Package ‘tuneR’

April 17, 2024

Version 1.4.7
Date 2024-04-16
Title Analysis of Music and Speech
Depends R (>= 3.0.0)
Encoding UTF-8
Suggests pastecs
Imports signal, methods
Description Analyze music and speech, extract features like MFCCs, handle wave files and their representation in various ways, read mp3, read midi, perform steps of a transcription, ...
Also contains functions ported from the 'rastamat' 'Matlab' package.
License GPL-2 | GPL-3
NeedsCompilation yes
Author Uwe Ligges [aut, cre, cph] (<https://orcid.org/0000-0001-5875-6167>),
Sebastian Krey [aut, cph],
Olaf Mersmann [aut, cph],
Sarah Schnackenberg [aut, cph],
Guillaume Guénard [aut, cph] (for the 'pulse' functionality),
Daniel P. W. Ellis [aut, cph] (functions ported from 'rastamat'),
Underbit Technologies [aut, cph] (for the included 'libmad MPEG audio decoder library'),
Andrea Preusser [ctb],
Anita Thieler [ctb],
Johanna Mielke [ctb],
Claus Weihs [ctb],
Brian D. Ripley [ctb],
Matthias Heymann [ctb] (for ideas from the former 'sound' package)
Maintainer Uwe Ligges <ligges@statistik.tu-dortmund.de>
Repository CRAN
Date/Publication 2024-04-17 14:08:24
R topics documented:

- Arith-methods ...................................................... 3
- audspec ............................................................. 3
- bind ................................................................. 5
- channel .............................................................. 5
- deltas ................................................................. 6
- dolpc ................................................................. 7
- downsample ......................................................... 8
- equalWave .......................................................... 9
- extractWave ......................................................... 9
- FF ..................................................................... 11
- freqconv ........................................................... 12
- getMidiNotes ....................................................... 13
- length ............................................................... 14
- lifter ................................................................. 15
- lilyinput ............................................................ 16
- lpc2cep ............................................................. 17
- MCnames ........................................................... 18
- melfcc ............................................................... 19
- melodyplot ......................................................... 21
- Mono-Stereo ....................................................... 23
- nchannel ............................................................ 24
- normalize-methods ............................................... 24
- noSilence .......................................................... 25
- noteFromFF ......................................................... 26
- notenames .......................................................... 27
- panorama ........................................................... 28
- periodogram-methods ............................................. 29
- play-methods ....................................................... 32
- plot-Wave .......................................................... 33
- plot-Wspec ........................................................ 34
- plot-WspecMat ..................................................... 35
- postaud ............................................................. 36
- powspec ............................................................ 37
- prepComb ........................................................... 38
- quantize ............................................................. 39
- quantplot ........................................................... 40
- readMidi ............................................................ 42
- readMP3 ........................................................... 43
- readWave .......................................................... 44
- show-WaveWspec-methods ........................................ 45
- smoother .......................................................... 46
- spec2cep ............................................................ 47
- summary-methods ................................................ 48
- tuneR ............................................................... 48
- updateWave ......................................................... 50
- Wave ............................................................... 51
Arith-methods

Description

Methods for arithmetics on Wave and WaveMC objects

Methods

object = "Wave"  An object of class Wave.
object = "WaveMC"  An object of class WaveMC.
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, WaveMC-class, WaveMC

audspec  

Frequency band conversion

Description

Perform critical band analysis (see PLP), which means the reduction of the fourier frequencies of a signal’s powerspectrum to a reduced number of frequency bands in an auditory frequency scale.

Usage

audspec(pspecrum, sr = 16000, nfilts = ceiling(hz2bark(sr/2)) + 1, ftype = c("bark", "mel", "htkmel", "fcmel"), minfreq = 0, maxfreq = sr/2, sumpower = TRUE, bwidth = 1)
Arguments

pspectrum Output of \texttt{powspec}, matrix with the powerspectrum of each time frame in its columns.

sr Sample rate of the original recording.

nfilts Number of filters/frequency bins in the auditory frequency scale.

fbtype Used auditory frequency scale.

minfreq Lowest frequency.

maxfreq Highest frequency.

sumpower If \texttt{sumpower = TRUE}, the frequency scale transformation is based on the powerspectrum, if \texttt{sumpower = FALSE}, it is based on its squareroot (absolute value of the spectrum) and squared afterwards.

bwidth Modify the width of the frequency bands.

Value

aspectrum Matrix with the auditory spectrum of each time frame in its columns.

wts Weight matrix for the frequency band conversion.

Author(s)

Sebastian Krey <krey@statistik.tu-dortmund.de>

References

Daniel P. W. Ellis: \url{https://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/}

See Also

\texttt{fft2melmx, fft2barkmx}

Examples

\begin{verbatim}
testsound <- normalize(sin(400) + sin(1000) + square(250), "16")
pspectrum <- powspec(testsound@left, testsound@samp.rate)
aspectrum <- audspec(psppectrum, testsound@samp.rate)
\end{verbatim}
bind

**Concatenating Wave objects**

**Description**

Generic function for concatenating objects of class Wave or WaveMC.

**Usage**

```r
bind(object, ...)
```

**Arguments**

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

**Value**

An object of class `Wave` or class `WaveMC` that corresponds to the class of the input.

**Author(s)**

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

**See Also**

`prepComb` for preparing the concatenation, `Wave-class`, `Wave`, `WaveMC-class`, `WaveMC`, `extractWave`, `stereo`

channel

**Channel conversion for Wave objects**

**Description**

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

**Usage**

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

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

Details

For objects of `WaveMC-class`, channel selection can be performed by simple matrix indexing, e.g. `WaveMCobject[,2]` selects the second channel.

Value

Wave object including channels specified by `which`.

Author(s)

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

See Also

`Wave`, `Wave-class`, `mono`, `extractWave`

deltas

Calculate delta features

Description

Calculate the deltas (derivatives) of a sequence of features using a w-point window with a simple linear slope.

Usage

deltas(x, w = 9)

Arguments

- **x**: Matrix of features. Every column represents one time frame. Each row is filtered separately.
- **w**: Window width (usually odd).

Details

This function mirrors the delta calculation performed in HTKs ‘feacalc’.

Value

Returns a matrix of the delta features (one column per frame).
**Author(s)**
Sebastian Krey <krey@statistik.tu-dortmund.de>

**References**

**Examples**
```r
testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
m <- melfcc(testsound, frames_in_rows=FALSE)
d <- deltas(m)
```

---

**dolpc**

*(Perceptive) Linear Prediction*

**Description**
Compute autoregressive model from spectral magnitude samples via Levinson-Durbin recursion.

**Usage**
```r
dolpc(x, modelorder = 8)
```

**Arguments**
- `x` Matrix of spectral magnitude samples (each sample/time frame in one column).
- `modelorder` Lag of the AR model.

**Value**
Returns a matrix of the normalized AR coefficients (depending on the input spectrum: LPC or PLP coefficients). Every column represents one time frame.

**Author(s)**
Sebastian Krey <krey@statistik.tu-dortmund.de>

**References**

**See Also**
- `levinson`
Examples

testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
pспектrum <- powspec(testsound@left, testsound@samp.rate)
aspectrum <- audspec(pспектrum, testsound@samp.rate)$aspectrum
lpcas <- dolpc(aspectrum, 10)

downsample  

Description

Downsampling an object of class Wave or class WaveMC.

Usage

downsample(object, samp.rate)

Arguments

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

Value

An object of class Wave or class WaveMC.

Author(s)

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

See Also

Wave-class, Wave, WaveMC-class, WaveMC
equalWave

Description

Internal S4 generic function that checks for some kind of equality of objects of class Wave or class WaveMC.

Usage

equalWave(object1, object2)

Arguments

object1, object2

Object(s) of class Wave or class WaveMC (both of the same class).

Value

Does not return anything. It stops code execution with an error message indicating the problem if the objects are not of the same class (either Wave oder WaveMC) or if the two objects don’t have the same properties, i.e. identical sampling rate, resolution (bit), and number of channels (for WaveMC, resp. stereo/mono for Wave).

