identify  returns the time and amplitude coordinates of points chosen with a cursor on the oscillogram.

nidetify  a numeric vector of length 1, specifies the number of points to identified on wave if identify is TRUE.

plot  logical, if TRUE returns an oscillographic or envelope plot of wave (by default TRUE).

colwave  colour of the oscillogram or of the envelope.

coltitle  if title is TRUE, colour of the title.

cextitle  character size for the title.

fonttitle  font for the title.

cexlab  character size for axes labels.

fontlab  font for axes labels.

collab  colour of axes labels.

colline  colour of axes line.

colaxis  colour of the axis annotation.

fontaxis  font of axis annotation.

cexaxis  magnification for axis annotation.

coly0  colour of the y=0 line.

tcl  length of tick marks.

title  TRUE to add a title with information on wave duration and f, FALSE to live it blank, or a character string to add any desired title.

xaxt  equivalent to xaxt of par (by default ="s").

yaxt  equivalent to yaxt of par (by default ="n").

type  type of plot, by default "1". Use "n" for no plot.

bty  the type of box to be drawn around the oscillogram.
Value

Data are returned as one-column matrix if plot is FALSE. identify returns a two-column matrix with the time and amplitude coordinates of points successively chosen on the oscillogram.

Note

zoom is similar to but more visual than from and/or to. zoom and identify do work with a single-frame window only (i.e. with \( k = 1 \) and \( j = 1 \)).
Press ‘Stop’ button of the tools bar after choosing the appropriate points on the oscillogram.

Author(s)

Jerome Sueur <sueur@mnhn.fr> and Caroline Simonis <csimonis@mnhn.fr>.

See Also

dynoscillo, oscilloST, cutw, pastew, timer

Examples

data(tico)
# a simple oscillogram of a bird song
oscillo(tico)
# zoom in
op<-par(mfrow=c(4,1),mar=c(4.5,4,2,2))
oscillo(tico,22050,cexlab=0.75)
oscillo(tico,22050,from=0.5,to=0.9,cexlab=0.75)
oscillo(tico,22050,from=0.65,to=0.75,cexlab=0.75)
oscillo(tico,22050,from=0.68,to=0.70,cexlab=0.75)
par(op)
# the same divided in four lines
oscillo(tico,f=22050,k=4,j=1)
# the same divided in different numbers of lines and columns
oscillo(tico,f=22050,k=4,j=4)
oscillo(tico,f=22050,k=2,j=2,byrow=TRUE)
oscillo(tico,f=22050,k=2,j=2,byrow=FALSE)
# overplot of oscillographic and envelope representations
oscillo(tico,f=22050)
par(new=TRUE)
env(tico,f=22050,colwave=2)
# full colour modifications in a two-frame oscillogram
op<-par(bg="grey")
oscillo(tico,f=22050,k=4,j=1,title=TRUE,colwave="black",
coltitle="yellow",colab="red",colline="white",
colaxis="blue",coly0="grey50")
par(op)
# change the title
data(orni)
oscillo(orni,f=22050,title="The song of a famous cicada")
# move along the signal using scroll
oscilloST

Show a stereo time wave as oscillograms

Description

This graphical function displays a stereo (2 channels) time wave as an oscillogram in a two-frame plot. The envelope of the wave can also be shown.

Usage

oscilloST(wave1, wave2 = NULL, f, from = NULL, to = NULL,
identify = FALSE, plot = TRUE, colwave1 = "black",
colwave2 = "blue", coltitle = "black",
collab = "black", cexlab = 1, fontlab = 1, colaxis = "black",
cexaxis = 1, coly01 = "grey47", coly02 = "black", title = FALSE,
bty = "l")

Arguments

wave1 a first R object.
wave2 a second R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
from start of the oscillogram (in s).
to end of the oscillogram (in s).
identify returns the time coordinate of points chosen with a cursor on the bottom oscillogram.
plot logical, if TRUE returns an oscillographic or envelope plot of wave(by default TRUE).
colwave1 colour of the oscillogram or of the envelope of wave1.
colwave2 colour of the oscillogram or of the envelope of wave2.
coltitle if title is TRUE, colour of the title.
collab colour of axes title.
cexlab character size for axes title.
fontlab font for axes title.
colaxis colour of the axes
cexaxis magnification for axes annotation.
coly01 colour of the y=0 line of wave1.
coly02 colour of the y=0 line of wave1.
title logical, if TRUE plots the title with information on time and f (by default FALSE).
bty the type of box to be drawn around the oscillogram.
Value

Data are returned as two-column matrix if plot is FALSE. identify returns a numeric object with the time coordinate of points successively chosen on the bottom oscillogram.

Author(s)

Jerome Sueur and Caroline Simonis.

See Also

oscillo, dynoscillo

Examples

```r
a <- synth(f=8000,d=1,cf=2000,am=c(50,10),plot=FALSE)
b <- synth(f=8000,d=1,cf=1000,fm=c(0,0,2000,0,0),plot=FALSE)
oscillos(a, b, f=8000)
```

Description

This function pastes a first time wave to a second one. The time wave to be pasted, the time wave to be completed and the resulting time wave can be displayed in a three-frame oscillographic plot.

Usage

```r
pastew(wave1, wave2, f, at = "end",
join = FALSE, tjunction=0,
choose = FALSE, plot = FALSE,
marks = TRUE, output = "matrix", ...)
```

Arguments

- `wave1`: a first R object.
- `wave2`: a second R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `at`: `wave2` position in seconds where `wave1` will be pasted into. Can be also specified as "start", "middle" or "end".
- `join`: if TRUE the two waves will be pasted and jointed by removing the last point of `wave2`. See examples.
- `tjunction`: a numeric vector to remove clicks at the junction of ‘wave1’ and ‘wave2’. The value specifies the duration in seconds where the real values will be replaced by a linear interpolation. This duration should be a few milliseconds.
choose logical, if TRUE the point where wave1 will be pasted into wave2 (=at) can be graphically chosen with a cursor.

plot logical, if TRUE returns an oscillographic plot of wave1, wave2 and wave1 + wave2 (by default FALSE).

marks logical, if TRUE shows where wave1 has been pasted (by default TRUE).

output character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".

... other oscillo graphical parameters.

Details

If plot is TRUE returns a two-frame plot with three waves:
(1) the wave to be pasted (wave1),
(2) the wave to be completed (wave2),
(3) the resulting wave.

Value

If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Author(s)

Jerome Sueur, improved by Laurent Lellouch

See Also

oscillo, addsilw, cutw, deletew, fadew, mutew, revw, repw, zapsilw

Examples

data(tico)
# double a data set describing a bird song
a<-pastew(tico,tico,f=22050)
oscillo(a,f=22050)
# a direct way to see what has been pasted
pastew(tico,tico,f=22050,plot=TRUE)
# cut a section and then paste it at the beginning
a<-cutw(tico, f=22050, from=0.5, to=0.9)
pastew(a,tico,f=22050,at="start",plot=TRUE)
# or paste it at a specific location
pastew(a,tico,f=22050,at=1.4,plot=TRUE)
# setting the argument 'join' to TRUE might be useful
# to smooth pasting when some phase problem occur
# generate two sine waves
a <- synth(cf=50, f=400, d=0.1)
b <- synth(cf=100, f=400, d=0.1)
# paste it with 'join' turned to FALSE
# there is a click at the junction between the two waves
pastew(a, b, f=400, plot=TRUE)
peewit

# that can be removed by setting 'join' to TRUE
pastew(a, b, f=400, join=TRUE, plot=TRUE)
# or by using the argument 'tjunction'
pastew(a, b, f=400, tjunction=0.01, plot=TRUE)

peewit  

Song of the bird *Vanellus vanellus*

Description

Recording of a song emitted by a peewit (lapwing) male *Vanellus vanellus*

Usage

data(peewit)

Format

A Wave object.

Details

Duration = 0.706 s. Sampling frequency = 22050 hz.

Source

Recording by Thierry Aubin.

Examples

data(peewit)
oscillo(peewit, f=22050)

pellucens

Calling song of the tree cricket *Oecanthus pellucens*

Description

Recording of a calling song section emitted by the European tree cricket *Oecanthus pellucens*.

Usage

data(pellucens)

Format

A Wave object.
phaseplot

Details

Duration = 3.309 s. Sampling frequency = 11025 hz.

Source

Recording by Jerome Sueur.

Examples

data(pellucens)
oscillo(pellucens,f=11025)

phaseplot Phase-phase 2D or 3D plot of a time wave

Description

This function returns a 2D or 3D representation of a time wave according to its first, second and possibly third derivatives.

Usage

phaseplot(wave, f, dim = 3, plot = TRUE, type = "l", xlab = "1st derivative", ylab = "2nd derivative", zlab = "3rd derivative", ...)

Arguments

wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
dim a vector of length 1, the number of dimensions of the plot. Can be either 2 or 3.
plot logical, if TRUE plots phase-phase plot (by default TRUE).
type type of plot that should be drawn. See plot for details (by default "l" for lines).
xlab title of the x axis.
ylab title of the y axis.
zlab title of the z axis.
... other plot or plot3d graphical parameters of the package rgl.

Value

If plot is FALSE then a 2 or 3 column matrix is returned. The position of the column is related to the order of the derivative (i.e. first column = first derivative.)
**Note**

Phase-phase plot can be used to test non-linearity.

**Author(s)**

Jerome Sueur

**References**


**See Also**

`phaseplot2`

**Examples**

```r
## Not run:
require(rgl)
data(tico)
phaseplot(tico)

## End(Not run)
s <- synth(d=0.05, f=44100, cf=440, out="Wave")
n <- noise(d=0.05, f=44100, out="Wave")
par(mfrow=c(2,1))
phaseplot(s, dim=2)
phaseplot(n, dim=2)
```

---

**phaseplot2**  
*Phase-phase 2D plot of a time wave*

**Description**

This function returns a 2D representation of a time wave against a delayed version of itself.

**Usage**

```r
phaseplot2(wave, f, tau = 1, type = "1",
xlab = "x(t)",
ylab = paste("x(t+", tau, ")", sep = ",")
```
Arguments

- **wave**: an R object.
- **f**: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- **tau**: the time delay to apply in number of samples.
- **type**: type of plot that should be drawn. See `plot` for details (by default "l" for lines).
- **xlab**: title of the x axis.
- **ylab**: title of the y axis.
- **...**: other `plot` parameters.

Details

The principle consists in displaying in a single x-y graph the original time wave with a delayed version of itself. The delay is controlled with the argument `tau` that needs to be specified in number of samples. The conversion of `tau` in second is obtained by calculating `tau/f`, with `f` the sampling frequency.

Value

Nothing is returned except an x-y plot.

Note

Phase-phase plot can be used to test non-linearity.

Author(s)

Jerome Sueur

References


See Also

`phaseplot`

Examples

```r
s <- synth(d=0.05, f=44100, cf=440, out="Wave")
n <- noisew(d=0.05, f=44100, out="Wave")
par(mfrow=c(2,1))
phaseplot2(s)
phaseplot2(n)
```
playlist

Play a list of sound files

Description

This function works as a playlist, ie it plays back a list of sound files.

Usage

playlist(directory, sample = FALSE, loop = 1)

Arguments

directory  a character vector indicating the path to the directory where sound files to played are saved.

sample  a logical, if TRUE the order of sounds files to be played back is shuffled.

loop  a numeric vector of length 1, number of loops.

Details

The success of using this function depends on the wave player in use. This works particularly well with SoX under Linux. The type of files (.mp3, .wav, .ogg etc) depends on the wave player as well.

Value

None. Listen and enjoy!

Note

The function is mainly based on play

Author(s)

Jérôme Sueur

See Also

play, listen

Examples

## Not run:
playlist("MyMusic", sample = TRUE, loop=2)

## End(Not run)
Description

A pre-emphasis frequency filter for speech

Usage

preemphasis(wave, f, alpha = 0.9, plot = FALSE, output = "matrix", ...)

Arguments

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `alpha`: time constant, see Details.
- `plot`: a logical, if TRUE plots the spectrogram of the filtered `wave` and the frequency response of the comb filter.
- `output`: character string, the class of the object to return, either 'matrix', 'Wave', 'Sample', 'audioSample' or 'ts'.
- `...`: other arguments to be passed to `spectro` except `scale` and `osc` that are set by default to FALSE.

Details

The function applies a pre-emphasis filter usually applied in speech analysis. The filter is a kind of high-pass frequency filter that amplifies the high-frequency content of the sample. The filter is defined with:

\[ y(n) = x(n) - \alpha \times x(n - 1) \]

where alpha is a time constant usually set between 0.9 and 1.

The frequency response of the filter is obtained with:

\[ H(f) = 1 + a^2 - 2 \times \alpha \times \cos(2 \times \pi \times f / f_s) \]

Value

A new wave is returned. The class of the returned object is set with the argument `output`.

Author(s)

Jerome Sueur
See Also

bwfilter, combfilter, ffilter, fir, lfs, afilter

Examples

data(sheep)
f <- 150
f <- sheep$samp.rate
alpha <- exp(-2*pi*fc/f)
res <- preemphasis(sheep, alpha=alpha, output="Wave")

pulsew

Generate rectangle pulse

Description

This function generates a rectangle pulse.

Usage

pulsew(dbefore, dpulse, dafter, f, plot = FALSE, output = "matrix", ...)

Arguments

dbefore  duration of the silent period before the pulse
dpulse   duration of the pulse to generate
dafter  duration of silent period after the pulse
f        sampling frequency of the signal to be generated (in Hz)
plot     logical, if TRUE returns an oscillographic plot of the pulse generated (by default FALSE).
output  character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
...      other plot parameters.

Value

If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

synth, noisew
Examples

```r
pulsew(dbefore=0.5, dpulse=0.1, dafter=0.3, f=8000, plot=TRUE)
```

---

Q  
*Resonance quality factor of a frequency spectrum*

Description

This function estimates the frequency pureness of a time wave by returning the resonant quality factor Q at a specific dB level.

