

Package ‘tuneR’

November 19, 2009

Version 0.2-12

Date 2009-11-18

Title Analysis of music

Author Uwe Ligges <ligges@statistik.tu-dortmund.de> with contributions from Andrea Preusser and Claus Weihs, as well as code fragments and ideas from the former package ‘sound’ by Matthias Heymann.

Maintainer Uwe Ligges <ligges@statistik.tu-dortmund.de>

Depends R (>= 2.5.0), methods

Suggests pastecs

Description Collection of tools to analyze music, handling wave files, transcription, ...

License GPL-2

Repository CRAN

Date/Publication 2009-11-19 13:01:00

R topics documented:

Arith-methods	2
bind	3
channel	3
downsample	4
equalWave	5
extractWave	5
FF	7
length	8
lilyinput	9
melodyplot	10
MFCC	12
Mono-Stereo	14
normalize	15

noSilence	16
noteFromFF	17
notenames	18
panorama	18
periodogram-methods	19
play-methods	22
plot	23
plot-Wave	23
plot-Wspec	24
plot-WspecMat	25
prepComb	26
quantize	27
quantplot	28
show-WaveWspec-methods	30
smoother	31
summary-methods	31
tuneR	32
Wave	34
Wave-class	35
Waveforms	35
WaveIO	37
WavPlayer	38
Wspec-class	39
WspecMat-class	40
[-methods	41

Index 42

Arith-methods *Arithmetics on Waves*

Description

Methods for arithmetics on Wave objects

Methods

object = "Wave" An object of class [Wave](#).

object = "numeric" For, e.g., adding a number to the whole Wave, e.g. useful for demeaning.

object = "missing" For unary Wave operations.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

For the S3 generic: [groupGeneric](#), [Wave-class](#), [Wave](#)

bind	<i>Concatenating Wave objects</i>
------	-----------------------------------

Description

Concatenating objects of class `Wave`.

Usage

```
bind(...)
```

Arguments

... Objects of class `Wave`, each of the same kind (checked by `equalWave`), i.e. identical sampling rate, resolution (bit), and number of channels (stereo/mono).

Value

An object of class `Wave`.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[prepComb](#) for preparing the concatenation, [Wave-class](#), [Wave](#), [extractWave](#), [stereo](#)

channel	<i>Channel conversion for Wave objects</i>
---------	--

Description

Convenient wrapper to extract one or more channels (or mirror channels) from an object of class `Wave`.

Usage

```
channel(object, which = c("both", "left", "right", "mirror"))
```

Arguments

`object` Object of class `Wave`.
`which` Character indicating which channel(s) should be returned.

Value

Wave object including channels specified by `which`.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave](#), [Wave-class](#), [mono](#), [extractWave](#)

downsample

Downsampling a Wave object

Description

Downsampling an object of class `Wave`.

Usage

```
downsample(object, samp.rate)
```

Arguments

<code>object</code>	Object of class Wave .
<code>samp.rate</code>	Sampling rate the object is to be downsampled to. <code>samp.rate</code> must be in <code>[2000, 192000]</code> ; typical values are 11025, 22050, and 44100 for CD quality. If the object's sampling rate is already equal or smaller than <code>samp.rate</code> , the object will be returned unchanged.

Value

An object of class [Wave](#).

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [Wave](#)

`equalWave`*Checking Wave objects*

Description

Checks for some kind of equality of objects of class `Wave`.

Usage

```
equalWave(object1, object2)
```

Arguments

`object1`, `object2`
Object(s) of class `Wave`.

Value

Does not return anything. It `stops` code execution with an error message indicating the problem if the two objects don't have the same properties, i.e. identical sampling rate, resolution (bit), and number of channels (stereo/mono).

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [Wave](#)

`extractWave`*Extractor for Wave objects*

Description

Extractor function that allows to extract inner parts for `Wave` objects (interactively).

Usage

```
extractWave(object, from = 1, to = length(object@left),  
            interact = interactive(), xunit = c("samples", "time"), ...)
```

Arguments

object	Object of class Wave .
from	Sample number or time in seconds (see <code>xunit</code>) at which to <i>start</i> extraction.
to	Sample number or time in seconds (see <code>xunit</code>) at which to <i>stop</i> extraction. If <code>to < from</code> , object will be returned as is.
interact	Logical indicating whether to choose the range to be extracted interactively (if <code>TRUE</code>). See Section Details.
xunit	Character indicating which units are used to specify the range to be extracted (both in arguments <code>from</code> and <code>to</code> , and in the plot, if <code>interact = TRUE</code>). If <code>xunit = "time"</code> , the unit is time in seconds, otherwise the number of samples.
...	Parameters to be passed to the underlying plot function (plot-methods), if <code>interact = TRUE</code> .

Details

This function allows interactive selection of a range to be extracted from an object of class [Wave](#). The default is to use interactive selection, if the current R session is [interactive](#). In case of interactive selection, [plot-methods](#) plot the [Wave](#) object, and the user may click on the starting and ending points of his selection (given neither `from` nor `to` have been specified, see below). The cut-points are drawn and the corresponding selection will be returned in form of a [Wave](#) object.

Setting `interact = TRUE` in a non-interactive session does not work.

Setting arguments `from` or `to` explicitly means that the specified one does not need to be selected interactively, hence only the non-specified one will be selected interactively. Moreover, setting both `from` or `to` implies `interact = FALSE`.

Value

An object of class [Wave](#).

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [Wave](#), [bind](#), [channel](#), [mono](#)

Examples

```
Wobj <- sine(440, bit = 16)
# extracting the middle 0.5 seconds of that 1 sec. sound:
Wobj2 <- extractWave(Wobj, from = 0.25, to = 0.75, xunit = "time")
Wobj2

## Not run:
# or interactively:
```

```
Wobj2 <- extractWave(Wobj)

## End(Not run)
```

Description

Estimation of Fundamental Frequencies from an object of class `Wspec`. Additionally, some heuristics are used to distinguish silence, noise (and breathing for singers) from real tones.

Usage

```
FF(object, peakheight = 0.01, silence = 0.2, minpeak = 9, diapason = 440,
    notes = NULL, interest.frqs = seq(along = object@freq),
    search.par = c(0.8, 10, 1.3, 1.7))

FFpure(object, peakheight = 0.01, diapason = 440,
        notes = NULL, interest.frqs = seq(along = object@freq),
        search.par = c(0.8, 10, 1.3, 1.7))
```

Arguments

<code>object</code>	An object of class <code>Wspec</code> .
<code>peakheight</code>	The peak's proportion of the maximal peak height to be considered for fundamental frequency detection. The default (0.01) means peaks smaller than 0.02 times the maximal peak height are omitted.
<code>silence</code>	The maximum proportion of periodograms to be considered as silence or noise (such as breathing). The default (0.2) means that less than 20 out of 100 periodograms represent silence or noise.
<code>minpeak</code>	If more than <code>minpeak</code> peaks are considered for detection and passed argument <code>peakheight</code> , such periodograms are detected to be silence or noise (if <code>silence > 0</code>).
<code>diapason</code>	Frequency of diapason a, default is 440 (Hertz).
<code>notes</code>	Optional, a vector of integers indicating the notes (in halftones from diapason a) that are expected. By applying this restriction, the "detection error" might be reduced in some cases.
<code>interest.frqs</code>	Optional, either a vector of integers indicating the indices of (fundamental) frequencies in <code>object</code> that are expected, or one of the character strings "bass", "tenor", "alto" or "soprano". For these voice types, only typical frequency ranges are considered for detection. By applying this restriction, the "detection error" might be reduced in some cases.

`search.par` Parameters to look for peaks:

1. The first peak larger than `peakheight * 'largest_peak'` is taken.
2. Its frequency is multiplied by `1+search.par[1]` Now, any larger peak between the old peak and that value is taken, if (a) it exists and if (b) it is above the `search.par[2]`-th Fourier-Frequency.
3. Within the interval of frequencies `'current peak' * search.par[3:4]`, another high peak is looked for. If any high peak exists in that interval, it can be assumed we got the wrong partial and the 'real' fundamental frequency can be re-estimated from the next two partials.

Details

`FFpure` just estimates the fundamental frequencies for all periodograms contained in the `object` (of class `Wspec`).