Author(s)

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

See Also

Wave-class, Wave, WaveMC-class, WaveMC

extractWave

Extractor for Wave and WaveMC objects

Description

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

Usage

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

equalWave
Checking Wave objects

extractWave
Extractor for Wave and WaveMC objects
Arguments

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

Details

This function allows interactive selection of a range to be extracted from an object of class `Wave` or class `WaveMC`. The default is to use interactive selection if the current R session is interactive. In case of interactive selection, `plot-methods` plot the `Wave` or `WaveMC` 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` or `WaveMC` 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` or class `WaveMC`.

Author(s)

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

See Also

`Wave-class`, `Wave`, `WaveMC-class`, `WaveMC`, `bind`, `channel`, `mono`

Examples

```r
Wobj <- sine(440)
# 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:
```
FF

Estimation of Fundamental Frequencies from a Wspec object

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

```r
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

- **object**: An object of class Wspec.
- **peakheight**: 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.
- **silence**: 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.
- **minpeak**: If more than minpeak peaks are considered for detection and passed argument peakheight, such periodograms are detected to be silence or noise (if silence > 0).
- **diapason**: Frequency of diapason a, default is 440 (Hertz).
- **notes**: 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.
- **interest.frqs**: Optional, either a vector of integers indicating the indices of (fundamental) frequencies in object 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 \( \text{peakheight} \times \' \text{largest_peak}' \) is taken.
2. Its frequency is multiplied by \( 1 + \text{search.par}[1] \). Now, any larger peak between the old peak and that value is taken, if (a) it exists and if (b) it is above the \( \text{search.par}[2] \)-th Fourier-Frequency.
3. Within the interval of frequencies \( \text{current peak} \times \text{search.par}[3:4] \), 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

\( \text{FFpure} \) just estimates the fundamental frequencies for all periodograms contained in the object (of class \text{Wspec}).

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

\text{Wspec}, \text{periodogram} (including an example), \text{noteFromFF}, and \text{tuneR} for a very complete example.

---

**freqconv**

*Frequency scale conversion*

Description

Perform frequency scale conversions between Hertz, Bark- and different variants of the Melscale.

Usage

- \text{bark2hz(z)}
- \text{hz2bark(f)}
- \text{hz2mel(f, htk = FALSE)}
- \text{mel2hz(z, htk = FALSE)}
getMidiNotes

Arguments

- `f` Frequency in Hertz
- `z` Frequency in the auditory frequency scale
- `htk` Use the HTK-Melscale (htk = TRUE) or Slaney’s Melscale from the Auditory Toolbox (htk = FALSE)

Value

The value of the input in the target frequency scale.

Author(s)

Sebastian Krey <krey@statistik.tu-dortmund.de>

References


Examples

```r
hz2bark(440)
bark2hz(hz2bark(440))
hz2mel(440, htk = TRUE)
me2hz(hz2mel(440, htk = TRUE), htk = TRUE)
hz2mel(440, htk = FALSE)
me2hz(hz2mel(440, htk = FALSE), htk = FALSE)
```

---

getMidiNotes | Extract note events from objects returned by readMidi

Description

Extract only note events from an object returned by the readMidi function.

Usage

```r
getMidiNotes(x, ...)
```

Arguments

- `x` A data.frame returned by the readMidi function.
- `...` Further arguments are passed to the notenames function for extracting the human readable note names rather than their integer representations.
Value

A data frame with columns

- time: start time
- length: length
- track: track number
- channel: channel number
- note: note
- notename: notename
- velocity: note velocity

Author(s)

Uwe Ligges and Johanna Mielke

See Also

readMidi

Examples

```r
content <- readMidi(system.file("example_files", "Bass_sample.mid", package="tuneR"))
getMidiNotes(content)
```

<table>
<thead>
<tr>
<th>length</th>
<th>S4 generic for length</th>
</tr>
</thead>
</table>

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.
- **x = "WaveMC"** The length for each of the time series in the `WaveMC` will be returned.
- **object = "ANY"** For compatibility.

See Also

For the primitive: `length`
**lifter**  

**Lifting of cepstra**

**Description**

Apply liftering to a matrix of cepstra.

**Usage**

lifter(x, lift = 0.6, inv = FALSE, htk = FALSE)

**Arguments**

- **x**: Matrix of cepstra, one sample/time frame per column.
- **lift**: Liftering exponent/length.
- **inv**: Invert the liftering (undo a previous liftering).
- **htk**: Switch liftering type.

**Details**

If htk = FALSE, then perform $x_i^{lift}$, $i = 1, \ldots, \text{nrow}(x)$ liftering. If htk = TRUE, then perform HTK-style sin-curve liftering with length lift.

**Value**

Matrix of the liftered cepstra.

**Author(s)**

Sebastian Krey <krey@statistik.tu-dortmund.de>

**References**


**Examples**

```r
testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
m <- melfcc(testsound, frames_in_rows=FALSE)
unlm <- lifter(m, inv=TRUE)
```
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):
  • note: Integer - the notes' pitch in halftones from diapason (a), i.e. 0 for
diapason a, 3 for c', ...
  • duration: 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 LilyPond'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 LilyPond) is desirable.
tempo Character specifying the tempo to be used for the Midi file if midi = TRUE. 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 LilyPond.
linewidth Linewidth of the sheet music to be written by LilyPond.
indent Indentation of the sheet music to be written by LilyPond.
fontsize Fontsize of the sheet music to be written by LilyPond.
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

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.

---

lpc2cep  
LPC to cepstra conversion

Description
Convert the LPC coefficients in each column of a into frames of cepstra.

Usage
lpc2cep(a, nout = nrow(a))

Arguments
a  
Matrix of LPC coefficients.
nout  
Number of cepstra to produce.

Value
Matrix of cepstra (one column per time frame).
Author(s)
Sebastian Krey <krey@statistik.tu-dortmund.de>

References

See Also
spec2cep

Examples
```
testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
pspectrum <- powspec(testsound@left, testsound@samp.rate)
aspectrum <- audspec(pspectrum, testsound@samp.rate)
lpcas <- dolpc(aspectrum$aspectrum, 8)
cepstra <- lpc2cep(lpcas)
```

---

**MCnames**

*Default channel ordering for multi channel wave files*

Description
A data frame representing the default channel ordering with id, descriptive label, and abbreviated name for multi channel wave files.

Format
A data frame with 18 observations on the following 3 variables:
- id: id of the channel
- label: full label for the channel
- name: abbreviated name for the channel

Source
Data derived from the technical documentation given at [https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-waveformatextensible](https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-waveformatextensible).

References

Examples
MCnames # the 18 predefined channels in a multi channel Wave file (WaveMC object)
melfcc  

**MFCC Calculation**

**Description**

Calculate Mel-frequency cepstral coefficients.

**Usage**