Usage

```r
Q(spec, f = NULL, level = -3, mel = FALSE, plot = TRUE, colval = "red",
  cexval = 1, fontval = 1, flab = NULL,
  alab = "Relative amplitude (dB)", type = "l", ...)
```

Arguments

- **spec** a data set resulting of a spectral analysis obtained with `spec`, or `meanspec` (in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).
- **f** sampling frequency of the wave used to obtain spec (in Hz). Not necessary if spec is a two columns matrix obtained with `spec` or `meanspec`.
- **level** frequency bandwidth set by an amplitude value relative to spectrum (in dB).
- **mel** a logical, if TRUE the (htk-)mel scale is used.
- **plot** logical, if TRUE returns the spectrum with Q plotted (by default TRUE).
- **colval** colour of plotting Q.
- **cexval** character size of plotting Q.
- **fontval** font of plotting Q.
- **flab** title of the frequency axis.
- **alab** title of the amplitude axis.
- **type** if plot is TRUE, type of plot that should be drawn. See `plot` for details (by default "l" for lines).
- **...** other `plot` graphical parameters.

Details

A high Q value indicates a highly resonant system.
Value

A list is returned with the following four items:

- **Q**: a numeric vector of length 1 returning the Q factor (no units)
- **dfreq**: a numeric vector of length 1 the dominant frequency (kHz)
- **fmin**: a numeric vector of length 1 returning the minimum frequency of the -dB level bandwidth (kHz)
- **fmax**: a numeric vector of length 1 returning the minimum frequency of the -dB level bandwidth (kHz)
- **bwd**: a numeric vector of length 1 returning the bandwidth, i.e. fmax - fmin (kHz)

Note

This function is based on an linear interpolation of the spectrum so that the result should be considered as an estimation, not an exact measure.

Author(s)

Jerome Sueur, improved by Laurent Lellouch

See Also

`spec`, `meanspec`, `corspec`, `fft`.

Examples

```r
# bird song
data(tico)
t<-spec(tico,f=22050,at=1.1,plot=FALSE,db="max0")
op<-par(mfrow=c(2,1),las=1)
Q(t,type="l")
Q(t,type="l",xlim=c(3.8,4.2),ylim=c(-60,0))
title("zoom in")
par(op)
# cricket, changing the dB level
data(pellucens)
p<-spec(pellucens,f=11025,at=0.5,plot=FALSE,db="max0")
op<-par(mfrow=c(3,1))
Q(p,type="l",xlim=c(1.8,2.6),ylim=c(-70,0))
title("level = - 3 (default value)",col.main="red")
Q(p,type="l",level=-6,
   xlim=c(1.8,2.6),ylim=c(-70,0),colval="blue")
title("level = - 6",col.main="blue")
Q(p,type="l",level=-9,
   xlim=c(1.8,2.6),ylim=c(-70,0),colval="green")
title("level = - 9",col.main="green")
par(op)
```
### Description

This function repeats a time wave.

### Usage

```r
repw(wave, f, times = 2, join = FALSE, plot = FALSE, output = "matrix", ...)
```

### Arguments

- **wave**: an R object.
- **f**: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- **times**: a numeric of length 1 describing the number of times the wave has to be repeated.
- **join**: if TRUE the last point of `wave` will be removed for smoothing junction between repetitions. See examples.
- **plot**: logical, if TRUE plots the repeated time wave.
- **output**: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
- **...**: other `oscillo` graphical parameters.

### Value

If `plot` is FALSE, a new wave is returned. The class of the returned object is set with the argument `output`.

### Author(s)

Jerome Sueur <sueur@mnhn.fr>

### See Also

`oscillo`, `addsilw`, `cutw`, `deletew`, `fadew`, `mutew`, `pastew`, `revw`, `zapsilw`

### Examples

```r
data(tico)
repw(tico,f=22050,plot=TRUE)
# use 'join' for smooth pasting
par(mfrow=c(2,1))
a <- synth(cf=50, f=400, d=0.1)
repw(a, f=400, plot=TRUE)
```
resamp

Describes
This function resamples (down- or over-samples) a time wave. This corresponds to a sampling frequency change.

Usage
resamp(wave, f, g, output="matrix")

Arguments
- wave: an R object.
- f: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
- g: new sampling frequency of wave (in Hz).
- output: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".

Value
If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Note
Resampling might change frequency properties of the time wave.

Author(s)
Jerome Sueur <sueur@mnhn.fr>

Examples
data(peewit)
# downsampling
a<-resamp(peewit,f=22050,g=11025)
# oversampling
b<-resamp(peewit,f=22050,g=44100)
Time reverse of a time wave

Description
Reverse the wave along the time axis.

Usage
revw(wave, f, env = TRUE, ifreq = TRUE,
plot = FALSE, output = "matrix", ...)

Arguments
wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
env logical, if TRUE the amplitude envelope is reversed.
ifreq logical, if TRUE the instantaneous frequency is reversed.
plot logical, if TRUE returns an oscillographic plot of the reversed wave (by default FALSE).
output character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
... other oscillo graphical parameters.

Details
If plot is TRUE returns an oscillogram of the reversed wave. The amplitude and the instantaneous frequency can be independently reversed thanks to the arguments env and ifreq. See the examples.

Value
If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Author(s)
Jerome Sueur <sueur@mnhn.fr>

References

See Also
oscillo, addsilw, deletew, fadew, pastew, mutew
Examples

data(tico)
# simple reverse
rew(tico,f=22050,plot=TRUE)
# envelope reverse only
rew(tico,f=22050,ifreq=FALSE, plot=TRUE)
# instantaneous frequency reverse only
rew(tico,f=22050,env=FALSE, plot=TRUE)

rmam

Remove the amplitude modulations of a time wave

Description

This function removes the amplitude modulation of a time wave through the Hilbert amplitude envelope.

Usage

rmam(wave, f, plot = FALSE, listen = FALSE, output = "matrix", ...)

Arguments

- wave: an R object.
- f: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
- plot: logical, if TRUE returns an oscillographic plot of the new time wave (by default FALSE).
- listen: if TRUE the new sound is played back.
- output: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
- ...: other oscillo graphical parameters.

Details

The new time wave is obtained by dividing the original time wave by its Hilbert amplitude envelope.

Value

If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Author(s)

Jerome Sueur <sueur@mnhn.fr>
References


See Also

hilbert.

Examples

```r
# generate a new sound with amplitude modulation
a<-synth(f=8000, d=1, cf=1500, am=c(50,10))
# remove the amplitude modulation and plot the result
rmam(a,f=8000,plot=TRUE)
```

**Description**

This function removes background noise by smoothing

**Usage**

`rmnoise(wave, f, output = "matrix", ...)`

**Arguments**

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `output`: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
- `...`: other `smooth.spline` arguments.

**Details**

This function is based on `smooth.spline`. You can use the arguments of the later to modify the smoothing.

**Value**

A new wave is returned. The class of the returned object is set with the argument output.

**Note**

Low frequency noise might not be removed out properly.
Author(s)
Jerome Sueur <sueur@mnhn.fr>

See Also
afilter, noiselw

Examples
# synthesis of a 440 Hz sound with background noise
n <- noiselw(d=1,f=8000)
s <- synth(d=1,f=8000,cf=440)
ns <- n+s
# remove noise (but low frequency content still there)
a <- rmnoise(ns,f=8000)

---

rmoffset

Remove the offset of a time wave

Description
This function removes the offset of a time wave.

Usage

rmoffset(wave, f, FUN = mean, plot = FALSE, output = "matrix", ...)

Arguments

wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
FUN a function used to apply the offset correction. See Details.
plot logical, if TRUE returns an oscillographic plot of the wave after removing the offset (by default FALSE).
output character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
... other oscillo graphical parameters.

Value
The offset is removed by subtracting the wave by its mean (argument FUN). But other function can be used. For instance, it can be more appropriate to use the median to remove the offset and transients. See Examples.
If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.
Author(s)
Jerome Sueur <sueur@mnhn.fr>

See Also
oscillo

Examples

data(tico)
# artificially generates an offset
tico2<-tico+0.1
# see the wave with an offset
oscillo(tico2, f=22050)
# remove the offset with the mean (by default)
rmoffset(tico2, f=22050, plot=TRUE)
# remove the offset with the median
rmoffset(tico2, f=22050, FUN=median, plot=TRUE)

---

 rms  Root Mean Square

Description
This function computes the root mean square or quadratic mean.

Usage

rms(x, ...)

Arguments

x  an R object

...  further arguments passed to mean

Details
The Root Mean Square or quadratic mean is computed according to:

\[ RMS = \sqrt{\frac{1}{n} \times \sum_{i=1}^{N} x_i^2} \]

Value
A numeric vector of length 1
**roughness**

**Author(s)**

Jerome Sueur <sueur@mnhn.fr>

**See Also**

`mean`

**Examples**

```r
# simple rms
rms(1:10)
# rms of a normalized envelope
data(sheep)
env <- env(sheep, f=8000)
rms(env)
```

<table>
<thead>
<tr>
<th>roughness</th>
<th>Roughness or total curvature</th>
</tr>
</thead>
</table>

**Description**

This function computes the roughness or total curvature of a curve, i.e. of a time wave or of a spectrum.

**Usage**

`roughness(x, std = FALSE)`

**Arguments**

- `x`: a vector
- `std`: a logical, if set to TRUE then `x` is standardized by its maximum.

**Details**

Roughness or total curvature is the integrated squared second derivative:

\[
\text{roughness} = \int |D^2x(t)|^2 \, dt
\]

**Value**

A vector of length 1.

**Note**

The value has not unit.
rugosity of a time wave

Author(s)
Jerome Sueur

References

See Also
rug0, rms, sh, th, h.

Examples

data(tico)
spec <- meanspec(tico, plot=FALSE)[,2]
roughness(spec)


data(tico)
spec <- meanspec(tico, plot=FALSE)[,2]
roughness(spec)

Description
This function computes the rugosity of a time wave or time series

Usage
rug0(x, ...)

Arguments
x a vector
... other mean parameters.

Details
The formula has been slightly modified from Mezquida & Martinez (2009: 826) to fit with the classical definition of the root-mean-square (see rms).
The rugosity is then computed as following:

\[
\text{rug0} = \sqrt{\frac{\sum_{i=1}^{n-1} (x_{i+1} - x_i)^2}{n}}
\]

for a vector x of length n.

Value
A vector of length 1.
Note

The rugosity of a noisy signal will tend to be higher than that of a pure tone signal, all other things being equal.

Author(s)

Jerome Sueur

References


See Also

roughness, rms, sh, th, H.

Examples

```r
data(tico); tico <- tico@left  # rugosity of the original recording normalised
rugo(tico/max(tico))
# synthesis of white noise with the same duration as tico
noise <- noisew(d=length(tico)/22050, f=22050)
# tico is normalised to get similar amplitude with the noise
tico.norm <- tico/max(tico)
# addition of noise to tico
tico.noisy <- tico.norm + 0.5*noise
# new rugosity (higher) on normalised signal
rugo(tico.noisy/max(tico.noisy))```

savewav

*Save a .wav file*

Description

Save sound data as .wav file

Usage

`savewav(wave, f, filename = NULL, rescale = NULL, ...)`
Arguments

- **wave**: an R object.
- **f**: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- **filename**: name of the new file. (by default the name of `wave`).
- **rescale**: a numeric vector of length 2 giving the lower (negative value) and upper (positive value) amplitude limits of the .wav file to be exported.
- **...**: other arguments to be passed to `writeWave`.

Details

This function uses three functions from the package `tuneR`: `Wave`, `normalize` and `writeWave`.

Note

The file automatically overwrites an existing file with the same name.
The amplitude (volume) of the .wav file is normalized by defaults but can be changed with the argument `rescale`. See examples

Author(s)

Jerome Sueur <sueur@mnhn.fr>, Ethan C. Brown for the argument `rescale`

See Also

`export`.

Examples

```r
require(tuneR)
a<-synth(f=8000,d=2,cf=2000,plot=FALSE)
# the name of the file is automatically the name of the object
# here: "a.wav"
savewav(a,f=22050)
unlink("a.wav")
# if you wish to change the name, use the 'file' argument
savewav(a,f=22050,file="b.wav")
unlink("b.wav")
# if you wish to change the amplitude of the file, use the argument 'rescale'
# this will turn down the volume of a 16 bit sound
# which amplitude was originally ranging between -2^15 and +2^15
savewav(a, f=22050, file="c.wav", rescale=c(-1500,1500))
unlink("c.wav")
```
SAX

Symbolic Aggregate approxImation

Description

This function converts a numeric times series into a series of letters with a specific length and alphabet.

Usage

SAX(x, alphabet_size, PAA_number, breakpoints = "gaussian", collapse = NULL)

Arguments

- **x**: a numeric vector.
- **alphabet_size**: a numeric vector of length 1 setting the size of the alphabet.
- **PAA_number**: a numeric vector of length 1 setting the number of elements (subsequences) of the Piecewise Aggregate Approximation (PAA).
- **breakpoints**: either a character vector ("gaussian", "quantiles") or a numeric vector specifying the sorted values of the breakpoints along the distribution of x. See details and examples.
- **collapse**: a character vector of length 1, specifying the way to collapse the output letters, see paste. By default letters are returned separated.

Details

The SAX method has been developed to reduce the dimensionality of a numerical series into a short chain of characters. SAX follows a two-step process: (1) Piecewise Aggregate Approximation (PAA) and (2) conversion a PAA sequence into a series of letters.

PAA consists in a Z-normalisation, a segmentation of the series of length n into w segments, and the computation of each segment average.

The conversion of the PAA into a series of letters is achieved by attributing with equiprobability each value of the PAA to a letter in reference to a Gaussian distribution. This process therefore assumes that the distribution of the numeric series x follows a Gaussian distribution. To relax the constraints of normality we here added the possibility to directly work on the quantiles of the original data distribution or to specify particular breakpoints along the distribution of x. See the examples.

Value

A character vector of length (when collapse is NULL) or number of character (when collapse is not NULL) corresponding to PAA_number argument.
Note

SAX has been used recently to search similar times series in a soundcape data base (Kasten et al., 2012).

Author(s)

Laurent Lellouch. An improvement added by Pavel Senin.

References


See Also

discrets, symba, soundscapespec

Examples

data(tico)
spec <- soundscapespec(tico, plot=FALSE)[,2]
SAX(spec, alphabet = 5, PAA = 10)

# change breakpoints
SAX(spec, alphabet = 5, PAA = 10, breakpoints="quantiles")
SAX(spec, alphabet = 5, PAA = 10, breakpoints=c(0, 0.5, 0.75, 1))
SAX(spec, alphabet = 5, PAA = 10, breakpoints=c(0, 0.33, 0.66, 1))

# different output formats
SAX(spec, alphabet = 5, PAA = 10, collapse="")
SAX(spec, alphabet = 5, PAA = 10, collapse="-")

---

**sddb**

*Standard deviation of dB values*

Description

This function estimates the standard deviation of dB values

Usage

`sddb(x, level = "IL")`
Arguments

- **x**
  - a numeric vector.

- **level**
  - intensity level ("IL") or sound pressure level ("SPL")

Details

The standard deviation of dB values is not linear. The function is an estimation not an exact computation which is not possible.

Value

A numeric vector of length 1.

Author(s)

Jérôme Sueur

References


See Also

- `meandB`, `moredB`, `convSPL`, `dBweight`

Examples