`FF` additionally uses some heuristics to distinguish silence, noise (and breathing for singers) from real tones. It is recommended to use the wrapper function `FF` rather than `FFpure`. If silence detection can be omitted by specifying `silence = 0`.

Value

Vector of estimated fundamental frequencies (in Hertz) for each periodogram contained in `object`.

Note

These functions are still in development and may be changed in due course.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

`Wspec`, `periodogram` (including an example), `noteFromFF`, and `tuneR` for a very complete example.

length

S4 generic for length

Description

S4 generic for length.

Methods

`x = "Wave"` The length of the left channel (in samples) of this object of class `Wave` will be returned.

`object = "ANY"` For compatibility.

See Also

For the primitive: [length](#)

lilyinput

Providing LilyPond compatible input

Description

A function (*in development!*) that writes a file to be processed by *LilyPond* by extracting the relevant information (e.g. pitch, length, ...) from columns of a data frame. The music notation software *LilyPond* can “transcribe” such an input file into sheet music.

Usage

```
lilyinput(X, file = "Rsong.ly", Major = TRUE, key = "c",
         clef = c("treble", "bass", "alto", "tenor"), time = "4/4",
         endbar = TRUE, midi = TRUE, tempo = "2 = 60",
         textheight = 220, linewidth = 150, indent = 0, fontsize = 14)
```

Arguments

X	A data frame containing 4 named components (columns): <ul style="list-style-type: none"> • note: Integer - the notes' pitch in halftones from diapason (a), i.e. 0 for diapason a, 3 for c', ... • length: Integer - denominator of lengths of the notes, e.g. 8 for a quaver. • punctate: Logical - whether to punctate a note. • slur: Logical - TRUE indicates to start a slur, or to end it. That means that the first, third, ... occurrences of TRUE start slurs, while the second, fourth, ... occurrences end slurs. Note that it is only possible to draw one slur at a time.
file	The file to be written for <i>LilyPond</i> 's input.
Major	Logical indicating major key (if TRUE) or minor key.
key	Keynote, necessary to set sharps/flats.
clef	Integer indicating the kind of clef, supported are 'treble' (default), 'bass', 'alto', and 'tenor'.
time	Character indicating which meter to use, examples are: "3/4", "4/4".
endbar	Logical indicating whether to set an ending bar at the end of the sheet music.
midi	Logical indicating whether Midi output (by <i>LilyPond</i>) is desirable.
tempo	Character specifying the tempo to be used for the Midi file if <code>midi = TRUE</code> . The default, "2 = 60" indicates: 60 half notes per minute, whereas "4 = 90" indicates 90 quarters per minute.
textheight	Textheight of the sheet music to be written by <i>LilyPond</i> .
linewidth	Linewidth of the sheet music to be written by <i>LilyPond</i> .
indent	Indentation of the sheet music to be written by <i>LilyPond</i> .
fontsize	Fontsize of the sheet music to be written by <i>LilyPond</i> .

Details

Details will be given when development has reached a stable stage ...!

Value

Nothing is returned, but a file is written.

Note

This function is in development!!!

Everything (and in particular its user interface) is subject to change!!!

Author(s)

Andrea Preußer and Uwe Ligges, <ligges@statistik.tu-dortmund.de>

References

The LilyPond development team (2005): *LilyPond - The music typesetter*. <http://www.lilypond.org/>, Version 2.7.20.

Preußer, A., Ligges, U. und Weihs, C. (2002): *Ein R Exportfilter für das Notations- und Midi-Programm LilyPond*. Arbeitsbericht 35. Fachbereich Statistik, Universität Dortmund. (german)

See Also

[quantMerge](#) prepares the data to be written into the LilyPond format; [quantize](#) and [quantplot](#) generate another kind of plot; and exhaustive example is given in [tuneR](#).

melodyplot

Plotting a melody

Description

Plot a observed melody and (optional) an expected melody, as well as corresponding energy values (corresponding to the loudness of the sound).

Usage

```
melodyplot(object, observed, expected = NULL, bars = NULL,
  main = NULL, xlab = NULL, ylab = "note", xlim = NULL, ylim = NULL,
  observedtype = "l", observedcol = "red", expectedcol = "grey",
  gridcol = "grey", lwd = 2, las = 1, cex.axis = 0.9,
  mar = c(5, 4, 4, 4) + 0.1, notenames = NULL, thin = 1,
  silence = "silence", plotenergy = TRUE, ...,
  axispar = list(ax1 = list(side=1),
                 ax2 = list(side=2),
                 ax4 = list(side=4)),
```

```

boxpar = list(),
energylabel = list(text="energy", side=4, line=2.5, at=rg.s-0.25, las=3),
energypar = list(),
expectedpar = list(),
gridpar = list(col = gridcol),
observedpar = list(col=observedcol, type=observedtype, lwd=2, pch=15)

```

Arguments

object	An object of class Wspec .
observed	Observed notes, probably as a result from noteFromFF (or a smoothed version). This should correspond to the Wspec object. It can also be a matrix of k columns where those k notes in the same row are displayed at the same timepoint.
expected	Expected notes (optional; in order to compare results), same format as <code>observed</code> .
bars	Number of bars to be plotted (a virtual static segmentation takes place). If <code>NULL</code> (default), time rather than bars are used.
main	Main title of the plot.
xlab, ylab	Annotation of x/y-axes.
xlim, ylim	Range of x/y-axis, where <code>ylim</code> must be an integer that represents the range of note heights that should be displayed.
observedtype	Type (either "p" for points or "l" for lines) used for representing observed notes. "l" (the default) is not sensible for polyphonic representations.
observedcol	Colour for the observed melody.
expectedcol	Colour for the expected melody.
gridcol	Colour of the grid.
lwd	Line width, see par for details.
las	Orientation of axis labels, see par for details.
cex.axis	Size of tick mark labels, see par for details.
mar	Margins of the plot, see par for details.
notenames	Optionally specify other notenames (character) for the y axis.
thin	Amount of thinning of notenames, i.e. only each <code>thin</code> th notename is displayed on the y-axis.
silence	Character string for label of the 'silence' (default) axis.
plotenergy	Logical (default: <code>TRUE</code>), whether to plot energy values in the bottom part of the plot.
...	Additional graphical parameters to be passed to underlying <code>plot</code> function.
axispar	A named list of three other lists (<code>ax1</code> , <code>ax2</code> , and <code>ax4</code>) containing parameters passed to the corresponding axis calls for the three axis time (<code>ax1</code>), notes (<code>ax2</code>), and energy (<code>ax4</code>).
boxpar	A list of parameters to be passed to the box generating functions.
energylabel	A list of parameters to be passed to the energy-label generating <code>mtext</code> call.

energypar	A list of parameters to be passed to the lines function that draws the energy curve.
expectedpar	A list of parameters to be passed to the rect function that draws the rectangles for expected values.
gridpar	A list of parameters to be passed to the abline function that draws the grid lines.
observedpar	A list of parameters to be passed to the lines function that draws the observed values.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[noteFromFF](#), [FF](#), [quantplot](#); for an example, see the help in [tuneR](#).

 MFCC

Mel Frequency Cepstral Coefficients

Description

Computation of MFCCs (Mel Frequency Cepstral Coefficients) for a `Wave` object.

Usage

```
MFCC(object, a = 0.1, HW.width = 0.025, HW.overlapping = 0.25,
      T.number = 24, T.overlapping = 0.5, K = 12)
```

Arguments

object	Object of class Wave .
a	Coefficient for a first order difference filter, which is used to pre-emphasize the signal in first step of feature extraction.
HW.width	Width of Hamming window in seconds, which is used to divide the signal into frames.
HW.overlapping	Fraction of how much the Hamming windows should overlap.
T.number	Number of triangular channels on the mel scaled spectrum, which are mapped to the signal.
T.overlapping	Fraction of how much the triangular filters should overlap.
K	Number of desired output frequencies the inverse discrete cosine transformation.