```r
melfcc(samples, sr = samples@samp.rate, wintime = 0.025,
       hoptime = 0.01, numcep = 12, lifterexp = 0.6, htklifter = FALSE,
       sumpower = TRUE, preemph = 0.97, dither = FALSE,
       minfreq = 0, maxfreq = sr/2, nbands = 40, bwidth = 1,
       dcttype = c("t2", "t1", "t3", "t4"),
       fbtype = c("mel", "htkmel", "fcmel", "bark"), usecmp = FALSE,
       modelorder = NULL, spec_out = FALSE, frames_in_rows = TRUE)
```

**Arguments**

- **samples**: Object of Wave-class or WaveMC-class. Only the first channel will be used.
- **sr**: Sampling rate of the signal.
- **wintime**: Window length in sec.
- **hoptime**: Step between successive windows in sec.
- **numcep**: Number of cepstra to return.
- **lifterexp**: Exponent for liftering; 0 = none.
- **htklifter**: Use HTK sin lifter.
- **sumpower**: If `sumpower = TRUE` the frequency scale transformation is based on the power-spectrum, if `sumpower = FALSE` it is based on its squareroot (absolute value of the spectrum) and squared afterwards.
- **preemph**: Apply pre-emphasis filter \([1 - \text{preemph}]\) (0 = none).
- **dither**: Add offset to spectrum as if dither noise.
- **minfreq**: Lowest band edge of mel filters (Hz).
- **maxfreq**: Highest band edge of mel filters (Hz).
- **nbands**: Number of warped spectral bands to use.
- **bwidth**: Width of spectral bands in Bark/Mel.
- **dcttype**: Type of DCT used - 1 or 2 (or 3 for HTK or 4 for feacalc).  
- **fbtype**: Auditory frequency scale to use: "mel", "bark", "htkmel", "fcmel".
- **usecmp**: Apply equal-loudness weighting and cube-root compression (PLP instead of LPC).
- **modelorder**: If `modelorder > 0`, fit a linear prediction (autoregressive-) model of this order and calculation of cepstra out of lpcas.
- **spec_out**: Should matrices of the power- and the auditory-spectrum be returned.
- **frames_in_rows**: Return time frames in rows instead of columns (original Matlab code).
Details

Calculation of the MFCCs includes the following steps:

1. Preemphasis filtering
2. Take the absolute value of the STFT (usage of Hamming window)
3. Warp to auditory frequency scale (Mel/Bark)
4. Take the DCT of the log-auditory-spectrum
5. Return the first `ncep` components

Value

cepstra  Cepstral coefficients of the input signal (one time frame per row/column)
aspectrum  Auditory spectrum (spectrum after transformation to Mel/Bark scale) of the signal
psspectrum  Power spectrum of the input signal.
lpcas  If `modelorder > 0`, the linear prediction coefficients (LPC/PLP).

Note

The following non-default values nearly duplicate Malcolm Slaney’s `mfcc` (i.e. `melfcc(d, 16000, wintime=0.016, lifterexp=0, minfreq=133.33, maxfreq=6855.6, sumpower=False) = \log(10) * 2 * mfcc(d, 16000)` in the Auditory toolbox for Matlab).

The following non-default values nearly duplicate HTK’s MFCC (i.e. `melfcc(d, 16000, lifterexp=22, htklifter=True, nbands=20, maxfreq=8000, sumpower=False, fbtype="htkmel", dcttype="t3") = 2 * htkmelfcc(:,[13,[1:12]])` where HTK config has ‘PREEMCOEF = 0.97’, ‘NUM-CHANS = 20’, ‘CEPLIFTER = 22’, ‘NUMCEPS = 12’, ‘WINDOWSIZE = 250000.0’, ‘USE-HAMMING = T’, ‘TARGETKIND = MFCC_0’).

For more detail on reproducing other programs’ outputs, see https://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/mfccs.html

Author(s)

Sebastian Krey <krey@statistik.tu-dortmund.de>

References

Daniel P. W. Ellis: https://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/
Examples

```r
testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
m1 <- melfcc(testsound)
```

```r
#Use PLP features to calculate cepstra and output the matrices like the
#original Matlab code (note: modelorder limits the number of cepstra)
m2 <- melfcc(testsound, numcep=9, usecmp=TRUE, modelorder=8,
    spec_out=TRUE, frames_in_rows=FALSE)
```

---

**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**

```r
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 `observed`.
- **bars**: Number of bars to be plotted (a virtual static segmentation takes place). If `NULL` (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 ylim 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 \texttt{par} for details.
las Orientation of axis labels, see \texttt{par} for details.
cex.axis Size of tick mark labels, see \texttt{par} for details.
mar Margins of the plot, see \texttt{par} for details.
notenames Optionally specify other notenames (character) for the y axis.
thin Amount of thinning of notenames, i.e. only each \texttt{thin}th notename is displayed on the y-axis.
silence Character string for label of the ‘silence’ (default) axis.
plotenergy Logical (default: \texttt{TRUE}), whether to plot energy values in the bottom part of the plot.

\ldots Additional graphical parameters to be passed to underlying \texttt{plot} function.

axispar A named list of three other lists (ax1, ax2, and ax4) containing parameters passed to the corresponding \texttt{axis} calls for the three axis time (ax1), notes (ax2), and energy (ax4).

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

\textbf{Author(s)}

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

\textbf{See Also}

\texttt{noteFromFF, FF, quantplot}; for an example, see the help in \texttt{tuneR}.
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

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

Details

For objects of `WaveMC-class`, a mono channel can be created by simple matrix indexing, e.g. `WaveMObject[,2]` selects the second channel.

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

```r
Wobj <- sine(440)
Wobj
Wobj2 <- stereo(Wobj, Wobj)
Wobj2
mono(Wobj2, "right")
```

### nchannel

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the number of channels from a Wave or WaveMC object</td>
</tr>
</tbody>
</table>

#### Usage

```r
nchannel(object)
```

#### Arguments

- `object`: Object of class `Wave` or class `WaveMC`.

#### Value

An integer, the number of channels given in the object.

### See Also

`Wave-class`, `WaveMC-class`

### normalize-methods

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rescale the range of values</td>
</tr>
</tbody>
</table>

#### Usage

```r
normalize(object, unit = c("1", "8", "16", "24", "32", "64", "0"),
          center = TRUE, level = 1, rescale = TRUE, pcm = object@pcm)
```
noSilence

Arguments

object Object of class Wave or WaveMC.
unit Unit to rescale to.
   "1" (default) for rescaling to numeric 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 either to integers in [-2147483647, 2147483647]
   (PCM Wave format if pcm=TRUE) or to numeric values in [-1, 1] (FLOAT_IEEE
   Wave format if pcm = FALSE),
   "64" (i.e. 64-bit) for rescaling to real values in [-1, 1] (FLOAT_IEEE Wave
   format), and
   "0" for not rescaling (hence only centering if center = TRUE).
center If TRUE (default), values are centered around 0 (or 127 if unit = "8").
level Maximal percentage of the amplitude used for normalizing (default is 1).
rescale Logical, whether to rescale to the maximal possible dynamic range.
pcm Logical. By default, the pcm information from the object is kept. Otherwise,
   if TRUE, the object is coerced to the PCM Wave format. If FALSE, the object is
   coerced to the FLOAT_IEEE format, i.e. numeric values in [-1, 1].

Value

An object containing the normalized data of the same class as the input object, i.e. either Wave or
WaveMC.

Author(s)

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

See Also

writeWave, Wave-class, Wave, WaveMC-class, WaveMC

Description

Generic function to cut off silence or low noise at the beginning and/or at the end of an object of
class Wave or class WaveMC.

Usage

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

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

Details

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

Value

An object of class `Wave` or `WaveMC`.

Author(s)

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

See Also

`silence`, `Wave-class`, `Wave`, `WaveMC-class`, `WaveMC`, `extractWave`
notenames

Arguments

x
Fundamental frequency.
diapason
Frequency of diapason a, default is 440 (Hertz).
roundshift
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 roundshift 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 \( x = 452 \) and diapason = 440, the internally calculated real value of 0.46583 is rounded to 0, but for roundshift = 0.1 we get 0.56583 and it is rounded to note 1.

Details

The formula used is simply \( \text{round}(12 \times \log(x / \text{diapason}, 2) + \text{roundshift}) \).

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 \( \text{diapason} \). 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