```r
sddb(c(89,90,95))
sddb(c(89,90,95), level="SPL")
```

---

**seedata**

A quick look at quantitative data

Description

See quantitative data at a glance

Usage

```r
seedata(data, na.rm = FALSE, col = "grey")
```

Arguments

- **data**
  - a numeric vector describing quantitative data.

- **na.rm**
  - logical, if TRUE removes NA.

- **col**
  - main color.
Details

The red curves depict the corresponding Normal law (same mean and sd as data).

Value

A multi-plot graphic is returned.

Author(s)

Caroline Simonis <csimonis@mnhn.fr> and Jerome Sueur <sueur@mnhn.fr>.

Examples

seewave

seedata(rnorm(1000))

---

seewave

Sound analysis and synthesis

Description

seewave provides functions for analysing, manipulating, displaying, editing and synthesizing time waves (particularly sound). This package processes in particular time analysis (oscillograms and envelopes), spectral content, resonance quality factor, entropy, cross correlation and autocorrelation, zero-crossing, frequency coherence, dominant frequency, analytic signal, 2D and 3D spectrograms.

Details

Package: seewave
Type: Package
Version: 2.1.0
Date: 2018-03-02
License: GPL version 2 or newer
Contributors: Ethan C. Brown, Camille Desjonqueres, Marion Depraetere, Francois Fabianek, Amandine Gasc, Eric Kasten, Laurent Lellouch, Stefanie LaZerte, Jonathan Lees, Jean Marchal, Sandrine Pavoine, David Pinaud, Luis J. Villanueva-Rivera, Zev Ross, Carl G. Witthoft, Hristo Zhivomirov
Acknowledgments: Andrey Anikin, Michel Baylac, Charlotte Cure, Denis Dupeyron, Kurt Fristrup, Arnold Fertin, Kurt Hornik, Emiliano A. Laca, Uwe Ligges, Morgane Paprin, Emmanuel Paradis, Daniel Ridley-Ellis,
setenv

Brian Ripley, Jesse Ross,  
Zev Ross, Pavel Senin,  
Arvind Sowmyan, Simon Urbanek  
George Zhang

Webpage: http://rug.mnhn.fr/seewave
Discussion group : http://groups.google.com/group/seewave

Author(s)
Jerome Sueur <sueur@mnhn.fr>  
Thierry Aubin  
Caroline Simonis  
Maintainer: Jerome Sueur <sueur@mnhn.fr>

setenv

Set the amplitude envelope of a time wave to another one

Description
This function sets the amplitude envelope of a time wave to another one

Usage
setenv(wave1, wave2, f, envt="hil", msmooth = NULL, ksmooth = NULL,  
plot = FALSE, listen = FALSE, output = "matrix", ...)

Arguments
wave1 a first R object.
wave2 a second R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded  
in wave.
envt the type of envelope to be used for wave2: either "abs" for absolute amplitude  
envelope or "hil" for Hilbert amplitude envelope. See env.
msmooth a vector of length 2 to smooth the amplitude envelope of wave2 with a mean  
sliding window. The first component is the window length (in number of points).  
The second component is the overlap between successive windows (in %). See  
env.
ksmooth kernel smooth via kernel to apply to the amplitude envelope of wave2. See env.
plot    if TRUE returns the oscillogram of the new time wave (by default FALSE).
listen  if TRUE the new sound is played back.
output  character string. the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
...
other oscillo graphical parameters.

Details
wave1 and wave2 can have different duration (length)
Smoothing the envelope with smooth or ksmooth can significantly change the value returned.

Value
If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Author(s)
Jerome Sueur <sueur@mnhn.fr>

See Also
drawenv, env, synth

Examples
data(tico)
a<-synth(d=1,f=22050,cf=1000)
# apply 'tico' ammpitude envelope to 'a' that has a square amplitude envelope
setenv(a,tico,f=22050,plot=TRUE)
# the same but with smoothing the envelope
setenv(a,tico,f=22050,ksmooth=kernel("daniell",50),plot=TRUE)

sfm                 Spectral Flatness Measure

Description
This function estimates the flatness of a frequency spectrum.

Usage
sfm(spec)

Arguments
spec a data set resulting of a spectral analysis obtained with spec or meanspec (not in dB).
Details

SFM is calculated as the ratio between the geometric mean and the arithmetic mean:

\[ F = N \times \frac{\sqrt[1]{\prod_{i=1}^{N} y_i}}{\sum_{i=1}^{N} y_i} \]

with:
\[ y = \text{relative amplitude of the } i \text{ frequency,} \]
and \( N = \text{number of frequencies.} \)

Value

A single value varying between 0 and 1 is returned. The value has no unit.

Note

The SFM of a noisy signal will tend towards 1 whereas the SFM of a pure tone signal will tend towards 0.
See sh for another measure of signal noisiness/pureness.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

sh, csh

Examples

```r
a <- synth(f=8000, d=1, cf=2000, plot=FALSE)
specia <- spec(a, f=8000, at=0.5, plot=FALSE)
sfm(specia)
# [1] 0
b <- noise(d=1, f=8000)
specb <- spec(b, f=8000, at=0.5, plot=FALSE)
sfm(specb)
# [1] 0.8233202
```

---

sh Shannon and Renyi spectral entropy

Description

This function computes the Shannon or Renyi entropy of a frequency spectrum

Usage

```
sh(spec, alpha = "shannon")
```
Arguments

**spec**
a data set resulting of a spectral analysis obtained with `spec` or `meanspec` (not in dB).

**alpha**
a character string, by default "shannon" to compute Shannon entropy, "simpson" to compute Simpson entropy otherwise a numeric vector of length 1 with a value superior to 0 but different to 1 to compute Renyi entropy. See the examples.

Details

. Shannon spectral entropy is calculated according to:

\[ S = - \sum_{i=1}^{N} y_i \log_2(y_i) \]

. Simpson or Gini-Simpson spectral entropy (or index) is computed according to:

\[ GS = 1 - \sum_{i=1}^{N} y_i^2 \]

. Renyi spectral entropy of order alpha is calculated according to:

\[ R = \frac{1}{1-\alpha} \times \log_2\left(\sum_{i=1}^{N} y_i^\alpha\right) \]

with

\[ \alpha \geq 0 \]

\[ \alpha \neq 1 \]

\( y = \) relative amplitude of the \( i \) frequency,

\[ \sum_{i=1}^{N} y_i = 1 \]

and \( N \) = number of frequencies.

Value

A numeric vector of length 1 is returned. The value has no unit.

Note

The Shannon spectral entropy of a noisy signal will tend towards 1 whereas the Shannon spectral entropy of a pure tone signal will tend towards 0. See Han et al. for details regarding the Renyi entropy.

Author(s)

Jerome Sueur and Laurent Lellouch
sheep

References


See Also
csh, th, h, sfm

Examples

a<-synth(f=8000,d=1,cf=2000,plot=FALSE)
specia<-spec(a,f=8000,at=0.5,plot=FALSE)
## Shannon spectral entropy
sh(specia)
# [1] 0.2336412
b<-noise(d=1,f=8000)
specb<-spec(b,f=8000,at=0.5,plot=FALSE)
sh(specb)
# close to 1
## Renyi spectral entropy
sh(specb, alpha=2)
sh(specb, alpha=3)

sheep Sheep bleat

Description

Recording of a sheep bleat.

Usage
data(sheep)

Format

A Wave object.
**Details**

Duration = 2.47 s. Sampling frequency = 8000 hz.

**Source**

Recording by Frederic Sebe.

**Examples**

```r
data(sheep)
oscillo(sheep,f=8000)
```

---

**simspec**

*Similarity between two frequency spectra*

**Description**

This function estimates the similarity between two frequency spectra.

**Usage**

```r
simspec(spec1, spec2, f = NULL, mel = FALSE, norm = FALSE, PMF = FALSE, plot = FALSE, type = "1", lty =c(1, 2, 3), col = c(2, 4, 1), flab = NULL, alab = "Amplitude (percentage)", flim = NULL, alim = NULL, title = TRUE, legend = TRUE, ...)
```

**Arguments**

- `spec1` a first data set resulting of a spectral analysis obtained with `spec` or `meanspec` (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).
- `spec2` a first data set resulting of a spectral analysis obtained with `spec` or `meanspec` (not in dB). This can be either a two-column matrix (col1 = frequency, col2 = amplitude) or a vector (amplitude).
- `f` sampling frequency of waves used to obtain `spec1` and `spec2` (in Hz). Not necessary if `spec1` and/or `spec2` is a two columns matrix obtained with `spec` or `meanspec`.
- `mel` a logical, if TRUE the (htk)-mel scale is used.
- `norm` a logical, if TRUE `spec1` and `spec2` are normalised (scaled) between 0 and 1.
- `PMF` a logical, if TRUE `spec1` and `spec2` are transformed into probability mass functions.
- `plot` logical, if TRUE plots both spectra and similarity function (by default FALSE).
**simspec**

- **type** if `plot` is TRUE, type of plot that should be drawn. See `plot` for details (by default "l" for lines).

- **lty** a vector of length 3 for the line type of `spec1`, `spec2` and of the similarity function if type="l".

- **col** a vector of length 3 for the colour of `spec1`, `spec2`, and the similarity function.

- **flab** title of the frequency axis.

- **alab** title of the amplitude axis.

- **flim** the range of frequency values.

- **alim** range of amplitude axis.

- **title** logical, if TRUE, adds a title with S value.

- **legend** logical, if TRUE adds a legend to the plot.

- **...** other `plot` graphical parameters.

**Details**

Spectra similarity is assessed according to:

\[ S = \frac{100}{N} \sum_{i=1}^{N} \frac{\min{\text{spec1}(i), \text{spec2}(i)}}{\max{\text{spec1}(i), \text{spec2}(i)}} \]

with \( S \) in %.

**Value**

The similarity index is returned. This value is in %.

When `plot` is TRUE, both spectra and the similarity function are plotted on the same graph. The similarity index is the mean of this function.

**Author(s)**

Jerome Sueur, improved by Laurent Lellouch

**References**


**See Also**

`spec`, `meanspec`, `corspec`, `difspect`, `diffenv`, `kl.dist`, `ks.dist`, `logspect.dist`, `itakura.dist`
Examples

```r
a <- noise(f=8000, d=1)
b <- synth(f=8000, d=1, cf=2000)
c <- synth(f=8000, d=1, cf=1000)
d <- noise(f=8000, d=1)
speca <- spec(a, f=8000, at=0.5, plot=FALSE)
specb <- spec(b, f=8000, at=0.5, plot=FALSE)
specc <- spec(c, f=8000, at=0.5, plot=FALSE)
specd <- spec(d, f=8000, at=0.5, plot=FALSE)
simspec(speca, speca)
simspec(speca, specb)
simspec(speca, specc, plot=TRUE)
simspec(specb, specc, plot=TRUE)
```

Description

This function tries to smooth with a sum sliding window a time wave, and then to remove residual noise.

Usage

```r
smoothw(wave, f, wl, padding=TRUE, output = "matrix")
```

Arguments

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl`: window length in number of points (samples).
- `padding`: a logical, if `TRUE` add 0 values at the start and end of the file to match wave length (duration).
- `output`: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
Details

A window slides along the signal and sums up the sample amplitude values. Zero values are added at the end of the wave to keep wave length (duration).

Value

A new wave is returned. The class of the returned object is set with the argument `output`. If `padding` is `TRUE`, the new wave starts and ends up with 0 values to match the size of `wave`.

Warning

This function should be used with care as this kind of filter may change the frequency content of the sound. See the examples section for an illustration.

Author(s)

Jerome Sueur

See Also

fir, filter

Examples