Details

This function computes Mel Frequency Cepstral Coefficients (MFCC) for an object of class [Wave](#). In speech recognition MFCCs are used to extract the stimulus of the vocal tract from speech. The process to create the MFCC features consist of five steps. First the signal from `object` is filtered with a finite impulse response (FIR) filter to pre-amplify high frequencies. Only the left channel of `object`, i.e. a mono signal, is used for the extraction. The parameter `a` controls the FIR filter. The filtered signal $S.fil$ at time t is obtained by $S.fil(t) = S(t) - a * S(t - 1)$. In a second step the signal is converted to frames, each of length `HW.width`. A Hamming window is used to avoid any negative effects on the edges of each frame due to the conversion. After a discrete Fourier transformation (DFT) the signal is mapped to the Mel scale filter bank. The filter bank consists of `T.number` triangular filters, which overlap by `T.overlapping`. This performs a perceptual weighting of frequencies. In a last step an inverse discrete cosine transformation is applied to the signal. `K` controls the order, up to which MFCC features are computed.

Value

A matrix (number of Hamming windows)-rows and `K+1` columns. The first column is the energy, the following `K` columns the extracted MFCC features.

Note

This function is still in development and highly EXPERIMENTAL!!!

Author(s)

Julia Schiffner <schiffner@statistik.tu-dortmund.de> and Gero Szepannek <szepannek@statistik.tu-dortmund.de> and Uwe Ligges <ligges@statistik.tu-dortmund.de>

References

Young, S., Everman, G., Gales, M., Hain, T., Kershaw, D., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P. (2005): *The HTK-Book (v 3.3)*, Cambridge University Engineering Dept., 59-61.

See Also

[Wave-class](#), [Wave](#)

Examples

```
obj <- sine(440, bit = 16, duration = 5000)
MFCC(obj)
```

Mono-Stereo

Converting (extracting, joining) stereo to mono and vice versa

Description

Functions to extract a channel from a stereo `Wave` object, and to join channels of two monophonic `Wave` objects to a stereophonic one.

Usage

```
mono(object, which = c("left", "right", "both"))
stereo(left, right)
```

Arguments

<code>object</code>	Object of class <code>Wave</code> .
<code>which</code>	Character, indicating whether the “left” or “right” channel should be extracted, or whether “both” channels should be averaged.
<code>left</code>	Object of class <code>Wave</code> containing monophonic sound, to be used for the left channel.
<code>right</code>	Object of class <code>Wave</code> containing monophonic sound, to be used for the right channel (if missing, the left channel is duplicated). If <code>right</code> is missing, <code>stereo</code> returns whether <code>left</code> is stereo (<code>TRUE</code>) or mono (<code>FALSE</code>).

Value

An object of class `Wave`.

If argument `right` is missing in `stereo`, a logical values is returned that indicates whether `left` is stereo (`TRUE`) or mono (`FALSE`).

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [Wave](#)

Examples

```
Wobj <- sine(440, bit = 16)
Wobj
Wobj2 <- stereo(Wobj, Wobj)
Wobj2
mono(Wobj2, "right")
```

normalize	<i>Rescale the range of values</i>
-----------	------------------------------------

Description

Centering and rescaling the waveform of a `Wave` object to either `[-1,1]`, `[0, 254]`, `[-32767, 32767]`, `[-8388607, 8388607]`, or `[-2147483647, 2147483647]`.

Usage

```
normalize(object, unit = c("1", "8", "16", "24", "32", "0"), center = TRUE, level =
```

Arguments

object	Object of class <code>Wave</code> .
unit	Unit to rescale to. "1" (default) for rescaling to real values in <code>[-1,1]</code> , "8" (i.e. 8-bit) for rescaling to integers in <code>[0, 254]</code> , "16" (i.e. 16-bit) for rescaling to integers in <code>[-32767, 32767]</code> , 24 (i.e. 24-bit) for rescaling to integers in <code>[-8388607, 8388607]</code> , 32 (i.e. 32-bit) for rescaling to integers in <code>[-2147483647, 2147483647]</code> , and "0" for not rescaling (hence only centering, if <code>center=TRUE</code>).
center	If <code>TRUE</code> (default), values are centered around 0 (or 127, if <code>unit="8"</code>).
level	Maximal percentage of the amplitude used for normalizing (default is 1).

Value

An object of class `Wave`.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>, based on code from Matthias Heymann's former package 'sound'.

See Also

[Wave-class](#), [Wave](#), [writeWave](#)

noSilence

*Cut off silence from a Wave object***Description**

Cut off silence or low noise at the beginning and/or at the end of an object of class `Wave`.

Usage

```
noSilence(object, zero = 0, level = 0, where = c("both", "start", "end"))
```

Arguments

<code>object</code>	Object of class <code>Wave</code> .
<code>zero</code>	The zero level (default: 0) at which ideal cut points are determined (see Details). A typical alternative would be 127 for 8 bit <code>Wave</code> objects. If <code>zero=NA</code> , the mean of the left <code>Wave</code> channel is taken as zero level.
<code>level</code>	Values in the interval between <code>zero</code> and <code>zero - level/zero + level</code> are considered as silence.
<code>where</code>	One of “both” (default), “start”, or “end” indicating at where to prepare the <code>Wave</code> object for concatenation.

Details

Silence is removed at the locations given by `where` of the `Wave` object, where silence is defined such that (in both channels, if stereo) all values are in the interval between `zero` and `zero - level/zero + level`. All values before (or after, respectively) the first non-silent value are removed from the object.

Value

An object of class `Wave`.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>, based on code from Matthias Heymann's former package 'sound'.

See Also

[silence](#), [Wave-class](#), [Wave](#), [extractWave](#)

`noteFromFF`*Deriving notes from frequencies*

Description

Deriving notes from given (fundamental) frequencies.

Usage

```
noteFromFF(x, diapason = 440, roundshift = 0)
```

Arguments

<code>x</code>	Fundamental frequency.
<code>diapason</code>	Frequency of diapason a, default is 440 (Hertz).
<code>roundshift</code>	Shift that indicates from here to round to the next integer (note). The default (0) is “classical” rounding as described in round . A higher value means that <code>roundshift</code> is added to the calculated real note value before rounding to an integer. This is useful if it is unclear that some instruments really shift the note in the center between two theoretical frequencies. Example: if <code>x=452</code> and <code>diapason=440</code> , the internally calculated real value of 0.46583 is rounded to 0, but for <code>roundshift=0.1</code> we get 0.56583 and it is rounded to note 1.

Details

The formula used is simply `round(12 * log(x / diapason, 2) + roundshift)`.

Value

An integer representing the (rounded) difference in halftones from diapason a, i.e. indicating the note that corresponds to fundamental frequency `x` given the value of `diapason`. For example: 0 indicates diapason a, 3: c', 12: a', ...

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[FF](#), [periodogram](#), and [tuneR](#) for a very complete example.

`notenames`*Generating note names from numbers*

Description

A function that generates note names from numbers

Usage

```
notenames(notes, language = c("english", "german"))
```

Arguments

<code>notes</code>	An interger values vector, where 0 corresponds to a' , notes below and above have to be specified in the corresponding halftone distance.
<code>language</code>	Language of the note names. Currently only english and german are supported.

Value

A character vector of note names.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

Examples

```
notenames(c(-24, -12, 0, 12)) # octaves of a
notenames(3:15)                # chromaticism

## same in german:
notenames(3:15, language = "german")
```

`panorama`*Narrow the Panorama of a Stereo Sample*

Description

Narrow the panorama of a stereo Wave object.

Usage

```
panorama(object, pan = 1)
```

Arguments

object	Object of class Wave .
pan	Value in [-1,1] to narrow the panorama, see the Details below. The default (1) does not change anything.

Details

If $\text{abs}(\text{pan}) < 1$, mixtures of the two channels of the `Wave` objects are used for the left and the right channel of the returned `Sample` object, so that they appear closer to the center.

For $\text{pan} = 0$, both sounds are completely in the center (i.e. averaged).

If $\text{pan} < 0$, the left and the right channel are interchanged.

Value

Wave object with the transformed panorama.

Author(s)

Uwe Ligges, based on code by Matthias Heymann

See Also

[Wave-class](#), [Wave](#)

periodogram-methods

Periodogram (Spectral Density) Estimation on Wave objects

Description

This function estimates one or more periodograms (spectral densities) of the time series contained in an object of class `Wave` (or directly in a `Wave` file) using a window running through the time series (possibly with overlapping). It returns an object of class `Wspec`.