notes
An integer values vector, where 0 corresponds to a’, notes below and above have to be specified in the corresponding halftone distance.
language
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

```r
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

Generic function to narrow the panorama of a stereo Wave or WaveMC object.

Usage

`panorama(object, pan = 1)`

Arguments

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

Details

If `abs(pan) < 1`, mixtures of the two channels of the Wave or WaveMC objects are used for the left and the right channel of the returned Sample object if the object is of class `Wave`, resp. for the first and second channel of the returned Sample object if the object is of class `WaveMC`, so that they appear closer to the center.

For `pan = 0`, both sounds are completely in the center (i.e. averaged).

If `pan < 0`, the left and the right channel (for Wave objects, the first and the second channel for WaveMC objects) are interchanged.

Value

An object of class `Wave` or class `WaveMC` with the transformed panorama.
periodogram-methods

Author(s)

Uwe Ligges <ligges@statistik.tu-dortmund.de>, Sarah Schnackenberg, based on code by Matthias Heymann

See Also

Wave-class, Wave, WaveMC-class, WaveMC

Description

This function estimates one or more periodograms (spectral densities) of the time series contained in an object of class Wave or WaveMC (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 'WaveGeneral'
 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An object of class Wave, WaveMC, or a character string pointing to a Wave file.</td>
</tr>
<tr>
<td>width</td>
<td>A window of width ‘width’ running through the time series selects the samples from which the periodograms are to be calculated.</td>
</tr>
<tr>
<td>overlap</td>
<td>The window can be applied by each overlapping overlap samples.</td>
</tr>
<tr>
<td>starts</td>
<td>Start number (in samples) for a window. If not given, this value is derived from argument ends, or will be derived from width and overlap.</td>
</tr>
<tr>
<td>ends</td>
<td>End number (in samples) for a window. If not given, this value is derived from argument starts, or will be derived from width and overlap.</td>
</tr>
<tr>
<td>taper</td>
<td>proportion of data to taper. See spec.pgram for details.</td>
</tr>
<tr>
<td>normalize</td>
<td>Logical; if TRUE (default), two steps will be applied: (i) the input signal will be normalized to amplitude ( \max(\text{abs(ampplitude)}) = 1 ), (ii) the resulting spec values will be normalized to sum up to one for each periodogram.</td>
</tr>
</tbody>
</table>
periodogram-methods

frqRange 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.
from Where to start reading in the Wave file, in units.
to Where to stop reading in the Wave file, 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.
downsample Sampling rate the object is to be downsampled to. If NA, the default, no changes are applied. Otherwise downsample must be in [2000, 192000]; typical values are 11025, 22050, and 44100 for CD quality. See also downsample.
channel Character, indicating whether the “left” or “right” channel should be extracted (see mono for details) - stereo processing is not yet implemented.
pieces The Wave file will be read in in pieces steps in order to reduce the amount of required memory.
... Further arguments to be passed to the underlying function spec.pgram.

Value

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

freq Vector of frequencies at which the spectral density is estimated. See spectrum for details. (1)
spec List of vectors or matrices of the spec values returned by spec.pgram at frequencies corresponding to freq. Each element of the list corresponds to one periodogram estimated from samples of the window beginning at start of the Wave or WaveMC object.
kern The kernel argument, or the kernel constructed from spans returned by spec.pgram. (1)
df The distribution of the spectral density estimate can be approximated by a chi square distribution with df 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 Always FALSE (for back compatibility). (1)
samp.rate Sampling rate of the underlying Wave or WaveMC 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:

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

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

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 more than one channel of Wave or WaveMC 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, WaveMC-class, WaveMC.

Examples

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

# Calculate periodograms in windows of 4096 samples each - without
# any overlap - resulting in a 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")
# plot the spectrogram
image(Wspecobj, ylim=c(0, 2000))

# same again with normalize = FALSE and with logarithmic y-axis plotted:
Wspecobj2 <- periodogram(Wobj, width = 4096, normalize = FALSE)
Wspecobj2
### Description

Plays wave files and objects of class `Wave`.

### Usage

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

### Arguments

- **object**: Either a filename pointing to a Wave file, or an object of class `Wave` or `WaveMC`. If the latter, it is written to a temporary file by `writeWave`, played by the chosen player, and deleted afterwards.
- **player**: (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” or “wmplayer.exe” (if the former does not exists as under Windows 7) will be chosen as the default.
- **...**: Further arguments passed to the Wave file player. If no player and no further arguments are given under Windows, the default is: “/play /close”.

### Author(s)

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

### See Also

- `Wave-class`, `WaveMC-class`, `Wave`, `WaveMC`, `writeWave`, `setWavPlayer`
plot-Wave  Plotting Wave objects

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 = 2500, axes = TRUE, yaxt = par('yaxt'), las = 1,
     center = TRUE, ...)

## S4 method for signature 'WaveMC,missing'
plot(x, info = FALSE, xunit = c('time', 'samples'),
     ylim = NULL, main = NULL, sub = NULL, xlab = NULL, ylab = colnames(x),
     simplify = TRUE, nr = 2500, axes = TRUE, yaxt = par('yaxt'), las = 1,
     center = TRUE, mfrow = NULL, ...)

plot_Wave_channel(x, xunit, ylim, xlab, ylab, main, nr, simplify, axes = TRUE,
                   yaxt = par('yaxt'), las = 1, center = TRUE, ...)

Arguments

x Object of class Wave or WaveMC, respectively.
info Logical, whether to include (written) information on the Wave or WaveMC object
       within the plot.
xunit Character indicating which units are used for setting up user coordinates (see
       par) and x-axis labeling. If xunit = "time", the unit is time in seconds, otherwise
       the number of samples.
ylim The y (amplitude) limits of the plot.
main, sub A title / subtitle for the plot.
xlab Label for x-axis.
ylab Label for y-axis (on the right side of the plot). For WaveMC objects, this can
       be the default colnames(x) (i.e. channel names of the WaveMC object), NULL
       for “channel 1”, ..., “channel nc” where nc is ncol(x), NA for no labels, or a
       character vector of labels (one element for each channel). For Wave objects,
       this can be de default “left channel” (for mono) or “left channel” and “right channel”
       (for stereo), NA for no labels, or a character vector of labels (one element for each
       channel).
simplify Logical, whether the plot should be “simplified”. If TRUE (default), not all (thou-
       sand/millions/billions) of points (samples) of the Wave or WaveMC object are
drawn, but the nr (see below) ranges (in form of segments) within nr windows of the time series.
Plotting with simplify = FALSE may take several minutes (depending on the number of samples in the Wave or WaveMC) and output in any vector format may be really huge.

nr
Number of windows (segments) to be used approximately (an appropriate number close to nr is selected) to simplify (see above) the plot. Only used if simplify = TRUE and the number of samples of the Wave or WaveMC object x is larger.

axes
Whether to plot axes, default is TRUE.

yaxt
How to plot the y-axis ("n" for no y-axis).

las
The style of the axis labels, default is las = 1 (always horizontal), see par for details.

center
Whether to plot with y-axes centered around 0 (or 127 if 8-bit), default is TRUE.

mfrow
A vector indicating the arrangement of the figures, see par for details.

...
Further arguments to be passed to the underlying plot functions.

Details
Function plot_Wave_channel is a helper function to plot a single channel (left for a Wave object, first channel / first column of data slot of a WaveMC object); in particular it is not intended to be called by the user directly.

Author(s)
Uwe Ligges <ligges@statistik.tu-dortmund.de>, Sarah Schnackenberg

See Also
Wave-class, Wave, WaveMC-class, WaveMC 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

x Object of class \texttt{Wspec}.

which Integer indicating which of the periodograms contained in object \texttt{x} to plot. Default is to plot the first one.

type The default is to plot horizontal lines, rather than points. See \texttt{plot.default} for details.

xlabel, ylabel 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 \texttt{plot.default} for details.

Author(s)

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

See Also

see \texttt{Wspec}, \texttt{periodogram} and \texttt{tuneR} for the constructor function and some examples.

---

\textbf{plot-WspecMat} \quad \textit{Plotting WspecMat objects}

Description

Plotting a spectogram (image) of an object of class \texttt{Wspec} or \texttt{WspecMat}.

Usage

\begin{verbatim}
## S4 method for signature 'WspecMat,missing'
plot(x, xlabel = "time", ylabel = "frequency",
     xunit = c("samples", "time"), log = "", ...)

## S4 method for signature 'Wspec'
image(x, xlabel = "time", ylabel = "frequency",
       xunit = c("samples", "time"), log = "", ...)
\end{verbatim}

Arguments

x Object of class \texttt{WspecMat} (for \texttt{plot}) or \texttt{Wspec} (for \texttt{image}).

xlabel, ylabel Label for x-/y-axis.

xunit Character indicating which units are used to annotate the x-axis. If \texttt{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 \texttt{image} function. See \texttt{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.

---

**postaud**  
*Equal loudness compression*

Description

Do loudness equalization and cube root compression

Usage