```r
# An example to show that smooth() may change
# the frequency content of your sound
data(orni)
orni2 <- smooth(orni, wl=2, out="Wave")
orni10 <- smooth(orni, wl=10, out="Wave")
orni50 <- smooth(orni, wl=50, out="Wave")
orni100 <- smooth(orni, wl=100, out="Wave")
meanspec(orni)
lines(meanspec(orni2, plot=FALSE), col=2)
lines(meanspec(orni10, plot=FALSE), col=3)
lines(meanspec(orni50, plot=FALSE), col=4)
lines(meanspec(orni100, plot=FALSE), col=5)
legend("topright", col=1:5, lty=1, legend=c("original","wl=2","wl=10","wl=50","wl=100"))
```

songmeter

Reading and interpreting SongMeter file name

Description

This function reads and decomposes the files names generated by SongMeters, audio digal recorders produced by the society Wildlife Acoustics.

Usage

`songmeter(x)`
Arguments

x          a character vector with file names, either .wac or .wav

Details

The digital recorder SongMeter (either SM2, SM3, or SM4 device model) produced by the society 'Wildlife Acoustics' generates '.wav' or '.wac' files which names include useful information. Here are the character format of the files:

- **SM2 or SM4**: PREFIX_YYYYMMDD_HHMMSS.wav
- **SM3**:  
  - *without geolocalisation* PREFIX_XXX_YYYYMMDD_HHMMSS.wav  
  - *with geolocalisation* PREFIX_XXX_YYYYMMDD$HHMMSS.wav

with:

- **PREFIX**: prefix set when programming the SongMeter
- **XXX**: microphone information
- **YYYY**: year
- **MM**: month
- **DD**: day
- **HH**: hour
- **MM**: month
- **SS**: minute

This information is read and decomposed by the function songmeter().

Please note that the function does not read the content of audio file but the name of the file.

Value

The function returns a data.frame with the following columns:

- **model**: device model, either "SM2/SM4" or "SM3"
- **prefix**: prefix of the file, specifying for instance to recording site
- **mic**: microphone information specifying if the recording is mono left channel ("monoL"), mono right ("monoR") or stereo ("stereo"). This works for SM3 only, NA for SM2
- **year**: year of recording
- **month**: month of recording
- **day**: day of recording
- **hour**: hour of recording
- **min**: minute of recording
- **sec**: second of recording
- **time**: all time of recording information in POSIX format
- **geo**: logical, TRUE if the device was GPS synchronized
The file names of Songmeters may change with time. There is no guarantee that the function will be perfectly updated.

Author(s)
Jerome Sueur

References
See Wildlife Acoustics website for details regarding the SongMeters 2, 3 and 4: http://www.wildlifeacoustics.com/

See Also
strptime for the POSIX time format.

Examples
```r
goldfile <- "MNHN_20141225_234500.wav"  ## SM2 file
goldfile2 <- "CNRS_81_20130824_153000.wav"  ## SM3 file without geolocalisation
goldfile3 <- "PARIS_8_S_20150418019555.wav"  ## SM3 file with geolocalisation
goldfile4 <- "MNHN_20141225_234500.txt"  ## not a .wav or a .wac file
file5 <- "myfile.wav"  ## not a Wildlife Acoustics filename
goldfiles <- c(goldfile, goldfile2, goldfile3, goldfile4, goldfile5)
songmeter(goldfiles)
```

Description
This function returns a kHz binned spectrum as described by Kasten et al. (2012) for the description of a soundscape.

Usage
```r
soundscapespec(wave, f, wl = 1024, wn = "hamming", ovlp = 50,
plot = TRUE, xlab = "Frequency (kHz)", ylim = c(0, 1), ...)
```

Arguments
```r
wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
wl length of the window for the analysis (even number of points, by default = 1024).
wn window name, see ftwindow (by default "hamming").
```
overlap between two successive analysis windows (in %), by default = 50%.

if TRUE returns a barplot.

title of the barplot x axis.

range of the barplot y axis.

other barplot graphical parameters.

The soundscape frequency spectrum is based on the computation of a spectrogram power spectral density using Welch’s method (Welch & June, 1967). Parameters used in Kasten et al. (2012) were a Hamming window of 1024 samples with 50% of overlap and are used here as default values.

A two-column numeric matrix, the first column returning the frequency (kHz) bands and the second column returning the power value within each frequency band. A barplot is returned when plot is TRUE.

Jerome Sueur and Eric Kasten


spec, meanspec, SAX, NDSI

## Note that 'tico' is not a soundscape recording...
data(tico)
soundscapespec(tico, plot=TRUE, col="darkgreen")
**sox**  

*Calls SoX*

**Description**

This function calls SoX, the Swiss Army knife of sound processing programs.

**Usage**

sox(command, exename = NULL, path2exe = NULL)

**Arguments**

- **command**: the SoX command to invoke.
- **exename**: a character string specifying the name of the SoX binary file. If NULL, the default name "sox" will be used for Linux OS and "sox.exe" for Windows OS.
- **path2exe**: a character string giving the path to the SoX binary file. If NULL, the default path "c:\sox-14-4-0\" will be used for Windows OS.

**Details**

See the documentation of SoX for proper use.

**Note**

Sox must be installed to use this function but not to install the package seewave. As mentioned on the SoX webpage, the primary development platform is Linux. Using SoX with Windows from R might not be straightforward. In particular, it is advisable to pay attention to file path and exe name.

**Author(s)**

Jerome Sueur and Stefanie LaZerte

**References**


**Examples**

```
## Not run:
## Generate a simple sound file at 440 Hz
s <- synth(cf=440, f= 8000, d=1, output="Wave")
savewav(s, file="mysound.wav")
## Plays the file
sox("mysound.wav", exe="play")
## Slows down the audio tempo (but not its pitch)
sox("mysound.wav mysounds.wav tempo 0.5")
## Cuts the file
```
```r
sox("myslowsound.wav myslowcuttedsound.wav trim 0.25 0.75")
## Deletes example files
file.remove("mysound.wav", "myslowsound.wav", "myslowcuttedsound.wav")

## End(Not run)
```

**spec**

*Frequency spectrum of a time wave*

**Description**

This function returns the frequency spectrum (*i.e.* the relative amplitude of the frequency content) of a time wave. Results can be obtained either as absolute or dB data.

**Usage**

```r
spec(wave, f, wl = 512, wn = "hanning", fftw = FALSE, norm = TRUE,
    scaled = FALSE, PSD = FALSE, PMF = FALSE, correction="none",
    dB = NULL, dBref = NULL, at = NULL, from = NULL, to = NULL,
    identify = FALSE, col = "black", cex = 1,
    plot = 1, flab = "Frequency (kHz)",
    alab = "Amplitude", flim = NULL,
    alim = NULL, type="l", ...)
```

**Arguments**

- **wave**
  - an R object.
- **f**
  - sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
- **wl**
  - if at is not null, length of the window for the analysis (by default = 512).
- **wn**
  - window name, see ftwindow (by default "hanning").
- **fftw**
  - if TRUE calls the function FFT of the library fftw for faster computation. See Notes of the function spectro.
- **norm**
  - if TRUE the spectrum is normalised by its maximum.
- **scaled**
  - if TRUE the spectrum is scaled by the length of the FFT.
- **PSD**
  - if TRUE return Power Spectrum Density, *i.e.* the square of the spectrum.
- **PMF**
  - if TRUE return Probability Mass Function, *i.e.* the probability distribution of frequencies.
- **correction**
  - a character vector of length 1 to apply an amplitude ("amplitude") or an energy ("energy") correction to the FT window. This argument is useful only when one wish to obtain absolute values that is when norm=FALSE, scaled=FALSE, and PMF=FALSE. By default no correction is applied ("none").
- **dB**
  - a character string specifying the type dB to return: "max0" for a maximum dB value at 0, "A", "B", "C" and "D" for common dB weights.
Spec

- **dBRef**: a dB reference value when dB is not NULL. NULL by default but should be set to 2*10e-5 for a 20 microPa reference (SPL).
- **at**: position where to compute the spectrum (in s).
- **from**: start mark where to compute the spectrum (in s).
- **to**: end mark where to compute the spectrum (in s).
- **identify**: to identify frequency and amplitude values on the plot with the help of a cursor.
- **col**: colour of the spectrum.
- **cex**: pitch size of the spectrum.
- **plot**: if 1 returns frequency on x-axis, if 2 returns frequency on y-axis, (by default 1).
- **flab**: title of the frequency axis.
- **alab**: title of the amplitude axis.
- **flim**: range of frequency axis.
- **alim**: range of amplitude axis.
- **type**: if plot is TRUE, type of plot that should be drawn. See `plot` for details (by default "l" for lines).
- **...**: other `plot` graphical parameters.

**Details**

If `at`, `from` or `to` are FALSE then `spec` computes the spectrum of the whole signal.

**Value**

This function returns a two-column matrix, the first column corresponding to the frequency axis, the second column corresponding to the amplitude axis. If `identify` is TRUE, `spec` returns a list with two elements:

- **freq**: the frequency of the points chosen on the spectrum
- **amp**: the relative amplitude of the points chosen on the spectrum

**Warning**

The argument `peaks` is no more available (version > 1.5.6). See the function `fpeaks` for peak(s) detection.

**Note**

This function is based on `fft`.

**Author(s)**

Jerome Sueur
specprop

Spectral properties

Description

This function returns a list of statistical properties of a frequency spectrum.
Usage

specprop(spec, f=NULL, str = FALSE, flim=NULL, mel=FALSE, plot = FALSE, type = "l", xlab=NULL, ylab = NULL, col.mode = 2, col.quartiles = 4, ...)

Arguments

spec a data set resulting of a spectral analysis obtained with spec or meanspec (not in dB).
f sampling frequency of spec (in Hz).
str logical, if TRUE returns the results in a structured table.
flim a vector of length 2 to specify the frequency limits of the analysis (in kHz)
mel a logical, if TRUE the (htk-)mel scale is used.
plot if 1 returns the spectrum, if 2 returns the cumulative spectrum, both of them with the first quartile, the third quartile, the median and the mode plotted (by default FALSE).
type if plot is TRUE, type of plot that should be drawn. See plot for details (by default "l" for lines).
xlab label of the x axis.
ylab label of the y axis.
col.mode colour of the mode segments (by default blue).
col.quartiles colour of the quartiles segments (by default red).
... other arguments to be passed to plot

Details

The spectrum is converted in a probability mass function (PMF).
If a selected value has to be selected with $, the argument str has to be set to FALSE.

Value

A list of 15 values is returned

mean mean frequency (see mean)
sd standard deviation of the mean (see sd)
sem standard error of the mean
median median frequency (see median)
mode mode frequency, i.e. the dominant frequency
Q25 first quartile (see quantile)
Q75 third quartile (see quantile)
IQR interquartile range (see IQR)
cent centroid, see note
skewness  skewness, a measure of asymmetry, see note
kurtosis  kurtosis, a measure of peakedness, see note
sfm  spectral flatness measure (see sfm)
sh  spectral entropy (see sh)
prec  frequency precision of the spectrum

Note

Centroid is computed according to:

\[ C = \sum_{i=1}^{N} x_i \times y_i \]

with:
\( x = \) frequencies, \( y = \) relative amplitude of the \( i \) frequency,
\( N = \) number of frequencies.

Skewness is computed according to:

\[ S = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^3}{N - 1} \times \frac{1}{\sigma^3} \]

\( S < 0 \) when the spectrum is skewed to left,
\( S = 0 \) when the spectrum is symetric,
\( S > 0 \) when the spectrum is skewed to right.
Spectrum asymmetry increases with \( |S| \).

Kurtosis is computed according to:

\[ K = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^4}{N - 1} \times \frac{1}{\sigma^4} \]

\( K < 3 \) when the spectrum is platykurtic, \( i.e. \) it has fewer items at the center and at the tails than the normal curve but has more items in the shoulders,
\( K = 3 \) when the spectrum shows a normal shape,
\( K > 3 \) when the spectrum is leptokurtic, \( i.e. \) it has more items near the center and at the tails, with fewer items in the shoulders relative to normal distribution with the same mean and variance.

Author(s)

Jerome Sueur and Caroline Simonis, and a patch by Jesse Ross (Dec. 2012)

Examples

data(orni)
a<-meanspec(orni,f=22050,plot=FALSE)
specprop(a,f=22050)
# to get a single measure of the list
spectro

2D-spectrogram of a time wave

Description

This function returns a two-dimension spectrographic representation of a time wave. The function corresponds to short-term Fourier transform. An amplitude contour plot can be overlaid.

Usage

spectro(wave, f, wl = 512, wn = "hanning", zp = 0,
      ovlp = 0, complex = FALSE, norm = TRUE, correction="none",
      fftw = FALSE, dB = "max0", dBref = NULL, plot = TRUE,
      flog = FALSE, grid = TRUE, osc = FALSE, scale = TRUE, cont = FALSE,
      collevels = NULL, palette = spectro.colors,
      contlevels = NULL, colcont = "black",
      colbg = "white", colgrid = "black",
      colaxis = "black", collab="black",
      cexlab = 1, cexaxis = 1,
      tlab = "Time (s)",
      flab = "Frequency (kHz)",
      alab = "Amplitude",
      scalabe = "Amplitude\n(dB)",
      main = NULL,
      scalefontlab = 1, scalecexlab =0.75,
      axisX = TRUE, axisY = TRUE, tlim = NULL, trel = TRUE,
      flim = NULL, flimd = NULL,
      widths = c(6,1), heights = c(3,1),
      oma = rep(0.4),
      listen=FALSE,
      ...)

specprop(a,f=22050) # to get the results structured
specprop(a,f=22050,str=TRUE)
# to limit the analysis between 4 and 6 kHz
specprop(a,f=22050,flim=c(4,6),str=TRUE)
# plots
specprop(a,f=22050,plot=1)
specprop(a,f=22050,plot=2)
# (htk-)mel scale
require(tuneR)
mel <- melfcc(orni, nbands = 256, dcttype = "t3", fbtype = "htkmel", spec_out=TRUE)
melspec.mean <- apply(mel$aspectrum, MARGIN=2, FUN=mean)
specprop(melspec.mean, f=22050, mel=TRUE)
# be aware that flim is always given in kHz even if mel=TRUE
specprop(melspec.mean, f=22050, flim=c(4,6), mel=TRUE, plot=TRUE)
**Arguments**

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl`: window length for the analysis (even number of points) (by default = 512).
- `wn`: window name, see `ftwindow` (by default “hanning”).
- `zp`: zero-padding (even number of points), see Details.
- `ovlp`: overlap between two successive windows (in %).
- `complex`: if TRUE the STFT will be returned as complex numbers.
- `norm`: if TRUE the STFT is normalised (i.e. scaled) by its maximum.
- `correction`: a character vector of length 1 to apply an amplitude (“amplitude”) or an energy (“energy”) correction to each FT window. This argument is useful only when one wish to obtain absolute values that is when norm=FALSE. By default no correction is applied (“none”).
- `fftw`: if TRUE calls the function FFT of the library `fftw`. See Notes.
- `dB`: a character string specifying the type dB to return: “max0” (default) for a maximum dB value at 0, “A”, “B”, “C” and “D” for common dB weights. If set to NULL, then a linear scale is used.
- `dBref`: a dB reference value. NULL by default but should be set to 2*10e-5 for a 20 microPa reference.
- `plot`: logical, if TRUE plots the spectrogram (by default TRUE).
- `flog`: a logical to plot the frequency on a logarithmic scale.
- `grid`: logical, if TRUE plots a y-axis grid (by default TRUE).
- `osc`: logical, if TRUE plots an oscillogram beneath the spectrogram (by default FALSE).
- `scale`: logical, if TRUE plots a dB colour scale on the right side of the spectrogram (by default TRUE).
- `cont`: logical, if TRUE overplots contour lines on the spectrogram (by default FALSE).
- `collevels`: a set of levels which are used to partition the amplitude range of the spectrogram (in dB).
- `palette`: a color palette function to be used to assign colors in the plot, see Details.
- `contlevels`: a set of levels which are used to partition the amplitude range for contour overplot (in dB).
- `colcont`: colour for cont plotting.
- `colbg`: background colour.
- `colgrid`: colour for grid plotting.
- `colaxis`: color of the axes.
- `collab`: color of the labels.
- `cexlab`: size of the labels.
- `cexaxis`: size of the axes.
Details

Following Heisenberg uncertainty principle, the short-term Fourier transform cannot be precisely in both time and frequency. The temporal and frequency precisions of the function are actually dependent of the \( \text{wl} \) value. Choosing a high \( \text{wl} \) value will increase the frequency resolution but reduce the temporal one, and \textit{vice versa}. The frequency precision is obtained by calculating the ratio \( f/\text{wl} \), and the temporal precision is obtained by calculating the reverse ratio \( \text{wl}/f \). This problem can be reduced in some way with \( \text{zp} \) that adds 0 values on both sides of the analysis window. This increases frequency resolution without altering time resolution.

Any colour palette can be used. In particular, it is possible to use other palettes coming with \texttt{seewave}: \texttt{temp.colors}, \texttt{reverse.gray.colors.1}, \texttt{reverse.gray.colors.2}, \texttt{reverse.heat.colors}, \texttt{reverse.terrain.colors}, \texttt{reverse.topo.colors}, \texttt{reverse.cm.colors} corresponding to the reverse of \texttt{heat.colors}, \texttt{terrain.colors}, \texttt{topo.colors}, \texttt{cm.colors}.

Use \texttt{locator} to identify points.

Value

This function returns a list of three items:

- \texttt{time} a numeric vector corresponding to the time axis.
- \texttt{freq} a numeric vector corresponding to the frequency axis.
- \texttt{amp} a numeric or a complex matrix corresponding to the amplitude values. Each column is a Fourier transform of length \( \text{wl}/2 \).
Note

The argument `fftw` can be used to try to speed up process time. When set to `TRUE`, the Fourier transform is computed through the function `FFT` of the package `fftw`. This package is a wrapper around the fastest Fourier transform of the free C subroutine library FFTW (http://www.fftw.org/). FFT should be then installed on your OS.

Note

This function is based on `fft`, `contour` and `filled.contour`

Author(s)

Jerome Sueur and Caroline Simonis.

References


See Also

ggspectro, spectro3D, dynspec, wf, oscillo, dBscale, fft.

Examples