Usage

```
periodogram(object, ...)
## S4 method for signature 'Wave':
periodogram(object, width = length(object), overlap = 0,
             starts = NULL, ends = NULL, taper = 0, normalize = TRUE,
             frqRange = c(-Inf, Inf), ...)
## S4 method for signature 'character':
periodogram(object, width, overlap = 0, from = 1, to = Inf,
             units = c("samples", "seconds", "minutes", "hours"),
             downsample = NA, channel = c("left", "right"), pieces = 1, ...)
```

Arguments

<code>object</code>	An object of class <code>Wave</code> or a character string pointing to a Wave file.
<code>width</code>	A window of width ‘width’ running through the time series selects the samples from which the periodograms are to be calculated.
<code>overlap</code>	The window can be applied by each overlapping <code>overlap</code> samples.
<code>starts</code>	Start number (in samples) for a window. If not given, this value is derived from argument <code>ends</code> , or will be derived <code>width</code> and <code>overlap</code> .
<code>ends</code>	End number (in samples) for a window. If not given, this value is derived from argument <code>starts</code> , or will be derived from <code>width</code> and <code>overlap</code> .
<code>taper</code>	proportion of data to taper. See <code>spec.pgram</code> for details.
<code>normalize</code>	Logical; if TRUE (default), two steps will be applied: (i) the input signal will be normalized to amplitude $\max(\text{abs}(\text{amplitude})) == 1$, (ii) the resulting <code>spec</code> values will be normalized to sum up to one for each periodogram.
<code>freqRange</code>	Numeric vector of two elements indicating minimum and maximum of the frequency range that is to be stored in the resulting object. This is useful to reduce memory consumption.
<code>from</code>	Where to start reading in the Wave file, in <code>units</code> .
<code>to</code>	Where to stop reading in the Wave file, in <code>units</code> .
<code>units</code>	Units in which <code>from</code> and <code>to</code> is given, the default is “samples”, but can be set to time intervals such as “seconds”, see the Usage Section above.
<code>downsample</code>	Sampling rate the object is to be downsampled to. If NA, the default, no changes are applied. Otherwise <code>downsample</code> must be in <code>[2000, 192000]</code> ; typical values are 11025, 22050, and 44100 for CD quality. See also <code>downsample</code> .
<code>channel</code>	Character, indicating whether the “left” or “right” channel should be extracted (see <code>mono</code> for details) - stereo processing is not yet implemented.
<code>pieces</code>	The Wave file will be read in <code>pieces</code> steps in order to reduce the amount of required memory.
<code>...</code>	Further arguments to be passed to the underlying function <code>spec.pgram</code> .

Value

An object of class `Wspec` is returned containing the following slots.

<code>freq</code>	Vector of frequencies at which the spectral density is estimated. See <code>spectrum</code> for details. (1)
<code>spec</code>	List of vectors or matrices of the <code>spec</code> values returned by <code>spec.pgram</code> at frequencies corresponding to <code>freq</code> . Each element of the list corresponds to one periodogram estimated from samples of the window beginning at <code>start</code> of the <code>Wave</code> object.
<code>kernel</code>	The kernel argument, or the kernel constructed from spans returned by <code>spec.pgram</code> . (1)
<code>df</code>	The distribution of the spectral density estimate can be approximated by a chi square distribution with <code>df</code> degrees of freedom. (1)

taper	The value of the taper argument. (1)
width	The value of the width argument. (1)
overlap	The value of the overlap argument. (1)
normalize	The value of the normalize argument. (1)
starts	If the argument starts was given in the call, its value. If the argument ends was given in the call, 'ends - width'. If neither starts nor ends was given, the start points of all periodograms. In the latter case the start points are calculated from the arguments width and overlap.
stereo	Whether the underlying Wave object was stereo or not. (1)
samp.rate	Sampling rate of the underlying Wave object. (1)
variance	The variance of samples in each window, corresponding to amplitude / loudness of sound.
energy	The "energy" E , also an indicator for the amplitude / loudness of sound:

$$E(x_I) := 20 * \log_{10} \sum_{j \in I} |x_j|,$$

where I indicates the interval $I := \text{start}[i] : \text{end}[i]$ for all $i := 1, \dots, \text{length}(\text{starts})$.

Those slots marked with "(1)" contain the information once, because it is unique for all periodograms of estimated by the function call.

Note

Support for processing of stereo Wave objects has not yet been implemented.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

- for the resulting objects' class: [Wspec](#),
- for plotting: [plot-Wspec](#),
- for the underlying periodogram calculations: [spec.pgram](#),
- for the input data class: [Wave-class](#), [Wave](#).

Examples

```
# constructing a Wave object (1 sec.) containing sinus sound with 440Hz:
Wobj <- sine(440, bit = 16)
Wobj

# Calculate periodograms in windows of 4096 samples each - without
# any overlap - resulting in an Wspec object that is printed:
Wspecobj <- periodogram(Wobj, width = 4096)
```

```

Wspecobj

# Plot the first periodogram from Wspecobj:
plot(Wspecobj)
# Plot the third one and choose a reasonable xlim:
plot(Wspecobj, which = 3, xlim = c(0, 1000))
# Mark frequency that has been generated before:
abline(v = 440, col="red")

FF(Wspecobj)           # all ~ 440 Hertz
noteFromFF(FF(Wspecobj)) # all diapason a

```

play-methods

Playing Waves

Description

Plays wave files and objects of class `Wave`.

Usage

```
play(object, player, ...)
```

Arguments

<code>object</code>	Either a filename pointing to a <code>Wave</code> file, or an object of class <code>Wave</code> . If the latter, it is written to a temporary file by <code>writeWave</code> , played by the chosen player, and deleted afterwards.
<code>player</code>	(Path to) a program capable of playing a wave file by invocation from the command line. If under Windows and no player is given, "mplay32.exe" will be chosen as the default.
<code>...</code>	Further arguments passed to the <code>Wave</code> file player. If no <code>player</code> and no further arguments are given under Windows, the default is: <code>"/play /close"</code> .

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [Wave](#), [writeWave](#), [setWavPlayer](#)

plot	<i>S4 generic for plot</i>
------	----------------------------

Description

S4 generic for plot.

Methods

`x = "ANY", y = "ANY"` Any object for which a plot is desired.

See Also

For the S3 generic: `plot.default`

plot-Wave	<i>Plotting Wave objects</i>
-----------	------------------------------

Description

Plotting objects of class `Wave`.

Usage

```
## S4 method for signature 'Wave,missing':
plot(x, info = FALSE, xunit = c("time", "samples"),
     ylim = NULL, main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
     simplify = TRUE, nr = 1500, ...)

plot.Wave.channel(x, xunit, ylim, xlab, ylab, main, nr, simplify, ...)
```

Arguments

<code>x</code>	Object of class <code>Wave</code> .
<code>info</code>	Logical, whether to include (written) information on the <code>Wave</code> object within the plot.
<code>xunit</code>	Character indicating which units are used for setting up user coordinates (see <code>par</code>) and x-axis labeling. If <code>xunit = "time"</code> , the unit is time in seconds, otherwise the number of samples.
<code>ylim</code>	The y (amplitude) limits of the plot.
<code>main, sub</code>	A main / sub title for the plot.
<code>xlab, ylab</code>	Label for x-/y-axis.

simplify	Logical, whether the plot should be “simplified”. If TRUE (default), not all (thousand/millions/billions) of points (samples) of the Wave object are drawn, but the nr (see below) ranges (in form of segments) within nr windows of the time series. Plotting with <code>simplify = FALSE</code> may take several minutes (depending on the number of samples in the <code>Wave</code>) and output in any vector format may be really huge.
nr	Number of windows (segments) to be used <i>approximately</i> (an appropriate number close to nr is selected) to <code>simplify</code> (see above) the plot. Only used if <code>simplify = TRUE</code> and the number of samples of <code>Wave</code> object <code>x</code> is larger.
...	Further arguments to be passed to the underlying plot functions.

Details

Function `plot.Wave.channel` is a helper function to plot a single (left!) channel; in particular it is *not* intended to be called by the user directly.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [Wave](#) and [tuneR](#)

plot-Wspec

Plotting Wspec objects

Description

Plotting a periodogram contained in an object of class `Wspec`.