```r
postaud(x, fmax, fbtype = c("bark", "mel", "htkmel", "fcmel"),
        broaden = FALSE)
```

Arguments

- `x` Matrix of spectra (output of `audspec`).
- `fmax` Maximum frequency im Hertz.
- `fbtype` Auditory frequency scale.
- `broaden` Use two additional frequency bands for calculation.

Value

- `x` Matrix of the per sample/frame (columns) spectra after applying the frequency dependant loudness equalization and compression.
- `eql` Vector of the equal loudness curve.

Author(s)

Sebastian Krey <krey@statistik.tu-dortmund.de>

References

powspec

See Also

audspec, dolpc

Examples

testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
pспектrum <- powspec(testsound@left, testsound@samp.rate)
aspectrum <- audspec(pспектrum, testsound@samp.rate)
pасспектрум <- postaud(x = aspectrum$aspectrum, fmax = 5000, fbtype = "mel")

Description

Compute the powerspectrum of the input signal. Basically output a power spectrogram using a Hamming window.

Usage

powspec(x, sr = 8000, wintime = 0.025, steptime = 0.01, dither = FALSE)

Arguments

x Vector of samples.
sr Sampling rate of the signal.
wintime Window length in sec.
steptime Step between successive windows in sec.
dither Add offset to spectrum as if dither noise.

Value

Matrix, where each column represents a power spectrum for a given frame and each row represents a frequency.

Author(s)

Sebastian Krey <krey@statistik.tu-dortmund.de>

References

Daniel P. W. Ellis: https://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/

See Also

specgram
Examples

testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
p spectrum <- powspec(testsound@left, testsound@samp.rate)

prepComb

Description
Preparing the combination/concatenation of Wave or WaveMC objects

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

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

Details
This function is useful to prepare objects of class Wave or class WaveMC for binding/combination/concatenation. At the side(s) indicated by where small amounts of the Wave or WaveMC are removed in order to make the transition between two Waves or WaveMCs smooth (avoiding clicks).
This is done by dropping all values at the beginning of a Wave or WaveMC before the first positive point after the zero level is crossed from negative to positive. Analogously, at the end of a Wave or WaveMC 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 or class WaveMC.

Note
If stereo (for Wave), only the left channel is analyzed while the right channel will simply be cut at the same locations. If multi channel (for WaveMC), only the first channel is analyzed while all other channels will simply be cut at the same locations.
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 outcoming 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.

Examples

Wobj1 <- sine(440, duration = 520)
Wobj2 <- extractWave(sine(330, duration = 500), 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")
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/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(),
energypar = list(text="energy", side=4, line=2.5, at=rg.s-0.25, las=3),
energylabel = list(text="energy", side=4, line=2.5, at=rg.s-0.25, las=3),
energypar = list(pch=20),

quantplot
quantplot

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 notes (observed notes) and energy (corresponding energy values), e.g. the result from a call to quantize.
energy A vector of energy values with same quantization as observed (overwrites any given energy values if observed is a list).
expected Expected notes (optional; in order to compare results).
bars Number of bars to be plotted (e.g. corresponding to quantize 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 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.
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 TRUE) if energy values are specified, and FALSE otherwise.
... Additional graphical parameters to be passed to underlying plot function.
axispar A named list of three other lists (ax1, ax2, and ax4) containing parameters passed to the corresponding axis calls for the three axis time (ax1), notes (ax2), and energy (ax4).
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 mtext call.
energypar A list of parameters to be passed to the points function that draws the energy values.
expectedpar A list of parameters to be passed to the rect function that draws the rectangles for expected values.
gridpar A named list of two other lists (gridbar and gridinner) containing parameters passed to the abline functions that draw the grid lines (for bar separators and inner bar (note) separators).
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, melodyplot, quantize; for an example, see the help in tuneR.

readMidi
Read a MIDI file

Description
A MIDI file is read and returned in form of a structured data frame containing most event information (minus some meta events and minus all system events). For details about the represented information see the reference given below.

Usage
readMidi(file)

Arguments
file Filename of MIDI file.

Value
A data frame consisting of columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Time or delta-time of the events, depending on the MIDI format.</td>
</tr>
<tr>
<td>event</td>
<td>A factor indicating the event.</td>
</tr>
<tr>
<td>type</td>
<td>An integer indicating the type of a “meta event”, otherwise NA.</td>
</tr>
<tr>
<td>channel</td>
<td>The channel number or NA if not applicable.</td>
</tr>
<tr>
<td>parameter1</td>
<td>First parameter of an event, e.g. a representation for a note in a “note event”.</td>
</tr>
<tr>
<td>parameter2</td>
<td>Second parameter of an event.</td>
</tr>
<tr>
<td>parameterMetaSystem</td>
<td>Information in a “meta event”, currently all meta events are converted to a character representation (of hex, if all fails), but future versions may have more appropriate representations.</td>
</tr>
<tr>
<td>track</td>
<td>The track number.</td>
</tr>
</tbody>
</table>

Please see the given reference about the MIDI file format about details.

Note
The data structure may be changed or extended in future versions.
Author(s)
Uwe Ligges and Johanna Mielke

References

See Also
The function `getMidiNotes` extracts a more readable representation of note events only.
You may also want to read Wave (`readWave`) or MP3 (`readMP3`).

Examples
```r
content <- readMidi(system.file("example_files", "Bass_sample.mid", package="tuneR"))
str(content)
content
```

`readMP3`  
Read an MPEG-2 layer 3 file into a Wave object

Description
A bare bones MPEG-2 layer 3 (MP3) file reader that returns the results as 16bit PCM data stored in a Wave object.

Usage
`readMP3(filename)`

Arguments
- `filename` Filename of MP3 file.

Value
A `Wave` object.

Note
The decoder can currently only handle files which are either mono or stereo. This is a limitation of the Wave object and the underlying MAD decoder.

Author(s)
Olaf Mersmann <olafm@statistik.tu-dortmund.de>
**readWave**

**References**

The decoder source code is taken from the MAD library, see [http://www.underbit.com/products/mad/](http://www.underbit.com/products/mad/).

**See Also**

Wave

**Examples**

```r
## Not run:
## Requires an mp3 file named sample.mp3 in the current directory.
mpt <- readMP3("sample.mp3")
summary(mpt)

## End(Not run)
```

---

**readWave**

*Reading Wave files*

**Description**

Reading Wave files.

**Usage**

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

**Arguments**

- **filename**: Filename of the file to be read.
- **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.
- **toWaveMC**: If TRUE, a WaveMC-class object is returned. If NULL (default) or FALSE and a non-extensible Wave file or an extensible Wave file with no other than the “FL” and “FR” channels is found, a Wave-class object is returned, otherwise a WaveMC-class object.
Value

An object of class `Wave` or `WaveMC` or a list containing just the header information if `header = TRUE`. If the latter, some experimental support for reading bext chunks in Broadcast Wave Format files is implemented, and the content is returned as an unprocessed string (character).

Author(s)

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

See Also

`Wave-class`, `Wave`, `WaveMC-class`, `WaveMC`, `writeWave`

Examples

```r
Wobj <- sine(440)

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

Description

Showing objects

Methods

- `object = "Wave"` The `Wave` object is being shown. The number of samples, duration in seconds, Samplingrate (Hertz), Stereo / Mono, PCM / IEEE, 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 = "WaveMC"` The `WaveMC` object is being shown. The number of samples, duration in seconds, Samplingrate (Hertz), number of channels, PCM / IEEE, 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.
Meta Function for Smoothers

Description

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

Usage

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

Arguments

- **notes**: Series of notes, a vector of integers such as returned by `noteFromFF`.
- **method**: Currently, only a running 'median' (using `decmedian` in package `pastecs`) is available.
- **order**: The window used for the running median corresponds to 2*order + 1.
- **times**: 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>
**spec2cep**

---

**Spectra to Cepstra Conversion**

**Description**

Calculate cepstra from spectral samples (in columns of spec) through Discrete Cosine Transformation.

**Usage**

```r
spec2cep(spec, ncep = 12, type = c("t2", "t1", "t3", "t4"))
```

**Arguments**

- `spec`: Input spectra (samples/time frames in columns).
- `ncep`: Number of cepstra to return.
- `type`: DCT Type.

**Value**

- `cep`: Matrix of resulting cepstra.
- `dctm`: Returns the DCT matrix that `spec` was multiplied by to give `cep`.