```r
## Not run:
data(tico)
data(pellucens)
# simple plots
spectro(tico, f=22050)
spectro(tico, f=22050, osc=TRUE)
spectro(tico, f=22050, scale=FALSE)
spectro(tico, f=22050, osc=TRUE, scale=FALSE)
# change the dB scale by setting a different dB reference value (20microPa)
spectro(tico, f=22050, dBref=2*10^(-5))
# unnormalised spectrogram with a linear amplitude scale
spectro(tico, dB=NULL, norm=FALSE, scale=FALSE)
# manipulating wl
op<-par(mfrow=c(2,2))
spectro(tico, f=22050, wl=256, scale=FALSE)
title("wl = 256")
spectro(tico, f=22050, wl=512, scale=FALSE)
title("wl = 512")
spectro(tico, f=22050, wl=1024, scale=FALSE)
title("wl = 1024")
spectro(tico, f=22050, wl=4096, scale=FALSE)
title("wl = 4096")
par(op)
# vertical zoom using flim
spectro(tico, f=22050, flim=c(2,6))
spectro(tico, f=22050, flimd=c(2,6))
```
Description

This function returns a three-dimensional spectrographic representation of a time wave. The function corresponds to short-term Fourier transform.

Usage

```r
spectro3D(wave, f = NULL, wl = 512, wn = "hanning", zp = 0,
          ovlp = 0, norm = TRUE, correction = "none", fftw = FALSE,
          dB = "max0", dBref = NULL, plot = TRUE,
          magt = 10, magf = 10, maga = 2,
          palette = reverse.terrain.colors)
```

Arguments

- **wave**: an R object.
- **f**: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- **wl**: length of the window for the analysis (even number of points).
- **wn**: window name, see `ftwindow` (by default "hanning").
- **zp**: zero-padding (even number of points), see Details.
- **ovlp**: overlap between two successive windows (in %).
- **norm**: if TRUE the STFT is normalised (i.e. scaled) by its maximum.
correction  a character vector of length 1 to apply an amplitude ("amplitude") or an energy
("energy") correction to the FT window. This argument is useful only when one
wish to obtain absolute values that is when norm=FALSE, scaled=FALSE, and
pmf=FALSE. By default no correction is applied ("none").

fftw  if TRUE calls the function FFT of the library fftw. See Notes of the spectro.

dB  a character string specifying the type dB to return: "max0" for a maximum dB
value at 0, "A", "B", "C" and "D" for common dB weights.

dBref  a dB reference value when dB is TRUE. NULL by default but should be set to
2*10e-5 for a 20 microPa reference.

plot  logical, if TRUE plots the spectrogram (by default TRUE).

magt  magnification of the time axis.

magf  magnification of the frequency axis.

maga  magnification of the amplitude axis.

palette  a color palette function to be used to assign colors in the plot, see Details.

Details

Following Heisenberg uncertainty principle, the short-term Fourier transform cannot be precised in
both time and frequency. The temporal and frequency precisions of the function are actually depen-
dent of the wl value. Choosing a high wl value will increase the frequency resolution but reduce
the temporal one, and vice versa. The frequency precision is obtained by calculating the ratio f/wl,
and the temporal precision is obtained by calculating the reverse ratio wl/f. This problem can be
reduced in some way with zp that adds 0 values on both sides of the analysis window. This increases
frequency resolution without altering time resolution.

Any colour palette can be used. In particular, it is possible to use other palettes coming with
seeewave: reverse.gray.colors.1, reverse.gray.colors.2, spectro.colors, temp.colors,
reverse.heat.colors, reverse.cm.colors, reverse.topo.colors, corresponding to the re-
verse of heat.colors,topo.colors, cm.colors.

Use magt, magf and maga to resize the plot.

Value

This function returns a list of three items:

time  a numeric vector corresponding to the time axis.

freq  a numeric vector corresponding to the frequency axis.

amp  a numeric matrix corresponding to the amplitude values. Each column is a
Fourier transform of length wl/2.

Note

This function requires rgl and is based on fft. See examples of spectro for analysis arguments
(wl,zp, ovlp).

Author(s)

Jerome Sueur <sueur@mnhn.fr> and Caroline Simonis <csimonis@mnhn.fr>.
**squarefilter**

**See Also**

`spectro`, `dynspec`, `wf`, `fft`.

**Examples**

```r
## Not run:
require(rgl)
data(tico)
spectro3D(tico, f=22050, wl=512, ovlp=75, zp=16, maga=4, palette=reverse.terrain.colors)
# linear amplitude scale without a normalisation of the STFT matrix
# time and frequency scales need to be dramatically amplified
spectro3D(tico, norm=FALSE, dB=NULL, magt=100000, magf=100000)

## End(Not run)
```

---

**Description**

This function prepares the amplitude profile of a square frequency filter.

**Usage**

```r
squarefilter(f, from = NULL, to = NULL, bandpass = TRUE, wl = 1024)
```

**Arguments**

- `f` a numeric vector of length 1 for the sampling frequency of the object to be filtered (in Hz).
- `from` a numeric vector for the start frequencies (in Hz) where to apply the filter.
- `to` a numeric vector of the end frequencies (in Hz) where to apply the filter.
- `bandpass` if TRUE a band-pass filter is prepared between start and end frequencies (arguments `from` and `to`), if FALSE a bandstop filter is prepared.
- `wl` window length of the impulse filter (even number of points).

**Value**

The function returns a two-column matrix, the first column is the frequency in kHz and the second column is the amplitude of the filter (frequency response of the filter).

**Note**

This function can be used to prepare bandpass or bandstop filters to be used with `fir` and `ffilter`. See examples.
Author(s)
Laurent Lellouch

See Also
fir, drawfilter, ffilter, combfilter, bwfilter

Examples
f <- 44100
a <- noisew(f = f, d = 1)
p <- squarefilter(f, from = c(100, 1000, 4000), to = c(500, 3000, 8000))
plot(p, type="l")
h <- fir(a, f = f, custom = p, wl = 1024, output = 'Wave')
spectro(h)

Description
This function performs the short-term Fourier transform externally, i.e. outside R, on an audio .wav file

Usage
stft.ext(file, wl = 512, ovlp = 0, mean = FALSE, norm = FALSE, dB = FALSE, verbose = FALSE)

Arguments
file a character string vector of length 1, corresponding to the audio file name. Only .wav files are supported.
wl a numeric vector of length 1, window length for the analysis (even number of points) (by default = 512).
ovlp a numeric vector of length 1, overlap between two successive windows (in %).
mean logical, if TRUE calculates the mean spectrum.
norm logical, if TRUE the spectrogram matrix and the mean spectrum if mean is TRUE is/are normalized to a maximum of 1.
dB logical, if TRUE the values are in deciBel with a maximum of 0.
verbose logical, if TRUE displays audio file meta-information.
Value

If \texttt{mean} is \texttt{FALSE}, the function returns a two-column matrix corresponding to the amplitude values of the spectrogram. Each column is a discrete Fourier transform of length \(w_l\).

If \texttt{mean} is \texttt{TRUE}, the function returns a list with two elements:

\begin{itemize}
  \item \texttt{mean} a vector of length \(w_l/2\) corresponding to the mean spectrum
  \item \texttt{amp} the spectrogram matrix.
\end{itemize}

Note

This function was developed to speed up the process of computing a spectrogram or a mean spectrum. The function should be faster than \texttt{spectro} and \texttt{meanspec} respectively as it does not import the .wav file into R and uses \texttt{fftw3} (\url{http://www.fftw.org/}) and \texttt{libsndfile} (\url{http://www.mega-nerd.com/libsndfile/}) libraries. The function is typically made for batch processing or long files. Gain in process time is significant for files > 10 minutes and for more than 1000 files lasting one second.

A Hanning window shape is used by default.

Author(s)

Alicia Stotz

See Also

\texttt{spectro, meanspec, fft}.

Examples

\begin{verbatim}
  # tico data
data(tico)
  # write a local .wav file
  savewav(tico, file="tico.wav")
  # spectrogram - not normalised - linear scale - meta-information returned
  res <- stft.ext(file="tico.wav", verbose=TRUE)
  # spectrogram - normalised - linear scale - no meta-information
  res <- stft.ext(file="tico.wav", norm=TRUE)
  # spectrogram - dB scale - no meta-information
  res <- stft.ext(file="tico.wav", db=TRUE)
  # see how it looks like (no scale)
  filled.contour(t(res))
  # spectrogram and mean spectrum - normalised - linear scale
  res <- stft.ext(file="tico.wav", norm = TRUE, mean = TRUE)
  # remove .wav file
  unlink("tico.wav")
\end{verbatim}
**Description**

This function analyses one or two sequences of symbols from numeric (time) series.

**Usage**

```r
symba(x, y = NULL, symb = 5, collapse = TRUE, entropy = "abs", plot = FALSE, type = "l", lty1 = 1, lty2 = 2, col1 = 2, col2 = 4, cex1 = 0.75, cex2 = 0.75, xlab = "index", ylab = "Amplitude", legend=TRUE, ...)
```

**Arguments**

- `x`: a first R object.
- `y`: a second R object.
- `symb`: the number of symbols used for the discretisation, can be set to 3 or 5 only.
- `collapse`: logical, if `TRUE`, the symbols are pasted in a character string of length 1.
- `entropy`: either "abs" for an absolute value or "rel" for a relative value, i.e. between 0 and 1.
- `plot`: logical, if `TRUE` plots the series `x` (and `y`) and the respective symbols.
- `type`: if `plot` is `TRUE`, type of plot that should be drawn. See `plot` for details (by default "l" for lines).
- `lty1`: line type of the object `x` if `type="l".`
- `lty2`: line type of the object `y` if `type="l".`
- `col1`: colour of the object `x`.
- `col2`: colour of the object `y`.
- `cex1`: character size of `x` symbols.
- `cex2`: character size of `y` symbols.
- `xlab`: title of the x axis.
- `ylab`: title of the y axis.
- `legend`: logical, if `TRUE` and if `y` is not `NULL` adds a legend to the plot.
- `...`: other `plot` graphical parameters.

**Details**

The analysis consists in transforming the series into a sequence of symbols (see the function `discrets`) and in computing the absolute frequency of each symbol within the sequence. The entropy ($H$) is then calculated using the symbol frequencies. Using the argument `entropy`, the entropy can be expressed along an absolute scale or as a relative value varying between 0 and 1.

If two numeric (time) series are provided (`x` and `y`) the absolute symbol frequencies and entropy of
each series is returned. Besides the mutual information ($I$) is estimated according to:

$$I = H_x + H_y - H_{xy}$$

with $H_x$ the entropy of $x$ symbol series, $H_y$ the entropy of $y$ symbol series, and $H_{xy}$ the joint entropy of $x$ and $y$ symbol series.

**Value**

If $y$ is NULL a list of three items is returned ($s_1$, freq1, h1).
If $y$ is not NULL, a list of 6 items is returned ($s_1$, freq1, h1, s2, freq2, h2, I):

- $s_1$ the sequence of symbols of $x$,
- freq1 the relative frequency of each $x$ symbol,
- h1 the entropy of $x$ symbol sequence,
- s2 the sequence of symbols of $y$,
- freq2 the relative frequency of each $y$ symbol,
- h2 the entropy of $y$ symbol sequence,
- I the mutual information between $x$ and $y$.

**Note**

It might be useful to round the values of the input series (see examples). The mutual information ($I$) should increase with the similarity between the series to compare ($x$ and $y$).

**Author(s)**

Jerome Sueur <sueur@mnhn.fr>

**References**


**See Also**

discrets, SAX

**Examples**