Usage

```
## S4 method for signature 'Wspec,missing':
plot(x, which = 1, type = "h", xlab = "Frequency",
     ylab = NULL, log = "", ...)
```

Arguments

<code>x</code>	Object of class Wspec .
<code>which</code>	Integer indicating which of the periodograms contained in object <code>x</code> to plot. Default is to plot the first one.
<code>type</code>	The default is to plot horizontal lines, rather than points. See plot.default for details.
<code>xlab</code> , <code>ylab</code>	Label for x-/y-axis.

log Character - "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic (quite typical for some visualizations of periodograms), and "xy" or "yx" if both axes are to be logarithmic.

... Further arguments to be passed to the underlying plot functions. See [plot.default](#) for details.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

see [Wspec](#), [periodogram](#) and [tuneR](#) for the constructor function and some examples.

plot-WspecMat

Plotting WspecMat objects

Description

Plotting a spectrogram (image) of an object of class `Wspec` or `WspecMat`.

Usage

```
## S4 method for signature 'WspecMat,missing':
plot(x, xlab = "time", ylab = "Frequency",
     xunit = c("samples", "time"), log = "", ...)
## S4 method for signature 'Wspec':
image(x, xlab = "time", ylab = "Frequency",
      xunit = c("samples", "time"), log = "", ...)
```

Arguments

x Object of class [WspecMat](#) (for `plot`) or [Wspec](#) (for `image`).

xlab, ylab Label for x-/y-axis.

xunit Character indicating which units are used to annotate the x axis. If `xunit = "time"`, the unit is time in seconds, otherwise the number of samples.

log Character - "z" if the z values are to be logarithmic.

... Further arguments to be passed to the underlying [image](#) function. See [image](#) for details.

Details

Calling `image` on a `Wspec` object converts it to class `WspecMat` and calls the corresponding `plot` function.

Calling `plot` on a `WspecMat` object generates an [image](#) with correct annotated axes.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

see [image](#), [Wspec](#), [WspecMat](#), [periodogram](#) and [tuneR](#) for the constructor function and some examples.

```
prepComb
```

Preparing the combination/concatenation of Wave objects

Description

Preparing objects of class `Wave` for binding/combination/concatenation by removing small amounts at the beginning/end of the `Wave` in order to make the transition smooth by avoiding clicks.

Usage

```
prepComb(object, zero = 0, where = c("both", "start", "end"))
```

Arguments

<code>object</code>	Object of class <code>Wave</code> .
<code>zero</code>	The zero level (default: 0) at which ideal cut points are determined (see Details). A typical alternative would be 127 for 8 bit <code>Wave</code> objects. If <code>zero=NA</code> , the mean of the left <code>Wave</code> channel is taken as zero level.
<code>where</code>	One of “both” (default), “start”, or “end” indicating at <code>where</code> to prepare the <code>Wave</code> object for concatenation.

Details

This function is useful to prepare objects of class `Wave` for binding/combination/concatenation. At the side(s) indicated by `where` small amounts of the `Wave` are removed in order to make the transition between two `Waves` smooth (avoiding clicks).

This is done by dropping all values at the *beginning* of a `Wave` before the first positive point after the `zero` level is crossed from negative to positive. Analogously, at the *end* of a `Wave` all points are cut after the last negative value before the last `zero` level crossing from negative to positive.

Value

An object of class `Wave`.

Note

If stereo, only the left channel is analyzed while the right channel will be simply cut at the same locations.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>, based on code from Matthias Heymann's former package 'sound'.

See Also

[bind](#), [Wave-class](#), [Wave](#), [extractWave](#), and [noSilence](#) to cut off silence

Examples

```
Wobj1 <- sine(440, duration = 520, bit = 16)
Wobj2 <- extractWave(sine(330, duration = 500, bit = 16), from = 110, to = 500)
par(mfrow = c(2,1))
plot(bind(Wobj1, Wobj2), xunit = "samples")
abline(v = 520, col = "red") # here is a "click"!

# now remove the "click" by deleting a minimal amount of information:
Wobj1 <- prepComb(Wobj1, where = "end")
Wobj2 <- prepComb(Wobj2, where = "start")
plot(bind(Wobj1, Wobj2), xunit = "samples")
```

quantize

Functions for the quantization of notes

Description

These functions apply (static) quantization of notes in order to produce sheet music by pressing the notes into bars.

Usage

```
quantize(notes, energy, parts)
quantMerge(notes, minlength, barsize, bars)
```

Arguments

notes	Series of notes, a vector of integers such as returned by noteFromFF . At least one argument (notes and/or energy) must be specified.
energy	Series of energy values, a vector of numerics such as corresponding components of a Wspec object.
parts	Number of outgoing parts. The notes vector is divided into parts bins, the outcome is a vector of the modes of all bins.
minlength	1/(length of the shortest note). Example: if the shortest note is a quaver (1/8), set minlength=8.
barsize	One bar contains barsize number of notes of length minlength.
bars	We expect bars number of bars.

Value

`quantize` returns a list with components:

`notes` Vector of length `parts` corresponding to the input data. The data is binned and modes corresponding to the data in those bins are returned.

`energy` Same as `notes`, but for the `energy` argument.

`quantMerge` returns a `data.frame` with components:

`note` integer representation of a note (see `Arguments`).

`duration` $1/\text{duration}$ of a note (see `minlength` in `Section Arguments`), if `punctuation=FALSE`.

`punctuation` Whether the note should be punctuated. If `TRUE`, the real duration is 1.5 times the duration given in `duration`.

`slur` currently always `FALSE`, sensible processing is not yet implemented. It is supposed to indicate the beginning and ending positions of slurs.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

to get the input: `noteFromFF`, for plotting: `quantplot`, for further processing: `lilyinput`, to get notenames: `notenames`; for an example, see the help in `tuneR`.

quantplot

Plotting the quantization of a melody

Description

Plot an observed melody and (optional) an expected melody, as well as corresponding energy values (corresponding to the loudness of the sound) within a quantization grid.

Usage

```
quantplot(observed, energy = NULL, expected = NULL, bars,
  barseg = round(length(observed) / bars),
  main = NULL, xlab = NULL, ylab = "note", xlim = NULL, ylim = NULL,
  observedcol = "red", expectedcol = "grey", gridcol = "grey",
  lwd = 2, las = 1, cex.axis = 0.9, mar = c(5, 4, 4, 4) + 0.1,
  notenames = NULL, silence = "silence", plotenergy = TRUE, ...,
  axispar = list(ax1 = list(side=1), ax2 = list(side=2), ax4 = list(side=4)),
  boxpar = list(),
  energylabel = list(text="energy", side=4, line=2.5, at=rg.s-0.25, las=3),
  energypar = list(pch = 20),
  expectedpar = list(),
  gridpar = list(gridbar = list(col = 1), gridinner = list(col=gridcol)),
  observedpar = list(col = observedcol, pch = 15))
```