**Author(s)**

Sebastian Krey <krey@statistik.tu-dortmund.de>

**References**


**See Also**

- `lpc2cep`

**Examples**

```r
testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
p spectrum <- powspec(testsound@left, testsound@samp.rate)
aspectrum <- audspec(p spectrum, testsound@samp.rate)
cepstra <- spec2cep(aspectrum$aspectrum)
```
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 shown and an additional summary of the Wave-object’s (one or two) channels is given.

object = "WaveMC" The WaveMC object is being shown and an additional summary of the WaveMC-object’s channels is given.

object = "Wspec" The Wspec object is being shown 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 shown 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, WaveMC-class, WaveMC, Wspec, WspecMat, show

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 (multi channel) Wave files (`Wave`, `WaveMC`), 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> with contributions from Sebastian Krey, Olaf Mersmann, Sarah Schnackenberg, Andrea Preussner, Anita Thieler, and Claus Weihs, as well as code fragments and ideas from the former package `sound` by Matthias Heymann and functions from `rastamat` by Daniel P. W. Ellis. The included parts of the `libmad` MPEG audio decoder library are authored by Underbit Technologies.

Examples

```r
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), sine(220))
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'\textsuperscript{\prime}=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

updateWave

Update old Wave objects for use with new versions of tuneR

Description

Update old Wave objects generated with tuneR < 1.0.0 to the new class definition for use with new versions of the package.

Usage

updateWave(object)

Arguments

object An object of Wave-class.
**Details**

This function is only needed to convert Wave-class objects that have been saved with tuneR versions prior to 1.0-0 to match the new class definition.

**Value**

An object of Wave-class as implemented in tuneR versions >= 1.0-0.

**Author(s)**

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

**See Also**

Wave-class, Wave

**Examples**

```r
x <- sine(440)
updateWave(x)
```

---

**Wave**

Constructors and coercion for class Wave objects

**Description**

Constructors and coercion for class Wave objects

**Usage**

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

**Arguments**

- `left, right, samp.rate, bit, pcm`

  See Section “Slots” on the help page Wave-class. Except for numeric, the argument `left` can also be a matrix (1 or 2 columns), data.frame (1 or 2 columns), list (1 or 2 elements), or WaveMC (1 or 2 channels) object representing the channels.

  ... Further arguments to be passed to the numeric method.

**Details**

The class definition has been extended in tuneR version 1.0-0. Saved objects of class Wave generated with former versions can be updated with updateWave to match the new definition.
Value
An object of **Wave-class**.

Author(s)
Uwe Ligges <ligges@statistik.tu-dortmund.de>

See Also
Wave-class, WaveMC-class, writeWave, readWave, updateWave

Examples

```r
# 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)
```

---

**Wave-class**

**Class Wave**

**Description**
Class “Wave”.

**Details**
The class definition has been extended in tuneR version 1.0-0. Saved objects of class Wave generated with former versions can be updated with updateWave to match the new definition.

**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.
- **pcm**: Object of class "logical" indicating whether this is a PCM or IEEE_FLOAT Wave format.
Waveforms

Author(s)
Uwe Ligges <ligges@statistik.tu-dortmund.de>

See Also
Wave, updateWave, and for multi channel Wave files see WaveMC-class

Waveforms

Create Wave Objects of Special Waveforms

Description
Create a Wave object of special waveform such as silence, power law (white, red, pink, ...) noise, sawtooth, sine, square, and pulse.

Usage
noise(kind = c("white", "pink", "power", "red"), duration = samp.rate,
samp.rate = 44100, bit = 1, stereo = FALSE,
xunit = c("samples", "time"), alpha = 1, ...)
pulse(freq, duration = samp.rate, from = 0, samp.rate = 44100,
bit = 1, stereo = FALSE, xunit = c("samples", "time"),
width = 0.1, plateau = 0.2, interval = 0.5, ...)
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, “white”, “pink”, “power”, or “red” (these are not dB adjusted (!) but all except for “white” are linear decreasing on a log-log scale). Algorithm for generating power law noise is taken from Timmer and König (1995).

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 numeric 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 either to integers in [-2147483647, 2147483647]
     (PCM Wave format if pcm = TRUE) or to numeric values in [-1, 1] (FLOAT_IEEE
     Wave format if pcm = FALSE),
     64 (i.e. 64-bit) for rescaling to numeric values in [-1, 1] (FLOAT_IEEE Wave
     format), and
     0 for not rescaling at all. These numbers are internally passed to normalize.
     The Wave slot bit will be set to 32 if bit = 0, bit = 1 or bit = 32.
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.
alpha  The power for the power law noise (defaults are 1 for pink and 1.5 for red noise)
     $1/f^\alpha$.
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).
width  Relative pulses width: the proportion of time the amplitude is non-zero.
plateau  Relative plateau width: the proportion of the pulse width where amplitude is ±1.
interval  Relative interval between the up-going and down-going pulses with respect to
     the center of the wave period (0: immediately after up-going, 1: center of the
     wave period).
...  Further arguments to be passed to Wave through the internal function postWaveform.

Value

A Wave object.

Author(s)

Uwe Ligges <ligges@statistik.tu-dortmund.de>, partly based on code from Matthias Hey-
mann’s former package ‘sound’, Anita Thieler, Guillaume Guénard

References

WaveMC

See Also

Wave-class, Wave, normalize, noSilence

Examples

Wobj <- sine(440, duration = 1000)
Wobj2 <- noise(duration = 1000)
Wobj3 <- pulse(220, duration = 1000)
plot(Wobj)
plot(Wobj2)
plot(Wobj3)

WaveMC

Constructors and coercion for class WaveMC objects

Description

Constructors and coercion for class WaveMC objects

Usage

WaveMC(data, ...)
## S4 method for signature 'matrix'
WaveMC(data = matrix(numeric(0), 0, 0), samp.rate = 44100, bit = 16, pcm = TRUE, ...)

Arguments

data

Except for a numeric matrix, the argument data can also be a numeric vector (for one channel), data.frame (columns representing channels), list (elements containing numeric vectors that represent the channels), or Wave object.

samp.rate, bit, pcm

See Section “Slots” on the help page WaveMC-class.

...

Further arguments to be passed to the matrix method.

Value

An object of WaveMC-class.

Author(s)

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

See Also

WaveMC-class, Wave-class, writeWave, readWave
Examples

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

Description

Class “WaveMC”.

Details

This class has been added in **tuneR** version 1.0-0 for representation and construction of multi channel Wave files. Objects of class `Wave` can be transformed to the new class definition by calls of the form `as(..., "WaveMC")`. Coercion from the `WaveMC` class to the `Wave-class` works via `as(..., "Wave")` if there are no more than 2 channels. Coercing back to the `Wave-class` can be useful since some (very few) functions cannot yet deal with multi channel Wave objects.

Note that also the `Wave-class` definition has been extended in **tuneR** version 1.0-0. For more details see `Wave-class`.

Objects from the Class

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

Slots

- `.Data`: Object of class "matrix" containing numeric data, where each column is representing one channel. Column names are the appropriate way to name different channels. The data object `MCnames` contains a data frame of standard names for channels in multi channel Wave files.
- `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.
- `pcm`: Object of class "logical" indicating whether this is a PCM or IEEE_FLOAT Wave format.

Author(s)

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

See Also

`WaveMC`, `Wave-class`, `MCnames`
**WavPlayer**

*Getting and setting the default player for Wave files*

**Description**

Getting and setting the default player for Wave files

**Usage**

```r
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`

---

**writeWave**

*Writing Wave files*

**Description**

Writing Wave files.

**Usage**

```r
writeWave(object, filename, extensible = TRUE)
```

**Arguments**

- `object`: Object of class `Wave` or `WaveMC` to be written to a Wave file.
- `filename`: Filename of the file to be written.
- `extensible`: If TRUE (default), an extensible Wave format file is written. If FALSE, a non-extensible Wave file is written.
Details

It is only possible to write a non-extensible Wave format file for objects of class `Wave` or for objects of class `WaveMC` with one or two channels (mono or stereo).

If the argument object is a `Wave-class` object, the channels are automatically chosen to be “FL” (for mono) or “FL” and “FR” (for stereo).