```r
# analysis of a frequency spectrum
data(tico)
spec1<-spec(tico,f=22050,at=0.2,plot=FALSE)
symba(spec1[,2],plot=TRUE)
# it might be better to round the values
symba(round(spec1[,2],2),plot=TRUE)
# in that case the symbol entropy is close to the spectral entropy
symba(round(spec1[,2],2),entrop="rel")$h1
sh(spec1)
```
# to compare two frequency spectra
spec2<-spec(tico,f=22050,wi=512,at=1.1,plot=FALSE)
symba(round(spec1[,2],2),round(spec2[,2],2),plot=TRUE)

---

**synth**  
*Synthesis of time wave (additive model)*

## Description

This function synthesizes pure or harmonic tone sounds with amplitude modulation (am) and/or frequency modulation (fm).

## Usage

```r
synth(f, d, cf, a = 1, signal = "sine", shape = NULL, p = 0,  
am = c(0, 0, 0), fm = c(0, 0, 0, 0, 0), harmonics = 1,  
plot = FALSE, listen = FALSE, output = "matrix", ...)
```

## Arguments

- `f` sampling frequency (in Hz).
- `d` duration (in s).
- `cf` carrier frequency (in Hz).
- `a` amplitude (linear scale, relative when adding different waves).
- `signal` a character vector specifying the shape of the signal, see details.
- `shape` modification of the whole amplitude shape of the wave, see details.
- `p` phase (in radians).
- `am` a numeric vector of length 3 describing amplitude modulation parameters, see details.
- `fm` a numeric vector of length 5 describing frequency modulation parameters, see details.
- `harmonics` a numeric specifying the number and the relative amplitude of harmonics, see details.
- `plot` if TRUE returns the spectrogram of the synthesized sound (by default FALSE).
- `listen` if TRUE the new sound is played back.
- `output` character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
- `...` other spectro graphical parameters.
synth

Details

• signal is a character vector of length 1 that specifies the function used to synthesize the signal. There are three options:
  1. "sine": for a sinus function
  2. "tria": for a triangle function
  3. "square": for a square function
  4. "saw": for a square function

• shape is a character vector of length 1 that allows to modify the whole amplitude shape of the wave. There are four options:
  1. "incr": linear increase
  2. "decr": linear decrease
  3. "sine": sinusoid-like shape
  4. "tria": triangular shape

• am is a numeric vector of length 3 including:
  1. the amplitude modulation depth (in %)
  2. the frequency of the amplitude modulation (in Hz),
  3. the phase of the amplitude modulation (in radian).

• fm is a numeric vector of length 5 including:
  1. the maximum excursion of a sinusoidal frequency modulation (in Hz),
  2. the frequency of a sinusoidal frequency modulation (in Hz),
  3. the maximum excursion of a linear frequency modulation (in Hz).
  4. the phase of the frequency modulation (in radian).
  5. the maximum excursion of an exponential frequency modulation (in Hz).

• harmonics is a numeric vector that controls the number and the relative amplitude of harmonics synthesized.
  By default harmonics = 1 meaning that a pure tone made of a single harmonic (fundamental) will be produced.
  To produce harmonics, the length of harmonics has to be greater than 1. The length of harmonics will set the number of harmonics, including the first one (fundamental). The value of each element of harmonics specify the relative amplitude of each harmonic. The first value must equal to 1.
  Here are some examples:
  - harmonics = c(1, 0.5, 0.25) will produce a sound with three harmonics (fundamental + 2 harmonics), the second harmonic having an amplitude half the fundamental amplitude and the second harmonic an amplitude a quarter of the fundamental amplitude.
  - harmonics = c(1, 0, 0.25) will produce a sound with two harmonics (fundamental + 1 harmonic) the second harmonic having a null relative amplitude.
  - harmonics = rep(1,4) will produce a sound with four harmonics (fundamental + 3 harmonics) of equal amplitude.

Value

If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.
Author(s)
Jerome Sueur and Laurent Lellouch.

References

See Also
`synth2`, `noisew`, `pulse`, `echo`

Examples
```
## You can use plot=TRUE and spectro() options
## to directly 'see' the new-built sounds
f <- 8000  # sampling frequency
d <- 1     # duration (1 s)
cf <- 440  # carrier frequency (440 Hz, i.e. flat A tone)
# pure sinusoidal tone
s <- synth(f=f, d=d, cf=cf)
# pure triangular tone
s <- synth(f=f, d=d, cf=cf, signal="tria")
# pure tone with triangle overall shape
s <- synth(f=f, d=d, cf=cf, shape="tria")
# pure tones with am
s <- synth(f=f, d=d, cf=cf, am=c(50, 10))
# pure tones with am
# and phase shift of pi radian (180 degrees)
s <- synth(f=f, d=d, cf=cf, am=c(50, 10, pi))
# pure tone with +1000 Hz linear fm
s <- synth(f=f, d=d, cf=cf, fm=c(0, 0, 1000, 0, 0))
# pure tone with sinusoidal fm
# (maximum excursion of 250 Hz, frequency of 10 Hz)
s <- synth(f=f, d=d, cf=cf, fm=c(250, 10, 0, 0, 0))
# pure tone with sinusoidal fm
# (maximum excursion of 250 Hz, frequency of 10 Hz,
# phase shift of pi radian (180 degrees))
s <- synth(f=f, d=d, cf=cf, fm=c(250, 10, 0, pi, 0))
# pure tone with sinusoidal am
# (maximum excursion of 250 Hz, frequency of 10 Hz)
# and linear fm (maximum excursion of 500 Hz)
s <- synth(f=f, d=d, cf=cf, fm=c(250, 10, 500, 0, 0, 0))
# the same with am
s <- synth(f=f, d=d, cf=cf, am=c(50, 10), fm=c(250, 10, 250, 0, 0))
# the same with am and a triangular overall shape
s <- synth(f=f, d=d, cf=cf, shape="tria", am=c(50, 10), fm=c(250, 10, 250, 0, 0))
# an harmonic sound
s <- synth(f=f, d=d, cf=cf, harmonics=c(1, 0.5, 0.25))
# a clarinet-like sound
clarinet <- c(1, 0, 0.5, 0, 0.14, 0, 0.5, 0, 0.12, 0, 0.17)
s <- synth(f=f, d=d, cf = 235.5, harmonics=clarinet)
# inharmonic FM sound built 'manually'
```
Description

This function synthesizes pure tone sound based on an amplitude envelope and an instantaneous frequency contour. The function can also be used to modify a reference sound.

Usage

```r
synth2(env = NULL, ifreq, f, plot = FALSE, listen = FALSE, output = "matrix", ...)
```

Arguments

- `env`: a numeric vector describing the amplitude envelope (i.e. the amplitude modulation). By default NULL, generating a squared envelope.
- `ifreq`: a numeric vector describing the instantaneous frequency (in Hz).
- `f`: a numeric vector for the sampling frequency (in Hz)
- `plot`: if TRUE returns the spectrogram of the synthesized sound (by default FALSE).
- `listen`: if TRUE the new sound is played back.
- `output`: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
- `...`: other `spectro` graphical parameters.

Details

`env` and `ifreq` must have exactly the same length.

The amplitude envelope can be obtained with the Hilbert envelope (function `env`) and the instantaneous frequency can be obtained with the Hilbert transform (function `ifreq`). This opens a great variety of signal modifications as shown in the example section.

Value

If `plot` is FALSE, a new wave is returned. The class of the returned object is set with the argument `output`.

Author(s)

Jérôme Sueur and Laurent Lellouch
Temporal entropy

Compute the entropy of a temporal envelope.
Usage

\texttt{th(env, breaks)}

Arguments

\begin{itemize}
  \item \texttt{env} a data set resulting of an envelope obtained using \texttt{env}
  \item \texttt{breaks} 'breaks' argument of \texttt{hist} to compute the entropy on the distribution obtained with an histogram.
\end{itemize}

Details

Temporal entropy is calculated according to:

\[ S = -\sum_{i=1}^{N} \frac{y_i \log_2(y_i)}{\log_2(N)} \]

with:
\( y = \) relative amplitude of the \( i \) envelope point,
and
\[ \sum_{i=1}^{N} y_i = 1 \]

and \( N = \) number of envelope points.

Value

A single value varying between 0 and 1 is returned. The value has no unit.

Note

The temporal entropy of a noisy signal with many amplitude modulations will tend towards 1 whereas the temporal entropy of quiet signal will tend towards 0.
Note, however, that a sustained sound with an almost flat envelope will also show a very high temporal entropy except if you compute the entropy on the distribution obtained with the histogram. See examples.

Author(s)

Jerome Sueur, George Zhan for the idea and implementation of the argument \texttt{breaks}.

See Also

\texttt{sh}, \texttt{csh}, \texttt{H}
Examples

```r
# Temporal entropy of a cicada song
data(orni)
envorni<-env(orni,f=22050,plot=FALSE)
th(envorni)
# Smoothing the envelope might slightly change the result.
envorniS<-env(orni,f=22050,smooth=c(50,0),plot=FALSE)
th(envorniS)
# If we mute a part of the cicada song, the temporal entropy decreases
orni2<-mute(orni,f=22050,from=0.3,to=0.55,plot=FALSE)
envorni2<-env(orni2,f=22050,plot=FALSE)
th(envorni2)
# The temporal entropy of noise tends towards 1
a<-noise(d=1,f=8000)
enva<-env(a,f=8000,plot=FALSE)
th(enva)
# But be aware that the temporal entropy of a sustained sound also tends towards 1
b<-synth(f=8000,d=1,cf=2000,plot=FALSE)
envb<-env(b,f=8000,plot=FALSE)
th(envb)
# except if you use the distribution of the histogram
th(envb, breaks="Sturges")
```

---

tico  

*Song of the bird *Zonotrichia capensis*

---

Description

Recording of a song emitted by a male of the neotropical sparrow *Zonotrichia capensis*.

Usage

data(tico)

Format

A Wave object.

Details

Duration = 1.795 s. Sampling frequency = 22050 hz.

Source

Recording by Thierry Aubin.
Examples

data(tico)
oscillo(tico,f=22050)

---

timer

*Time measurements of a time wave*

Description

This function computes and shows the duration of signal periods, pause periods and their ratio.

Usage

timer(wave, f, threshold = 5, dmin = NULL, envt = "abs",
      power = 1, msmooth = NULL, ksmooth = NULL,
      ssmooth = NULL, tlim = NULL, plot = TRUE, plotthreshold = TRUE,
      col = "black", colval = "red",
      xlab = "Time (s)", ylab = "Amplitude", ...)

Arguments

- **wave**: an R object.
- **f**: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
- **threshold**: amplitude threshold for signal detection (in %).
- **dmin**: time threshold (minimum duration) for signal detection (in s).
- **envt**: the type of envelope to be used: either "abs" for absolute amplitude envelope or "hil" for Hilbert amplitude envelope. See `env`.
- **power**: a power factor applied to the amplitude envelope. Increasing `power` will reduce low amplitude modulations and increase high amplitude modulations. This can be used to reduce background noise (by default equals to 1, i.e. no change).
- **msMOOTH**: a vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %). See `env`.
- **ksMOOTH**: kernel smooth for the amplitude envelope via `kernel`. See `env`.
- **ssMOOTH**: sum smooth for the amplitude envelope. See `env`.
- **tlim**: modifications of the time X-axis limits.
- **plot**: logical, if TRUE plots the envelope and the measurements (by default TRUE).
- **plotthreshold**: logical, if TRUE plots the threshold as a horizontal line on the graph (by default TRUE).
- **col**: colour of the envelope.
- **colVAL**: colour of plotted measurements.
- **xlab**: title of the x-axis.
- **ylab**: title of the y-axis.
- **...**: other `plot` graphical parameters.
Value

A list containing seven items:

- **s** duration of signal period(s) in seconds
- **p** duration of pause period(s) in seconds
- **r** ratio between the signal and silence periods(s)
- **positions** a list containing four elements:
  - **s.start** start position(s) of signal period(s)
  - **s.end** end position(s) of signal period(s)
  - **first** whether the first event detected is a pause or a signal

Warning

Setting to high values to `msmooth` or `ssmooth` might return unaccurate results. Double check your results if so.

Author(s)

Jerome Sueur

See Also

`env, cutw, pastew`

Examples

data(tico)
timer(tico, f=22050, threshold=5, msmooth=c(50, 0))
# to compare with an oscillographic representation
data(orni)
op<-par(mfrow=c(2, 1))
timer(orni, f=22050, threshold=5, msmooth=c(40, 0), tck=0.05,
   bty="l", colval="blue")
title(main="A cicada song made of five schemes", col="blue")
oscillo(orni, f=22050, k=1, j=1)
par(op)

TKEO  Teager-Kaiser energy tracking operator

Description

This function computes the Teager-Kaiser energy operator.
Usage

TKEO(wave, f, m = 1, M = 1, plot = TRUE,
  xlab = "Time (s)", ylab = "Energy",
  type = "l", bty = "l", ...)

Arguments

  wave    an R object.
  f       sampling frequency of wave (in Hz). Does not need to be specified if embedded
          in wave.
  m       a numeric vector of length 1 for the exponent parameter. See details.
  M       a numeric vector of length 1 for the lag parameter. See details.
  plot    logical, if TRUE returns a plot of the TK energy along time (by default TRUE).
  xlab    Label of time x-axis.
  ylab    Label of energy y-axis.
  type    if plot is TRUE, type of plot that should be drawn. See plot for details (by
          default "l" for lines).
  bty     the type of box to be drawn around the energy plot.
          ...
  ...     other plot graphical parameters.

Details

The discrete version of the Teager-Kaiser operator is computed according to:

\[ y_n = x_n^{2/m} - (x_{n-M} \times x_{n+M})^{1/m} \]

\[ , \]

with \( m \) the exponent parameter and \( M \) the lag parameter which both are usually equal to 1 for a
conventional operator.

The Teager-Kaiser operator can be used to track amplitude modulations (AM) and/or frequency
modulations (FM).

See examples.

Value

This function returns a two-column matrix, the first column is time and the second column includes
the successive energy values.

m/2 NA values are added at the start and end of the vector.

Author(s)

Jerome Sueur

References

operators*. University of Oslo, Department of Informatics, PhD Thesis, x + 100 p.
See Also
  
  env, ifreq.

Examples