Arguments

observed	Either a vector of observed notes resulting from some quantization, or a list with components <code>notes</code> (observed notes) and <code>energy</code> (corresponding energy values), e.g. the result from a call to <code>quantize</code> .
energy	A vector of energy values with same quantization as <code>observed</code> (overwrites any given energy values, if <code>observed</code> is a list).
expected	Expected notes (optional; in order to compare results).
bars	Number of bars to be plotted (e.g. corresponding to <code>quantize</code> arguments).
barseg	Number of segments (minimal length notes) in each bar.
main	Main title of the plot.
xlab, ylab	Annotation of x/y-axes.
xlim, ylim	Range of x/y-axis.
observedcol	Colour for the observed notes.
expectedcol	Colour for the expected notes.
gridcol	Colour of the inner-bar grid.
lwd	Line width, see <code>par</code> for details.
las	Orientation of axis labels, see <code>par</code> for details.
cex.axis	Size of tick mark labels, see <code>par</code> for details.
mar	Margins of the plot, see <code>par</code> for details.
notenames	Optionally specify other notenames (character) for the y axis.
silence	Character string for label of the ‘silence’ (default) axis.
plotenergy	Logical indicating whether to plot energy values in the bottom part of the plot; default is <code>TRUE</code>), if energy values are specified, and <code>FALSE</code> otherwise.
...	Additional graphical parameters to be passed to underlying <code>plot</code> function.
axispar	A named list of three other lists (<code>ax1</code> , <code>ax2</code> , and <code>ax4</code>) containing parameters passed to the corresponding <code>axis</code> calls for the three axis time (<code>ax1</code>), notes (<code>ax2</code>), and energy (<code>ax4</code>).
boxpar	A list of parameters to be passed to the box generating functions.
energylabel	A list of parameters to be passed to the energy-label generating <code>mtext</code> call.
energypar	A list of parameters to be passed to the <code>points</code> function that draws the energy values.
expectedpar	A list of parameters to be passed to the <code>rect</code> function that draws the rectangles for expected values.
gridpar	A named list of two other lists (<code>gridbar</code> and <code>gridinner</code>) containing parameters passed to the <code>abline</code> functions that draw the grid lines (for bar separators and inner bar (note) separators).
observedpar	A list of parameters to be passed to the <code>lines</code> function that draws the observed values.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[noteFromFF](#), [FF](#), [melodyplot](#), [quantize](#); for an example, see the help in [tuneR](#).

show-WaveWspec-methods

Showing objects

Description

Showing Wave, Wspec, and WspecMat objects.

Methods

object = "Wave" The Wave object is being showed. The number of samples, duration in seconds, Samplingrate (Hertz), Stereo / Mono, and the resolution in bits are printed. Note that it does not make sense to print the whole channels containing several thousands or millions of samples.

object = "Wspec" The number of periodograms, Fourier frequencies, window width (used amount of data), amount of overlap of neighboring windows, and whether the periodogram(s) has/have been normalized will be printed.

object = "WspecMat" The number of periodograms, Fourier frequencies, window width (used amount of data), amount of overlap of neighboring windows, and whether the periodogram(s) has/have been normalized will be printed.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [Wave](#), [Wspec](#), [WspecMat](#), [plot-methods](#), [summary-methods](#), and [periodogram](#) for the constructor function and some examples

`smoother`*Meta Function for Smoothers*

Description

Apply a smoother to estimated notes. Currently, only a running median (using `decmedian` in package `pastecs`) is available.

Usage

```
smoother(notes, method = "median", order = 4, times = 2)
```

Arguments

<code>notes</code>	Series of notes, a vector of integers such as returned by <code>noteFromFF</code> .
<code>method</code>	Currently, only a running 'median' (using <code>decmedian</code> in package <code>pastecs</code>) is available.
<code>order</code>	The window used for the running median corresponds to $2 * \text{order} + 1$.
<code>times</code>	The number of times the running median is applied (default: 2).

Value

The smoothed series of notes.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

`summary-methods`*Object Summaries*

Description

`summary` is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

Methods

- object = "ANY"** Any object for which a summary is desired, dispatches to the S3 generic.
- object = "Wave"** The `Wave` object is being showed, and an additional summary of the `Wave`-object's (one or two) channels is given.
- object = "Wspec"** The `Wspec` object is being showed, and as an additional output is given: `df`, `taper` (see `spectrum`), and for the underlying `Wave` object the number of channels and its sampling rate.
- object = "WspecMat"** The `WspecMat` object is being showed, and as an additional output is given: `df`, `taper` (see `spectrum`), and for the underlying `Wave` object the number of channels and its sampling rate.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

For the S3 generic: `summary.default`, `plot-methods`, `Wave-class`, `Wave`, `Wspec`, `WspecMat`

tuneR

tuneR

Description

tuneR, a collection of examples

Functions in tuneR

`tuneR` consists of several functions to work with and to analyze `Wave` files. In the following examples, some of the functions to generate some data (such as `sine`), to read and write `Wave` files (`readWave`, `writeWave`), to represent or construct `Wave` files (`Wave`), to transform `Wave` objects (`bind`, `channel`, `downsample`, `extractWave`, `mono`, `stereo`), and to `play` `Wave` objects are used.

Other functions and classes are available to calculate several periodograms of a signal (`periodogram`, `Wspec`), to estimate the corresponding fundamental frequencies (`FF`, `FFpure`), to derive the corresponding notes (`noteFromFF`), and to apply a `smoother`. Now, the melody and corresponding energy values can be plotted using the function `melodyplot`.

A next step is the quantization (`quantize`) and a corresponding plot (`quantplot`) showing the note values for binned data. Moreover, a function called `lilyinput` (and a data-preprocessing function `quantMerge`) can prepare a data frame to be presented as sheet music by postprocessing with the music typesetting software LilyPond.

Of course, `print` (`show`), `plot` and `summary` methods are available for most classes.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

Examples

```

library(tuneR) # in a regular session, we are loading tuneR

# constructing a mono Wave object (2 sec.) containing sinus
# sound with 440Hz and folled by 220Hz:
Wobj <- bind(sine(440, bit = 16), sine(220, bit = 16))
show(Wobj)
plot(Wobj) # it does not make sense to plot the whole stuff
plot(extractWave(Wobj, from = 1, to = 500))
## Not run:
play(Wobj) # listen to the sound

## End(Not run)

tmpfile <- file.path(tempdir(), "testfile.wav")
# write the Wave object into a Wave file (can be played with any player):
writeWave(Wobj, tmpfile)
# reading it in again:
Wobj2 <- readWave(tmpfile)

Wobjm <- mono(Wobj, "left") # extract the left channel
# and downsample to 11025 samples/sec.:
Wobjm11 <- downsample(Wobjm, 11025)
# extract a part of the signal interactively (click for left/right limits):
## Not run:
Wobjm11s <- extractWave(Wobjm11)

## End(Not run)
# or extract some values reproducibly
Wobjm11s <- extractWave(Wobjm11, from=1000, to=17000)

# calculating periodograms of sections each consisting of 1024 observations,
# overlapping by 512 observations:
WspecObject <- periodogram(Wobjm11s, normalize = TRUE, width = 1024, overlap = 512)
# Let's look at the first periodogram:
plot(WspecObject, xlim = c(0, 2000), which = 1)
# or a spectrogram
image(WspecObject, ylim = c(0, 1000))
# calculate the fundamental frequency:
ff <- FF(WspecObject)
print(ff)
# derive note from FF given diapason a'=440
notes <- noteFromFF(ff, 440)
# smooth the notes:
snotes <- smoother(notes)
# outcome should be 0 for diapason "a" and -12 (12 halftones lower) for "a"
print(snotes)
# plot melody and energy of the sound:
melodyplot(WspecObject, snotes)

# apply some quantization (into 8 parts):
qnotes <- quantize(snotes, WspecObject@energy, parts = 8)

```

```
# an plot it, 4 parts a bar (including expected values):
quantplot(qnotes, expected = rep(c(0, -12), each = 4), bars = 2)
# now prepare for LilyPond
qlily <- quantMerge(snotes, 4, 4, 2)
qlily
```

Wave

Constructors and coercion for class Wave objects

Description

Constructors and coercion for class Wave objects

Usage

```
Wave(left, ...)
## S4 method for signature 'numeric':
Wave(left, right = numeric(0), samp.rate = 44100, bit = 16, ...)
```

Arguments

left, right, samp.rate, bit

See Section “Slots”.

... Further arguments to be passed to the default method `Wave.default`.

Value

An object of [Wave-class](#).

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [writeWave](#), [readWave](#)

Examples

```
# constructing a Wave object (1 sec.) containing sinus sound with 440Hz:
x <- seq(0, 2*pi, length = 44100)
channel <- round(32000 * sin(440 * x))
Wobj <- Wave(left = channel)
Wobj

# or more easily:
Wobj <- sine(440, bit = 16)
```

Wave-class

Class Wave

Description

Class "Wave"

Objects from the Class

Objects can be created by calls of the form `new ("Wave", ...)`, or more conveniently using the function [Wave](#).

Slots

`left`: Object of class "numeric" representing the left channel.

`right`: Object of class "numeric" representing the right channel, NULL if mono.

`stereo`: Object of class "logical" indicating whether this is a stereo (two channels) or mono representation.

`samp.rate`: Object of class "numeric" - the sampling rate, e.g. 44100 for CD quality.