The channel mask used to arrange the channel ordering in multi channel Wave files is written according to Microsoft standards as given in the data frame `MCnames` containing the first 18 standard channels. In the case of writing a multi channel Wave file, the column names of the object `object` (`colnames(object)`) must be specified and must uniquely identify the channel ordering for `WaveMC` objects. The column names of the object of class `WaveMC` have to be a subset of the 18 standard channels and have to match the corresponding abbreviated names. (See `MCnames` for possible channels and the abbreviated names: “FL”, “FR”, “FC”, “LF”, “BL”, “BR”, “FLC”, “FRC”, “BC”, “SL”, “SR”, “TC”, “TFL”, “TFC”, “TFR”, “TBL”, “TBC” and “TBR”).

The function `normalize` can be used to transform and rescale data to an appropriate amplitude range for various Wave file formats (either pcm with 8-, 16-, 24- or 32-bit or IEEE_FLOAT with 32- or 64-bit).

Value

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

Author(s)

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

See Also

`Wave-class, Wave, WaveMC-class, WaveMC, normalize, MCnames, readWave`

Examples

```r
Wobj <- sine(440)

tdir <- tempdir()
tfile <- file.path(tdir, "myWave.wav")
writeWave(Wobj, filename = tfile)
list.files(tdir, pattern = "\.wav$")
newWobj <- readWave(tfile)
newWobj
file.remove(tfile)
```
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 or WaveMC 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`, `Wave`, `WaveMC-class`, and `WaveMC`)
- `WspecMat` for a similar class that represents the spectrum in form of a matrix.

---

### WspecMat-class

**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` or `WaveMC` 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".
Methods

Description

Operators act on objects to extract or replace subsets.

See Also

- Extract for the S3 generic.
Index

* CD quality
downsample, 8
* IO
  play-methods, 32
  readMidi, 42
  readMP3, 43
  readWave, 44
  writeWave, 57
* LilyPond
  lilyinput, 16
* MIDI
  readMidi, 42
* MP3
  readMP3, 43
* WaveMC
  downsample, 8
  noSilence, 25
  panorama, 28
  prepComb, 38
  WaveMC, 55
  WaveMC-class, 56
* Wave
  bind, 5
  channel, 5
  downsample, 8
  equalWave, 9
  extractWave, 9
  FF, 11
  Mono-Stereo, 23
  normalize-methods, 24
  noSilence, 25
  noteFromFF, 26
  panorama, 28
  periodogram-methods, 29
  play-methods, 32
  plot-Wave, 33
  plot-Wspec, 34
  plot-WspecMat, 35
  prepComb, 38
  readWave, 44
  show-WaveWspec-methods, 45
  tuneR, 48
  Wave, 51
  Wave-class, 52
  Waveforms, 53
  WaveMC, 55
  WavPlayer, 57
  writeWave, 57
  Wspec-class, 59
  WspecMat-class, 60
* aplot
  plot-Wave, 33
* arith
  Arith-methods, 3
* bark
  audspec, 3
  freqconv, 12
* bar
  quantize, 39
* bin
  quantize, 39
* bit
  Wave, 51
  Wave-class, 52
  WaveMC, 55
  WaveMC-class, 56
* cepstra
  lpc2cep, 17
  spec2cep, 47
* cepstrum
  melfcc, 19
* channel
  channel, 5
  Mono-Stereo, 23
  panorama, 28
  Wave, 51
  Wave-class, 52
  WaveMC, 55
INDEX

WaveMC-class, 56
* classes
  Wave-class, 52
  WaveMC-class, 56
  Wspec-class, 59
  WspecMat-class, 60
* compression
  postaud, 36
* conversion
  audspec, 3
  freqconv, 12
  lpc2cep, 17
  spec2cep, 47
* cut
  noSilence, 25
* datagen
  Waveforms, 53
* datasets
  MCnames, 18
* declick
  prepComb, 38
* deltas
  deltas, 6
* documentation
  tuneR, 48
* durbin
  dolpc, 7
* error
  equalWave, 9
* f0
  FF, 11
  noteFromFF, 26
* file
  lilyinput, 16
  readMidi, 42
  readMP3, 43
  readWave, 44
  writeWave, 57
* frequency
  audspec, 3
  FF, 11
  freqconv, 12
  noteFromFF, 26
* fundamental
  FF, 11
  noteFromFF, 26
* hertz
  freqconv, 12
* hplot
  melodyplot, 21
  plot-Wave, 33
  plot-Wspec, 34
  plot-WspecMat, 35
  quantplot, 40
* interface
  lilyinput, 16
  play-methods, 32
* iplot
  extractWave, 9
* levinson
  dolpc, 7
* liftering
  lifter, 15
* loudness
  postaud, 36
* lpc
  dolpc, 7
  lpc2cep, 17
  melfcc, 19
* manip
  bind, 5
  channel, 5
  downsample, 8
  extractWave, 9
  Mono-Stereo, 23
  normalize-methods, 24
  noSilence, 25
  panorama, 28
  prepComb, 38
* median
  smoother, 46
* melody
  melodyplot, 21
  quantplot, 40
* mel
  audspec, 3
  freqconv, 12
  melfcc, 19
* methods
  [-methods, 61
   Arith-methods, 3
   length, 14
   play-methods, 32
   plot-Wave, 33
   plot-Wspec, 34
   plot-WspecMat, 35
show-WaveWspec-methods, 45
summary-methods, 48
Wave, 51
WaveMC, 55
* mfcc
  melfcc, 19
* misc
  smoother, 46
* mono
  Mono-Stereo, 23
  Wave, 51
  Wave-class, 52
  WaveMC, 55
  WaveMC-class, 56
* music
  play-methods, 32
  plot-Wave, 33
  readMP3, 43
  readWave, 44
  tuneR, 48
  Wave, 51
  Wave-class, 52
  WaveMC, 55
  WaveMC-class, 56
  WavPlayer, 57
  writeWave, 57
  Wspec-class, 59
  WspecMat-class, 60
* noise
  noSilence, 25
* note
  lilyinput, 16
  melodyplot, 21
  noteFromFF, 26
  notenames, 27
  quantize, 39
  quantplot, 40
* periodogram
  FF, 11
  noteFromFF, 26
  plot-Wspec, 34
  plot-WspecMat, 35
  show-WaveWspec-methods, 45
  tuneR, 48
  Wspec-class, 59
  WspecMat-class, 60
* pitch
  FF, 11
  melodyplot, 21
  noteFromFF, 26
  quantplot, 40
* player
  play-methods, 32
  WavPlayer, 57
* plp
  dolpc, 7
  melfcc, 19
  postaud, 36
* powerspectrum
  powspec, 37
* print
  show-WaveWspec-methods, 45
  summary-methods, 48
* quantization
  quantize, 39
* recursion
  dolpc, 7
* running
  smoother, 46
* sample
  Waveforms, 53
* sampling rate
  downsample, 8
  Wave, 51
  Wave-class, 52
  WaveMC, 55
  WaveMC-class, 56
* sampling
  downsample, 8
  Wave, 51
  Wave-class, 52
  WaveMC, 55
  WaveMC-class, 56
* silence
  Waveforms, 53
* silence
  noSilence, 25
* smooth
  smoother, 46
* sound
  play-methods, 32
  readMP3, 43
  readWave, 44
  Waveforms, 53
  WavPlayer, 57
  writeWave, 57
INDEX

* spectogram
  plot-WspecMat, 35

* spectra
  spec2cep, 47

* spectrum
  periodogram-methods, 29
  Wspec-class, 59
  WspecMat-class, 60

* speech
  play-methods, 32
  plot-Wave, 33
  readMP3, 43
  readWave, 44
  Wave, 51
  Wave-class, 52
  WaveMC, 55
  WaveMC-class, 56
  WavPlayer, 57
  writeWave, 57
  Wspec-class, 59
  WspecMat-class, 60

* stereo
  Mono-Stereo, 23
  panorama, 28
  Wave, 51
  Wave-class, 52
  WaveMC, 55
  WaveMC-class, 56