```r
op <- par(mfrow=c(2,1))

## sinusoid AM
s1 <- synth(f=8000, d=0.1, cf=200, am=c(100,10), output="Wave")
oscillo(s1)
TKEO(s1)

## linear AM decrease
s2 <- synth(f=8000, d=0.1, cf=200, shape="decr", output="Wave")
oscillo(s2)
TKEO(s2)

## sinusoid FM
s3 <- synth(f=8000, d=0.1, cf=200, fm=c(150,50,0,0,0), output="Wave")
oscillo(s3)
TKEO(s3)

## linear FM increase
s4 <- synth(f=8000, d=0.1, cf=200, fm=c(0,0,600,0,0), output="Wave")
oscillo(s4)
TKEO(s4)

## AM and FM
s5 <- synth(f=8000, d=0.1, cf=200, am=c(100,10), fm=c(150,50,0,0,0), output="Wave")
oscillo(s5)
TKEO(s5)
par(op)
```

---

**wasp**  
*Wave length and Speed of sound*

**Description**

This function returns the wavelength and the speed of sound of a given frequency in air, fresh-water or sea-water.

**Usage**

```r
wasp(f, t = 20, c = NULL, s = NULL, d = NULL, medium = "air")
```

**Arguments**

- `f`  
  frequency (Hz).
- `t`  
  temperature (degree Celsius).
- `c`  
  celerity (m/s) if a wavelength is to be found at a particular speed of sound.
- `s`  
  salinity (parts per thousand) when medium is "sea".
- `d`  
  depth (m) when medium is "sea".
medium for sound propagation, either "air", "fresh" for fresh, or pure, water, "sea" for sea water.

Details

Speed of sound in air is computed according to:

\[ c = 331.4 + 0.6 \times t \]

Speed of sound in fresh-water is computed according to Marczak equation:

\[
\begin{align*}
    c &= 1.402385 \times 10^3 + 5.038813 \times t - 5.799136 \times 10^{-2} \times t^2 \\
    & \quad + 3.287156 \times 10^{-4} \times t^3 - 1.398845 \times 10^{-6} \times t^4 \\
    & \quad + 2.787860 \times 10^{-9} \times t^5 
\end{align*}
\]

with \( t = \) temperature in degrees Celsius; range of validity: 0-95 degrees Celcius at atmospheric pressure.

Speed of sound in sea-water is computed according to Mackenzie equation:

\[
\begin{align*}
    c &= 1448.96 + 4.591 \times t - 5.304 \times 10^{-2} \times t^2 \\
    & \quad + 2.374 \times 10^{-4} \times t^3 + 1.34 \times (s - 35) + 1.63 \times 10^{-2} \times d \\
    & \quad + 1.675 \times 10^{-7} \times d^2 - 1.025 \times 10^{-2} \times t \times (s - 35) \\
    & \quad - 7.139 \times 10^{-13} \times t \times d^3 
\end{align*}
\]

with \( t = \) temperature in degrees Celsius; \( s = \) salinity in parts per thousand; \( d = \) depth in meters; range of validity: temperature 2 to 30 degrees Celcius, salinity 25 to 40 parts per thousand, depth 0 to 8000 m.

Wavelength is obtained following:

\[ \lambda = \frac{c}{f} \]

with \( c = \) speed of sound in meters/second; \( f = \) frequency in Hertz.

Value

A list of two values is returned:

1. wavelength in meters
2. speed of sound in meters/second.

Author(s)

Jerome Sueur <sueur@mnhn.fr>
References

http://resource.npl.co.uk

Examples

```r
# wavelength (m) of a 2000 Hz air-borne sound at 20 degrees Celsius
wasp(f=2000)$1
# [1] 0.1717

# sound speed in sea at 0 and -500 m
# for a respective temperature of 22 degrees Celcius and 11 degrees Celcius
wasp(f=1000,s=30,d=c(0,500),t=c(22,11),medium="sea")$c
# [1] 1521.246 1495.414

# wavelength (m) of a 1000 Hz sound in a medium unspecified where c = 1497 m/s
wasp(f=1000,c=1497)$1
# [1] 1.497

# variation of wavelength according to frequency and air temperature
op<-par(bg="lightgrey")
a<-seq(1000,200000,by=100) ; na<-length(a)
b<-seq(-20,40,by=10) ; nb<-length(b)
res<-matrix(numeric(na*nb),nrow=na)
for(i in 1:nb) res[,i]<-wasp(a,t=b[i])$1
matplot(x=a,y=res,type="l",lty=1,col= spectro.colors(nb),
   xlab="Frequency (Hz)",ylab="Wavelength (m)"
title("Wavelength of air-borne sound at different temperatures (deg. C)")
legend(x=15000,y=0.3,c("-20","-10","0","10","20","30","40"),
   lty=1,col= spectro.colors(nb),bg="grey")
par(op)
```

**wav2flac**

**wav-flac file conversion**

Description

This function converts .wav files into .flac files and reversely

Usage

```r
wav2flac(file, reverse = FALSE, overwrite = FALSE, 
exename = NULL, path2exe = NULL)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>the .wav or .flac file to convert.</td>
</tr>
<tr>
<td>reverse</td>
<td>logical, if TRUE converts a .flac file into a .wav file.</td>
</tr>
<tr>
<td>overwrite</td>
<td>logical, if TRUE overwrites the file to convert.</td>
</tr>
</tbody>
</table>
**wav2flac**

**exename**

A character string specifying the name of the FLAC binary file. If NULL, the default name “flac” will be used for Linux OS and “flac.exe” for Windows OS.

**path2exe**

A character string giving the path to the FLAC binary file. If NULL, the default path "c:/Program Files/FLAC/" will be used for Windows OS.

**Details**

The function runs FLAC. FLAC has then to be installed first, if not the function will not work.

**Value**

A new file is created.

**Note**

FLAC must be installed to use this function but not to install the package seewave. Free Lossless Audio Codec (FLAC) is a file format by Josh Coalson for lossless audio data compression. FLAC reduces bandwidth and storage requirements without sacrificing the integrity of the audio source. Audio sources encoded to FLAC are typically reduced in size 40 to 50 percent.

**Author(s)**

Luis J. Villanueva-Rivera

**References**

FLAC website: [https://xiph.org/flac/](https://xiph.org/flac/)

**See Also**

savewav

**Examples**

```r
## Not run:
# synthesis of a 1kHz sound
a<-synth(d=10,f=8000,cf=1000)
# save it as a .wav file in the default working directory
savewav(a,f=8000)
# compress it to FLAC format and overwrite on the file a.wav
wav2flac("a.wav", overwrite=TRUE)
# back to .wav format
wav2flac("a.flac", reverse=TRUE)
# remove the files
unlink(c("a.wav","a.flac"))
## End(Not run)
```
Description

This function returns a waterfall display of a short-term Fourier transform or of any matrix.

Usage

```r
wf(wave, f, wl = 512, zp = 0, ovlp = 0, fftw= FALSE, dB = "max0",
  dBref = NULL, wn = "hanning", x = NULL,
  hoff = 1, voff = 1, col = heat.colors,
  xlab = "Frequency (kHz)", ylab = "Amplitude (dB)",
  xaxis = TRUE, yaxis = TRUE,
  density = NULL, border = NULL, lines = FALSE, lwd=NULL, ...)
```

Arguments

- `wave`: an R object.
- `f`: sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl`: window length for the analysis (even number of points). (by default = 512)
- `zp`: zero-padding (even number of points), see Details.
- `ovlp`: overlap between two successive windows (in %).
- `fftw`: if `true` calls the function `fft` of the library `fftw`. See Notes of the spectro.
- `dB`: a character string specifying the type `dB` to return: "max0" for a maximum `dB` value at 0, "A", "B", "C" and "D" for common `dB` weights.
- `dBref`: a `dB` reference value when `dB` is `true`. NULL by default but should be set to 2*10e-5 for a 20 microPa reference.
- `wn`: window name, see `ftwindow` (by default "hanning").
- `x`: a matrix if `wave` is not provided.
- `hoff`: horizontal 'offset' which shifts actual x-values slightly per row for visibility. Fractional parts will be removed.
- `voff`: vertical 'offset' which separates traces.
- `col`: a color or a color palette function to be used to assign colors in the plot
- `xlab`: title of the frequency x-axis.
- `ylab`: title of the amplitude y-axis.
- `xaxis`: a logical, if `TRUE` adds the frequency x-axis according to `f`.
- `yaxis`: a logical, if `TRUE` adds the amplitude y-axis according.
- `density`: argument of `polygon`: the density of shading lines, in lines per inch. The default value of 'NULL' means that no shading lines are drawn. A zero value of 'density' means no shading nor filling whereas negative values (and 'NA') suppress shading (and so allow color filling).
zapsilw

border argument of polygon: the color to draw the border. The default, 'NULL', means to use 'par("fg")'. Use 'border = NA' to omit borders.

lines a logical, if TRUE plots lines instead of surfaces (polygons).

lwd line width.

... other graphical arguments to passed to plot

Details

Data input can be either a time wave (wave) or a matrix (x). In that case, if xaxis is set to TRUE the x-axis will follow the row index. To change it, turn xaxis to FALSE and use axis afterwards. See examples.

Note

The function is well adapted to display short-term Fourier transform. However, any matrix can be called using the argument x instead of wave.

Author(s)

Carl G. Witthoft and Jerome Sueur <sueur@mnhn.fr>

See Also

spectro, spectro3D, dynspec

Examples

data(tico)
wf(tico, f=22050)
# changing the display parameters
jet.colors <- colorRampPalette(c("blue", "green"))
wf(tico, f=22050, hoff=0, voff=2, col=jet.colors, border = NA)
# matrix input instead of a time wave and transparent lines display
m <- numeric()
for(i in seq(-pi,pi,len=40)) {m <- cbind(m,10*(sin(seq(0,2*pi,len=100)+i)))}
wf(x=m, lines=TRUE, col="#0000FF50", xlab="Time", ylab="Amplitude", main="waterfall display")

zapsilw Zap silence periods of a time wave

Description

This function simply deletes the silence periods of a time wave.

Usage

zapsilw(wave, f, threshold = 5, plot = TRUE, output = "matrix", ...)
arguments

wave: an R object.
f: sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
threshold: amplitude threshold (in %) between silence and signal.
plot: logical, if TRUE plots the original and the new oscillograms (by default TRUE).
output: character string, the class of the object to return, either "matrix", "Wave", "Sample", "audioSample" or "ts".
... other oscillo graphical parameters.

value

If plot is FALSE, a new wave is returned. The class of the returned object is set with the argument output.

Note

Use the argument threshold to set the level of silence. See the examples.

Author(s)

Jerome Sueur <sueur@mnhn.fr>

See Also

afilter, oscillo

time

data(orni)
zapsilw(orni,f=22050,colwave="red")
# setting the threshold value
zapsilw(orni,f=22050,threshold=1)

---

Description

This function measures the period of a full oscillating cycle.

Usage

zc(wave, f, plot = TRUE, interpol = 1, threshold = NULL,
  xlab = "Time (s)", ylab = "Frequency (kHz)", ylim = c(0, f/2000),
  warning = TRUE, ...)

Instantaneous frequency of a time wave by zero-crossing
zc

Arguments

wave an R object.
f sampling frequency of wave (in Hz). Does not need to be specified if embedded in wave.
plot logical, if TRUE plots the dominant frequency along the time wave (by default TRUE).
interpol a numeric vector of length 1, interpolation factor.
threshold amplitude threshold for signal detection (in %).
xlab title of the x axis.
ylab title of the y axis.
ylim the range of y values.
warning a logical to specify if warning message should be displayed or not when interpol is > 100.
... other plot graphical parameters.

Details

If plot is FALSE, zc returns a vector of numeric data with the instantaneous frequency.

Value

If plot is FALSE, zc returns a two-column matrix, the first column corresponding to time in seconds (x-axis) and the second column corresponding to the instantaneous frequency of the time wave in kHz (y-axis).
‘NA’s correspond either to pause periods (e.g. detected applying threshold) or sections of the time wave not crossing the zero line. To remove ‘NA’s with na.omit allows to get only instantaneous frequency values but discards information about pause sections.

Note

interpol adds points to the time wave by linear interpolation (through approx). This increases measurement precision but as well time process. Type argument of plot cannot be set to “l”.