`bit`: Object of class "numeric", common is 16 for CD quality, or 8 for a rather rough representation.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave](#)

WaveForms

Create Wave Objects of Special Waveforms

Description

Create a [Wave](#) object of special waveform such as silence, (white/pink) noise, sawtooth, sine, and square.

Usage

```

noise(kind = c("white", "pink"), duration = samp.rate,
      samp.rate = 44100, bit = 1, stereo = FALSE,
      xunit = c("samples", "time"), ...)

sawtooth(freq, duration = samp.rate, from = 0, samp.rate = 44100, bit = 1,
         stereo = FALSE, xunit = c("samples", "time"), reverse = FALSE, ...)

silence(duration = samp.rate, from = 0, samp.rate = 44100, bit = 1,
        stereo = FALSE, xunit = c("samples", "time"), ...)

sine(freq, duration = samp.rate, from = 0, samp.rate = 44100, bit = 1,
     stereo = FALSE, xunit = c("samples", "time"), ...)

square(freq, duration = samp.rate, from = 0, samp.rate = 44100, bit = 1,
       stereo = FALSE, xunit = c("samples", "time"), up = 0.5, ...)

```

Arguments

kind	The kind of noise, either “white” or “pink” (the latter is not dB adjusted (!) but linear decreasing on a log-log scale).
freq	The frequency (in Hertz) to be generated.
duration	Duration of the Wave in xunit.
from	Starting value of the Wave in xunit.
samp.rate	Sampling rate of the Wave.
bit	Resolution of the Wave and rescaling unit. This may be 1 (default) for rescaling to real values in [-1,1], 8 (i.e. 8-bit) for rescaling to integers in [0, 254], 16 (i.e. 16-bit) for rescaling to integers in [-32767, 32767], 24 (i.e. 24-bit) for rescaling to integers in [-8388607, 8388607], 32 (i.e. 32-bit) for rescaling to integers in [-2147483647, 2147483647], and 0 for not rescaling at all. These numbers are internally passed to normalize . The Wave slot bit will be set to 8, if bit=8, and to 16 otherwise.
stereo	Logical, if TRUE, a stereo sample will be generated. The right channel is identical to the left one for sawtooth, silence, sine, and square. For noise, both channel are independent.
xunit	Character indicating which units are used (both in arguments duration and from). If xunit = "time", the unit is time in seconds, otherwise the number of samples.
reverse	Logical, if TRUE, the waveform will be mirrored vertically.
up	A number between 0 and 1 giving the percentage of the waveform at max value (= 1 - percentage of min value).
...	Further arguments to be passed to Wave through the internal function <code>postWaveform</code> .

Value

A [Wave](#) object.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>, partly based on code from Matthias Heymann's former package 'sound', code for pink noise adapted and simplified from C code of Steve Moshier.

See Also

[Wave-class](#), [Wave](#), [normalize](#), [noSilence](#)

Examples

```
Wobj <- sine(440, bit = 16, duration = 1000)
Wobj2 <- noise(bit = 16, duration = 1000)
plot(Wobj)
plot(Wobj2)
```

WaveIO

Reading and writing Wave files

Description

Reading and writing Wave files.

Usage

```
readWave(filename, from = 1, to = Inf,
          units = c("samples", "seconds", "minutes", "hours"), header = FALSE)
writeWave(object, filename)
```

Arguments

filename	Filename of the file to be read or written.
from	where to start reading (in order to save memory by reading wave file piecewise), in units.
to	where to stop reading (in order to save memory by reading wave file piecewise), in units.
units	units in which from and to is given, the default is "samples", but can be set to time intervals such as "seconds", see the Usage Section above.
header	if TRUE, just header information of the Wave file are returned, otherwise (the default) the whole Wave object.
object	Object of class Wave to be written to a Wave file.

Value

`readWave` returns an object of class `Wave` or a list containing just the header information if `header = TRUE`.

`writeWave` creates a Wave file, but returns nothing.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [Wave](#), [normalize](#)

Examples

```
Wobj <- sine(440, bit = 16)

tdir <- tempdir()
tfile <- file.path(tdir, "myWave.wav")
writeWave(Wobj, filename = tfile)
list.files(tdir, pattern = "\.wav$")
newWobj <- readWave(tfile)
newWobj
file.remove(tfile)
```

WavPlayer

Getting and setting the default player for Wave files

Description

Getting and setting the default player for Wave files

Usage

```
setWavPlayer(player)
getWavPlayer()
```

Arguments

`player` Set the character string to call a Wave file player (including optional arguments) using `options`.

Value

`getWavPlayer` returns the character string that has been set by `setWavPlayer`.

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

[Wave-class](#), [Wave](#), [play](#)

Wspec-class

Class Wspec

Description

Class “Wspec” (*Wave spectrums*). Objects of this class represent a bunch of periodograms (see [periodogram](#), each generated by [spectrum](#)) corresponding to one or several windows of one [Wave](#) object. Redundancy (e.g. same frequencies in each of the periodograms) will be omitted, hence reducing memory consumption.

Details

The subset function “[” extracts the selected elements of slots `spec`, `starts`, `variance` and `energy` and returns the other slots unchanged.

Objects from the Class

Objects can be created by calls of the form `new("Wspec", ...)`, but regularly they will be created by calls to the function [periodogram](#).

Slots

The following slots are defined. For details see the constructor function [periodogram](#).

`freq`: Object of class "numeric"
`spec`: Object of class "list"
`kernel`: Object of class "ANY"
`df`: Object of class "numeric"
`taper`: Object of class "numeric"
`width`: Object of class "numeric"
`overlap`: Object of class "numeric"
`normalize`: Object of class "logical"
`starts`: Object of class "numeric"
`stereo`: Object of class "logical"
`samp.rate`: Object of class "numeric"
`variance`: Object of class "numeric"
`energy`: Object of class "numeric"

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

- the `show`, `plot` and `summary` methods,
- for the constructor function and some examples: `periodogram` (and hence also `spec.pgram`, `Wave-class`, and `Wave`)
- `WspecMat` for a similar class that represents the spectrum in form of a matrix.

WspecMat-class *Class WspecMat*

Description

Class “WspecMat” (*Wave spectrums as Matrix*). Objects of this class represent a bunch of periodograms (see `periodogram`, each generated by `spectrum`) corresponding to one or several windows of one `Wave` object. Redundancy (e.g. same frequencies in each of the periodograms) will be omitted, hence reducing memory consumption.

Details

The subset function “[” extracts the selected elements of slots `spec`, `starts`, `variance` and `energy` and returns the other slots unchanged.

Objects from the Class

Objects can be created by calls of the form `new("WspecMat", ...)`, but regularly they will be created from a `Wspec` object by calls such as `as(Wspec_Object, "WspecMat")`.

Slots

The following slots are defined. For details see the constructor function `periodogram`.

```
freq: Object of class "numeric"
spec: Object of class "matrix"
kernel: Object of class "ANY"
df: Object of class "numeric"
taper: Object of class "numeric"
width: Object of class "numeric"
overlap: Object of class "numeric"
normalize: Object of class "logical"
starts: Object of class "numeric"
stereo: Object of class "logical"
samp.rate: Object of class "numeric"
variance: Object of class "numeric"
energy: Object of class "numeric"
```

Author(s)

Uwe Ligges, <ligges@statistik.tu-dortmund.de>

See Also

the `show`, `plot` and `summary` methods

[-methods

Extract or Replace Parts of an Object

Description

Operators act on objects to extract or replace subsets.

See Also

[Extract](#) for the S3 generic.