* tracking
  FF, 11
  melodyplot, 21
  noteFromFF, 26
  quantplot, 40

* transcribe
  lilyinput, 16

* transcription
  lilyinput, 16
  melodyplot, 21
  quantplot, 40

* ts
  FF, 11
  melfcc, 19
  periodogram-methods, 29
  smoother, 46

* utilities
  bind, 5
  channel, 5
  downsample, 8
  equalWave, 9
  extractWave, 9
  Mono-Stereo, 23
  noSilence, 25
  noteFromFF, 26
  notenames, 27
  play-methods, 32
  prepComb, 38
  quantize, 39
  WavPlayer, 57

* waveform
  Waveforms, 53
  [,ANY-method ([methods], 61
  [,Wave-method (Wave), 51
  [,WaveMC-method (WaveMC), 55
  [,Wspec-method (Wspec-class), 59
  [,WspecMat-method (WspecMat-class), 60
  [-methods, 61
  abline, 22, 41
  Arith, numeric, Wave-method
    (Arith-methods), 3
  Arith, numeric, WaveMC-method
    (Arith-methods), 3
  Arith, Wave, missing-method
    (Arith-methods), 3
  Arith, Wave, numeric-method
    (Arith-methods), 3
  Arith, Wave, Wave-method (Arith-methods), 3
  Arith, WaveMC, numeric-method
    (Arith-methods), 3
  Arith, WaveMC, WaveMC-method
    (Arith-methods), 3
  Arith-methods, 3
  audspec, 3, 36, 37
  axis, 22, 41
  bark2hz (freqconv), 12
  bind, 5, 10, 39, 49
  bind, Wave-method (bind), 5
  bind, WaveMC-method (bind), 5
  channel, 5, 10, 49
  coerce, data.frame, Wave-method (Wave), 51
  coerce, data.frame, WaveMC-method
    (WaveMC), 55
  coerce, list, Wave-method (Wave), 51
  coerce, list, WaveMC-method (WaveMC), 55
coerce, matrix, Wave-method (Wave), 51
coerce, matrix, WaveMC-method (WaveMC), 55
correct, numeric, Wave-method (Wave), 51
correct, numeric, WaveMC-method (WaveMC), 55
correct, Wave, data.frame-method (Wave), 51
correct, Wave, WaveMC-method (WaveMC), 51
correct, WaveGeneral, list-method (Wave), 51
correct, WaveMC, data.frame-method (WaveMC), 55
correct, WaveMC, matrix-method (WaveMC), 55
correct, WaveMC, Wave-method (WaveMC), 55
correct, Wspec, WspecMat-method (WspecMat-class), 60
decmedian, 46
deltas, 6
dolpc, 7, 37
downsample, 8, 30, 49
equalWave, 5, 9
Extract, 61
extractWave, 5, 6, 9, 26, 39, 49
FF, 11, 22, 27, 42, 49
FFpure, 49
FFpure (FF), 11
fft2barkmx, 4
fft2melmx, 4
freqconv, 12
getMidiNotes, 13, 43
getWavPlayer (WavPlayer), 57
groupGeneric, 3
hz2bark (freqconv), 12
hz2mel (freqconv), 12
image, 35, 36
image, ANY-method (plot-WspecMat), 35
image, Wspec-method (plot-WspecMat), 35
image-Wspec (plot-WspecMat), 35
interactive, 10
length, 14, 14
length, ANY-method (length), 14
length, Wave-method (length), 14
length, WaveMC-method (length), 14
levinson, 7
lifter, 15
lilyinput, 16, 40, 49
lines, 22, 41
lpc2cep, 17, 47
MCnames, 18, 56, 58
mel2hz (freqconv), 12
melfcc, 19
melodyplot, 21, 42, 49
mono, 6, 10, 30, 49
mono (Mono-Stereo), 23
Mono-Stereo, 23
mtext, 22, 41
nchannel, 24
nchannel, Wave-method (nchannel), 24
nchannel, WaveMC-method (nchannel), 24
noise (Waveforms), 53
normalize, 54, 55, 58
normalize (normalize-methods), 24
normalize, Wave-method (normalize-methods), 24
normalize, WaveMC-method (normalize-methods), 24
normalize-methods, 24
noSilence, 25, 39, 55
noSilence, Wave-method (noSilence), 25
noSilence, WaveMC-method (noSilence), 25
noteFromFF, 12, 21, 22, 26, 39, 40, 42, 46, 49
notenames, 13, 27, 40
options, 57
panorama, 28
panorama, Wave-method (panorama), 28
panorama, WaveMC-method (panorama), 28
par, 22, 33, 34, 41
periodogram, 12, 27, 35, 36, 46, 49, 59, 60
periodogram (periodogram-methods), 29
periodogram, character-method (periodogram-methods), 29
periodogram, WaveGeneral-method (periodogram-methods), 29
periodogram-methods, 29
play, 49, 57
play (play-methods), 32
play, character-method (play-methods), 32
play, WaveGeneral-method (play-methods), 32
INDEX

plot, Wave, missing-method (plot-Wave), 33
plot, WaveMC, missing-method (plot-Wave), 33
plot, Wspec, missing-method (plot-Wspec), 34
plot, WspecMat, missing-method (plot-WspecMat), 35
plot-Wave, 33
plot-Wspec, 34
plot-WspecMat, 35
plot_Wave_channel (plot-Wave), 33
points, 41
postaud, 36
powspec, 4, 37
prepComb, 5, 38
pulse (Waveforms), 53
quantize, 17, 39, 41, 42, 49
quantMerge, 17, 49
quantMerge (quantize), 39
quantplot, 17, 22, 40, 40, 49
readMidi, 13, 14, 42
readMP3, 43, 43
readWave, 43, 44, 49, 52, 55, 58
rect, 22, 41
round, 27
sawtooth (Waveforms), 53
setWavPlayer, 32
setWavPlayer (WavPlayer), 57
show, 48
show, Wave-method
(show-WaveWspec-methods), 45
show, WaveMC-method
(show-WaveWspec-methods), 45
show, Wspec-method
(show-WaveWspec-methods), 45
show, WspecMat-method
(show-WaveWspec-methods), 45
show-WaveWspec-methods, 45
silence, 26
silence (Waveforms), 53
sine, 49
sine (Waveforms), 53
smoother, 46, 49
spec.pgram, 29–31, 60
spec2cep, 18, 47
specgram, 37
spectrum, 30, 48, 59, 60
square (Waveforms), 53
stereo, 5, 49
stereo (Mono-Stereo), 23
stop, 9
summary, ANY-method (summary-methods), 48
summary, Wave-method (summary-methods), 48
summary, WaveMC-method (summary-methods), 48
summary, Wspec-method (summary-methods), 48
summary, WspecMat-method (summary-methods), 48
summary-methods, 48
summary.default, 48
tuneR, 12, 17, 22, 27, 34–36, 40, 42, 48
tuneR-package (tuneR), 48
updateWave, 50, 51–53
Wave, 3, 5, 6, 8–10, 14, 23–26, 28–34, 38, 39,
43–46, 48, 49, 51, 51, 52–55, 57–60
Wave, ANY-method (Wave), 51
Wave, data.frame-method (Wave), 51
Wave, list-method (Wave), 51
Wave, matrix-method (Wave), 51
Wave, numeric-method (Wave), 51
Wave, WaveMC-method (Wave), 51
Wave-class, 3, 5, 6, 8–10, 19, 23–26, 29, 31,
32, 34, 39, 44–46, 48, 50–52, 52,
55–58, 60
Waveforms, 53
WaveMC, 3, 5, 8–10, 14, 24–26, 28–34, 38, 39,
45, 46, 48, 49, 55, 56–60
WaveMC, ANY-method (WaveMC), 55
WaveMC, data.frame-method (WaveMC), 55
WaveMC, list-method (WaveMC), 55
WaveMC, matrix-method (WaveMC), 55
WaveMC, numeric-method (WaveMC), 55
WaveMC, Wave-method (WaveMC), 55
WaveMC-class, 3, 5, 6, 8–10, 19, 23–26, 29,
31, 32, 34, 39, 44–46, 48, 52, 53, 55,
56, 58, 60
WavPlayer, 57
writeWave, 25, 32, 45, 49, 52, 55, 57