Author(s)

Jerome Sueur <sueur@mnhn.fr>, Caroline Simonis and Thierry Aubin

References


See Also

zc, ifreq
Examples

```r
data(pellucens)
pellu1 <- cutw(pellucens,f=22050,from=0,to=1,plot=FALSE)
# without interpolation
zc(pellu1,f=22050,threshold=5,pch=20)
# with interpolation
zc(pellu1,f=22050,threshold=5,interpol=20,pch=20)
# a way to plot with a line and to filter low frequencies
pellu2 <- zc(pellu1,f=22050,threshold=5,interpol=20,plot=FALSE)
pellu3 <- na.omit(pellu2[,2])
pellu4 <- pellu3[pellu3>3]
plot(x=seq(0,nrow(pellu1))/22050,length.out=length(pellu4)),
y=pellu4,type="l",xlab="Time(s)",ylab="Frequency(kHz)")
```

---

### zcr

**Zero-crossing rate**

**Description**

This function computes the zero-crossing rate of a time function, i.e. the average number the sign of a time wave changes.

**Usage**

```r
zcr(wave, f, wl = 512, ovlp = 0, plot = TRUE, type = "o", xlab = "Time (s)", ylab = "Zero crossing rate", ...)
```

**Arguments**

- `wave` an R object.
- `f` sampling frequency of `wave` (in Hz). Does not need to be specified if embedded in `wave`.
- `wl` length of the window for the analysis (even number of points, by default = 512). If `NULL` the zero-crossing rate is computed of the complete signal.
- `ovlp` overlap between two successive analysis windows (in %) if `wl` is not `NULL`.
- `plot` a logical, if `TRUE` plots a the zero-crossing rate results along time.
- `type` if `plot` is `TRUE`, type of plot that should be drawn. See `plot` for details (by default "l" for lines).
- `xlab` if `plot` is `TRUE`, label of the x axis.
- `ylab` if `plot` is `TRUE`, label of the y axis.
- `...` other `plot` graphical parameters.
Details

The zero-crossing rate is computed according to:

\[ zcr = \frac{1}{2 \times N} \sum_{t=0}^{N-1} |\text{sgn}[x(t+1)] - \text{sgn}[x(t)]| \]

with:
- \(N\) the length of the signal \(x\)
- \(\text{sgn}[x(t)] = 1\) if \(x(t) \geq 0\)
- \(\text{sgn}[x(t)] = -1\) if \(x(t) < 0\)

Value

There are two possibilities:

1. A numeric vector of length 1 if \(w_l\) is NULL,
2. A numeric two-column matrix is returned with the first column being time (s) and the second column being the zero-crossing rate (no scale) if \(w_l\) is not NULL.

Note

There are two possibilities:

1. If \(w_l\) is NULL then the zero-crossing rate is computed for the complete signal.
2. If \(w_l\) is not NULL then the zero-crossing rate is computed for a window sliding along the time wave.

The ZCR is supposed to help in detection of voiced/unvoiced sound sections.

Author(s)

Jerome Sueur

References


See Also

zc
Examples

data(tico)
## a single value for the complete signal, no plot
zcr(tico, w=0)
## a series of values computed for a sliding window of 512 samples, plot
zcr(tico)
Index

*Topic **IO**
  export, 67
  ftwindow, 81
  savewav, 141
  sox, 159
  wav2flac, 188
*Topic **datagen**
  drawenv, 54
  echo, 64
  noise2wav, 113
  pulse2wav, 129
  set2env, 147
  synth, 176
  synth2, 179
*Topic **datasets**
  orni, 116
  peewit, 123
  pellucens, 123
  sheep, 151
  tico, 182
*Topic **data**
  attenuation, 15
  songmeter, 155
*Topic **distribution**
  diffcumspec, 46
  itakura.dist, 92
  kl.dist, 93
  ks.dist, 94
  logspec.dist, 100
*Topic **dplot**
  ama, 12
  autoc, 16
  cccoh, 19
  ceps, 21
  cepstrogram, 23
  coh, 25
  corenv, 29
  corrspec, 30
  covspectrum, 33
  cutw, 39
dBscale, 40
delete, 43
dfreq, 44
diffenv, 48
diffspectrogram, 49
dynoscillo, 58
dynspec, 59
dynspectrogram, 61
dynspec1, 61
eenv, 65
fadew, 68
fbands, 69
fma, 77
fpeaks, 79
fund, 83
ggspectrogram, 84
ifreq, 88
localpeaks, 98
meanspec, 104
mutew, 111
oscillo, 117
oscilloST, 120
pastew, 121
phasew, 124
phasew2, 125
Q, 130
daver, 132
resapse, 133
rev, 134
rnoffset, 137
seedata, 145
seewave, 146
simspec, 152
soundscapespec, 157
spec, 160
specprop, 162
spectrogram, 165
spectro3D, 169
timer, 183
wf, 190
zc, 192

*Topic filter
afilter, 9
bwfilter, 17
combfilter, 26
drawfilter, 56
ffilter, 73
fir, 76
preemphasis, 128
squarefilter, 171

*Topic index
AR, 13
M, 101

*Topic input
songmeter, 155

*Topic maths
notefreq, 114
octaves, 115

*Topic math
convSPL, 28
dBweight, 41
dfoppler, 71
meandB, 103
mel, 106
melfilterbank, 108
micsens, 109
moredB, 110
sddb, 144
wasp, 186

*Topic ts
ACI, 4
acoustat, 6
afilter, 9
akamatsu, 10
ama, 12
AR, 13
autoc, 16
bwfilter, 17
ccoh, 19
ceps, 21
cepsstro, 23
coh, 25
combfilter, 26
corenv, 29
corspec, 30
covspectro, 33
crest, 35
csh, 36
cutspec, 37
cutw, 39
dBscale, 40
dBweight, 41
deletew, 43
dfreq, 44
difftumspec, 46
diffenv, 48
diffspec, 49
diffwave, 51
disccrets, 53
drawenv, 54
drawfilter, 56
duration, 57
dynoscillo, 58
dynspec, 59
dynspectro, 61
echo, 64
eenv, 65
fadew, 68
fbands, 69
ffilter, 73
field, 74
fir, 76
fma, 77
fpeaks, 79
ftwindow, 81
fund, 83
ggspectro, 84
H, 86
hilbert, 87
ifreq, 88
istft, 90
itakura.dist, 92
kl.dist, 93
ks.dist, 94
lfs, 96
listen, 97
localpeaks, 98
logspectro, 100
M, 101
meanspec, 104
melfilterbank, 108
mutew, 111
NDSI, 112
noisew, 113
oscillo, 117
oscilloST, 120
pastew, 121
phaseplot, 124
phaseplot2, 125
playlist, 127
preemphasis, 128
pulsew, 129
Q, 130
repw, 132
resamp, 133
reww, 134
rmam, 135
rmnoise, 136
rmoffset, 137
rms, 138
roughness, 139
rugo, 140
SAX, 143
seewave, 146
setenv, 147
sfm, 148
sh, 149
simspec, 152
smoothw, 154
soundscapespec, 157
spec, 160
specprop, 162
spectro, 165
spectro3D, 169
squarefilter, 171
stft.ext, 172
symba, 174
synth, 176
synth2, 179
th, 180
timer, 183
TKEO, 184
wf, 190
zapsilw, 191
zc, 192
zcr, 194

acf, 17
ACI, 4
acoustat, 6
addsilw, 8, 39, 44, 68, 112, 122, 132, 134
afilter, 9, 18, 74, 77, 129, 137, 192
akamatsu, 10
ama, 12, 78

approx, 193
AR, 13, 102
attenuation, 15, 28
autoc, 10, 16, 23, 25, 84
axis, 40, 191

barplot, 70, 158
bwidth, 17, 18, 27, 74, 77, 129, 172
ccoh, 19, 26
cps, 17, 21, 24, 25, 84
cpestro, 23, 23, 84
coh, 21, 25
comfilter, 26, 27, 56, 74, 129, 172
contour, 20, 24, 167, 168
convolve, 64, 76
cvsp, 16, 28, 42, 103, 110, 145
cor, 29–34
cor.test, 30–32
corenv, 29, 34, 49
correspec, 30, 30, 32, 34, 51, 105, 131, 153, 162
covspectro, 30, 32, 33, 82
crest, 35
csh, 10, 36, 87, 149, 151, 181
cutspec, 37
cutw, 9, 39, 44, 68, 112, 119, 122, 132, 184
dbscale, 40, 168
dbweight, 28, 41, 103, 110, 145
deltew, 9, 39, 43, 68, 112, 122, 132, 134
dfreq, 10, 44, 82
diffcumspec, 46, 51, 96
diffenv, 48, 51, 52, 153
diffspec, 47, 49, 49, 52, 93, 94, 96, 101, 105, 153
diffwave, 49, 51
dir, 13
discr, 53, 144, 174, 175
drawenv, 54, 56, 148
drawfilter, 27, 56, 172
duration, 57
dynoscillo, 58, 119, 121
dynspec, 58, 59, 63, 85, 105, 162, 168, 171, 191
dynspectro, 61, 61
echo, 64, 178, 180
einv, 13, 14, 29, 30, 48, 49, 52, 55, 60–63, 65, 86, 102, 147, 148, 179, 181, 183, 184, 186
INDEX

export, 67, 142
fadew, 9, 39, 44, 68, 112, 122, 132, 134
fbands, 69
fdoppler, 71
ffilter, 18, 27, 56, 73, 77, 91, 97, 129, 171, 172
fft, 22, 45, 61, 63, 76, 91, 105, 131, 161, 162, 168, 170, 171, 173
field, 74
filled.contour, 20, 40, 63, 168
filter, 155
fir, 27, 56, 74, 76, 129, 155, 171, 172
fma, 13, 77
fpeaks, 22, 79, 99, 100, 105, 161, 162
ftwindow, 5, 6, 33, 36, 44, 59, 62, 73, 76, 81, 90, 97, 104, 157, 160, 166, 169, 190
fund, 23, 25, 83
ggspectro, 84, 168

H, 86, 140, 141, 151, 181
hilbert, 66, 78, 87, 89, 90, 136
hist, 181

ifreq, 78, 88, 88, 179, 186, 193
image, 63
IQR, 163
istft, 73, 90, 97
itakura.dist, 47, 51, 92, 96, 101, 153

kernel, 29, 48, 52, 60, 62, 65, 86, 147, 183
kl.dist, 47, 51, 93, 96, 101, 153
ks.dist, 47, 51, 93, 94, 96, 101, 153
ks.test, 95

lfs, 18, 74, 77, 91, 96, 129
listen, 97, 127
localpeaks, 80, 98, 105, 162
locator, 20, 55, 167
log, 93
logspec.dist, 47, 51, 93, 94, 96, 100, 153

M, 14, 101
mean, 139, 140, 163
meandB, 103, 145
meanspec, 7, 12, 13, 31, 32, 38, 45, 46, 50, 51, 69, 70, 73, 76, 79, 80, 82, 92, 93, 95, 99, 100, 104, 130, 131, 148, 150, 152, 153, 158, 162, 163, 173
median, 163
mel, 106, 109
melfilterbank, 107, 108
micsens, 109
Mod, 66
moredB, 16, 28, 42, 103, 110, 145
mutew, 9, 39, 44, 68, 111, 122, 132, 134

na.omit, 193
NDSI, 112, 113, 158
noisew, 113, 129, 137, 178, 180
normalize, 142
notefreq, 114, 116

octaves, 115, 115
orni, 116
oscilloST, 58, 119, 120

par, 118
paste, 143
pastew, 9, 39, 44, 68, 112, 119, 121, 132, 134, 184
peewit, 123
pellucens, 123
phaseplot, 124, 126
phaseplot2, 125, 125
play, 98, 127
playlist, 127
plot, 6, 10, 12, 15, 16, 22, 25, 29, 31, 34, 36, 45, 46, 48, 50, 58, 60, 69, 79, 83, 89, 95, 99, 105, 124, 126, 129, 130, 153, 161, 163, 174, 183, 185, 191, 193, 194
polygon, 190, 191
preemphasis, 18, 74, 77, 128
pulse, 114, 178, 180
pulsw, 129

Q, 130
quantile, 163

repw, 122, 132
resamp, 133
revw, 9, 39, 44, 68, 112, 122, 132, 134
rmam, 135
INDEX

rmnoise, 136
rmoffset, 137
rms, 35, 36, 138, 140, 141
rnorm, 113
roughness, 139, 141
round, 79
rugo, 140, 140
runif, 113
savewav, 141, 189
SAX, 113, 143, 158, 175
sd, 163
sddb, 103, 144
seedata, 145
seewave, 146
seewave-package (seewave), 146
setenv, 55, 147
sfm, 38, 148, 151, 164
sh, 37, 38, 86, 87, 140, 141, 149, 149, 164, 181
sheep, 151
simspec, 47, 51, 93, 94, 96, 101, 105, 152
smooth.spline, 136
smoothw, 154
songmeter, 155
soundscapespec, 112, 113, 144, 157
sox, 159
spec, 31, 32, 38, 45, 46, 50, 51, 61, 63, 69, 70, 73, 76, 78–80, 82, 92, 93, 95, 99, 100, 105, 130, 131, 148, 150, 152, 153, 158, 160, 163
spec.pgram, 20, 21, 26
specprop, 7, 162
spectro, 5, 20, 21, 24, 26, 27, 34, 40, 41, 45, 61, 63, 73, 82, 84, 85, 91, 97, 128, 160, 165, 170, 171, 173, 176, 179, 191
spectro3D, 61, 63, 82, 85, 168, 169, 191
squarefilter, 27, 56, 171
stft.ext, 172
strptime, 157
symba, 54, 144, 174
synth, 55, 65, 114, 129, 148, 176
synth2, 178, 179, 180
th, 14, 37, 86, 87, 140, 141, 151, 180
tico, 182
timer, 10, 119, 183
TKEO, 184
tkeo (TKEO), 184
wasp, 72, 186
wav2flac, 188
Wave, 53, 142
wf, 61, 63, 168, 171, 190
write.table, 67
writeWave, 142
zapsilw, 9, 39, 44, 68, 112, 122, 132, 191
zc, 10, 90, 192, 193, 195
zc, 194