Index

- *Topic **IO**
 - play-methods, 21
 - WaveIO, 36
- *Topic **aplot**
 - plot, 22
 - plot-Wave, 22
- *Topic **arith**
 - Arith-methods, 1
- *Topic **classes**
 - Wave-class, 34
 - Wspec-class, 38
 - WspecMat-class, 39
- *Topic **datagen**
 - Waveforms, 34
- *Topic **documentation**
 - tuneR, 31
- *Topic **error**
 - equalWave, 4
- *Topic **file**
 - lilyinput, 8
 - WaveIO, 36
- *Topic **hplot**
 - melodyplot, 10
 - plot, 22
 - plot-Wave, 22
 - plot-Wspec, 23
 - plot-WspecMat, 24
 - quantplot, 27
- *Topic **interface**
 - lilyinput, 8
 - play-methods, 21
- *Topic **iplot**
 - extractWave, 5
- *Topic **manip**
 - bind, 2
 - channel, 3
 - downsample, 3
 - extractWave, 5
 - Mono-Stereo, 13
 - normalize, 14
 - noSilence, 15
 - panorama, 17
 - prepComb, 25
- *Topic **methods**
 - [-methods, 40
 - Arith-methods, 1
 - length, 8
 - play-methods, 21
 - plot, 22
 - plot-Wave, 22
 - plot-Wspec, 23
 - plot-WspecMat, 24
 - show-WaveWspec-methods, 29
 - summary-methods, 30
 - Wave, 33
- *Topic **misc**
 - smoother, 30
- *Topic **print**
 - show-WaveWspec-methods, 29
 - summary-methods, 30
- *Topic **ts**
 - FF, 6
 - MFCC, 11
 - periodogram-methods, 18
 - smoother, 30
- *Topic **utilities**
 - bind, 2
 - channel, 3
 - downsample, 3
 - equalWave, 4
 - extractWave, 5
 - Mono-Stereo, 13
 - noSilence, 15
 - noteFromFF, 16
 - notenames, 17
 - play-methods, 21
 - prepComb, 25
 - quantize, 26

- WavPlayer, 37
- [, ANY-method (*[-methods]*), 40
- [, Wave-method (*Wave*), 33
- [, Wspec-method (*Wspec-class*), 38
- [, WspecMat-method (*WspecMat-class*), 39
- [-methods, 40
- abline, 11, 28
- Arith, numeric, Wave-method (*Arith-methods*), 1
- Arith, Wave, missing-method (*Arith-methods*), 1
- Arith, Wave, numeric-method (*Arith-methods*), 1
- Arith, Wave, Wave-method (*Arith-methods*), 1
- Arith-methods, 1
- axis, 11, 28
- bind, 2, 6, 26, 31
- channel, 3, 6, 31
- coerce, data.frame, Wave-method (*Wave*), 33
- coerce, list, Wave-method (*Wave*), 33
- coerce, matrix, Wave-method (*Wave*), 33
- coerce, numeric, Wave-method (*Wave*), 33
- coerce, Wave, data.frame-method (*Wave*), 33
- coerce, Wave, list-method (*Wave*), 33
- coerce, Wave, matrix-method (*Wave*), 33
- coerce, Wspec, WspecMat-method (*WspecMat-class*), 39
- decmedian, 30
- downsample, 3, 19, 31
- equalWave, 2, 4
- Extract, 40
- extractWave, 2, 3, 5, 15, 26, 31
- FF, 6, 11, 16, 29, 31
- FFpure, 31
- FFpure (*FF*), 6
- getWavPlayer (*WavPlayer*), 37
- groupGeneric, 2
- image, 24, 25
- image, ANY-method (*plot-WspecMat*), 24
- image, Wspec-method (*plot-WspecMat*), 24
- image-methods (*plot-WspecMat*), 24
- image-Wspec (*plot-WspecMat*), 24
- interactive, 5
- length, 8, 8
- length, ANY-method (*length*), 8
- length, Wave-method (*length*), 8
- length-methods (*length*), 8
- lilyinput, 8, 27, 31
- lines, 11, 28
- melodyplot, 10, 29, 31
- MFCC, 11
- mono, 3, 6, 19, 31
- mono (*Mono-Stereo*), 13
- Mono-Stereo, 13
- mtext, 11, 28
- noise (*Waveforms*), 34
- normalize, 14, 35–37
- noSilence, 15, 26, 36
- noteFromFF, 7, 10, 11, 16, 26, 27, 29–31
- notenames, 17, 27
- options, 37
- panorama, 17
- par, 11, 22, 28
- periodogram, 7, 16, 24, 25, 29, 31, 38, 39
- periodogram (*periodogram-methods*), 18
- periodogram, character-method (*periodogram-methods*), 18
- periodogram, Wave-method (*periodogram-methods*), 18
- periodogram-methods, 18
- play, 31, 38
- play (*play-methods*), 21
- play, character-method (*play-methods*), 21
- play, Wave-method (*play-methods*), 21
- play-methods, 21

- plot, [22](#)
- plot, ANY, ANY-method (*plot*), [22](#)
- plot, Wave, missing-method (*plot-Wave*), [22](#)
- plot, Wspec, missing-method (*plot-Wspec*), [23](#)
- plot, WspecMat, missing-method (*plot-WspecMat*), [24](#)
- plot-methods, [5](#), [29](#), [31](#)
- plot-Wspec, [20](#)
- plot-methods (*plot*), [22](#)
- plot-Wave, [22](#)
- plot-Wspec, [23](#)
- plot-WspecMat, [24](#)
- plot.default, [22](#)–[24](#)
- plot.Wave.channel (*plot-Wave*), [22](#)
- points, [28](#)
- prepComb, [2](#), [25](#)

- quantize, [9](#), [26](#), [28](#), [29](#), [31](#)
- quantMerge, [9](#), [31](#)
- quantMerge (*quantize*), [26](#)
- quantplot, [9](#), [11](#), [27](#), [27](#), [31](#)

- readWave, [31](#), [33](#)
- readWave (*WaveIO*), [36](#)
- rect, [11](#), [28](#)
- round, [16](#)

- sawtooth (*Waveforms*), [34](#)
- setWavPlayer, [21](#)
- setWavPlayer (*WavPlayer*), [37](#)
- show, Wave-method (*show-WaveWspec-methods*), [29](#)
- show, Wspec-method (*show-WaveWspec-methods*), [29](#)
- show, WspecMat-method (*show-WaveWspec-methods*), [29](#)
- show-WaveWspec-methods, [29](#)
- silence, [15](#)
- silence (*Waveforms*), [34](#)
- sine, [31](#)
- sine (*Waveforms*), [34](#)
- smoother, [30](#), [31](#)
- spec.pgram, [19](#), [20](#), [39](#)
- spectrum, [19](#), [31](#), [38](#), [39](#)
- square (*Waveforms*), [34](#)
- stereo, [2](#), [31](#)
- stereo (*Mono-Stereo*), [13](#)
- stop, [4](#)
- summary, ANY-method (*summary-methods*), [30](#)
- summary, Wave-method (*summary-methods*), [30](#)
- summary, Wspec-method (*summary-methods*), [30](#)
- summary, WspecMat-method (*summary-methods*), [30](#)
- summary-methods, [29](#)
- summary-methods, [30](#)
- summary.default, [31](#)

- tuneR, [7](#), [9](#), [11](#), [16](#), [23](#)–[25](#), [27](#), [29](#), [31](#)
- tuneR-package (*tuneR*), [31](#)

- Wave, [1](#)–[6](#), [8](#), [12](#)–[15](#), [18](#)–[23](#), [25](#), [26](#), [29](#), [31](#), [33](#), [34](#)–[39](#)
- Wave, ANY-method (*Wave*), [33](#)
- Wave, data.frame-method (*Wave*), [33](#)
- Wave, list-method (*Wave*), [33](#)
- Wave, matrix-method (*Wave*), [33](#)
- Wave, numeric-method (*Wave*), [33](#)
- Wave-class, [2](#)–[4](#), [6](#), [13](#)–[15](#), [18](#), [20](#), [21](#), [23](#), [26](#), [29](#), [31](#), [33](#), [34](#), [36](#)–[39](#)
- Wave-methods (*Wave*), [33](#)
- Waveforms, [34](#)
- WaveIO, [36](#)
- WavPlayer, [37](#)
- writeWave, [14](#), [21](#), [31](#), [33](#)
- writeWave (*WaveIO*), [36](#)
- Wspec, [6](#), [7](#), [10](#), [19](#), [20](#), [23](#)–[26](#), [29](#), [31](#), [39](#)
- Wspec (*Wspec-class*), [38](#)
- Wspec-class, [38](#)
- WspecMat, [24](#), [25](#), [29](#), [31](#), [39](#)
- WspecMat (*WspecMat-class*), [39](#)
- WspecMat-class, [39](#)