

# Package ‘tuneR’

February 15, 2012

**Version** 0.4-0

**Date** 2010-06-29

**Title** Analysis of music and speech

**Author** Uwe Ligges <ligges@statistik.tu-dortmund.de> with contributions from Sebastian Krey, Olaf Mersmann, Andrea Preusser, 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.

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

**Depends** R (>= 2.10.0), methods

**Suggests** pastecs

**Imports** signal

**Description** Collection of tools to analyze music, extract features like MFCCs, handling wave files, read mp3, transcription, ... Also contains functions ported from the rastamat Matlab package.

**License** GPL-2

**Repository** CRAN

**Date/Publication** 2011-06-29 12:04:27

## R topics documented:

Arith-methods . . . . .	3
audspec . . . . .	3
bind . . . . .	4
channel . . . . .	5
deltas . . . . .	6
dolpc . . . . .	7
downsample . . . . .	8
equalWave . . . . .	8

extractWave . . . . .	9
FF . . . . .	10
freqconv . . . . .	12
length . . . . .	13
lifter . . . . .	13
lilyinput . . . . .	14
lpc2cep . . . . .	16
melfcc . . . . .	17
melodyplot . . . . .	19
MFCC . . . . .	21
Mono-Stereo . . . . .	21
normalize . . . . .	22
noSilence . . . . .	23
noteFromFF . . . . .	24
notenames . . . . .	25
panorama . . . . .	26
periodogram-methods . . . . .	27
play-methods . . . . .	29
plot . . . . .	30
plot-Wave . . . . .	31
plot-Wspec . . . . .	32
plot-WspecMat . . . . .	33
postaud . . . . .	34
powspec . . . . .	35
prepComb . . . . .	36
quantize . . . . .	37
quantplot . . . . .	38
readMP3 . . . . .	40
show-WaveWspec-methods . . . . .	41
smoother . . . . .	42
spec2cep . . . . .	42
summary-methods . . . . .	43
tuneR . . . . .	44
Wave . . . . .	45
Wave-class . . . . .	46
Waveforms . . . . .	47
WaveIO . . . . .	49
WavPlayer . . . . .	50
Wspec-class . . . . .	51
WspecMat-class . . . . .	52
[-methods . . . . .	53

---

Arith-methods                      *Arithmetics on Waves*

---

**Description**

Methods for arithmetics on Wave objects

**Methods**

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

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

**object = "missing"** For unary Wave operations.

**Author(s)**

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

**See Also**

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

---

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(pspectrum, sr = 16000, nfilts = ceiling(hz2bark(sr/2)) + 1,
        fbtype = c("bark", "mel", "htkmel", "fcmel"), minfreq = 0,
        maxfreq = sr/2, sumpower = TRUE, bwidth = 1)
```

**Arguments**

pspectrum	Output of <a href="#">powspec</a> , 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 `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.

bwidth Modify the width of the frequency bands.

### Value

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

wtw Weight matrix for the frequency band conversion.

### Author(s)

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

### References

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

### See Also

[fft2melmx](#), [fft2barkmx](#)

### Examples

```
testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
pspectrum <- powspec(testsound@left, testsound@samp.rate)
aspectrum <- audspec(pspectrum, testsound@samp.rate)
```

---

bind

*Concatenating Wave objects*

---

### Description

Concatenating objects of class `Wave`.

### Usage

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

### Arguments

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

### Value

An object of class `Wave`.

**Author(s)**

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

**See Also**

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

---

channel

*Channel conversion for Wave objects*

---

**Description**

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

**Usage**

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

**Arguments**

object	Object of class <a href="#">Wave</a> .
which	Character indicating which channel(s) should be returned.

**Value**

Wave object including channels specified by which.

**Author(s)**

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

**See Also**

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

---

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

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

### Examples

```
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

```
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

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

### See Also

[levinson](#)

### Examples

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

downsample                      *Downsampling a Wave object*

---

**Description**

Downsampling an object of class Wave.

**Usage**

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

**Arguments**

object	Object of class <a href="#">Wave</a> .
samp.rate	Sampling rate the object is to be downsampled to. <code>samp.rate</code> 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 <code>samp.rate</code> , the object will be returned unchanged.

**Value**

An object of class [Wave](#).

**Author(s)**

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

**See Also**

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

---

equalWave                      *Checking Wave objects*

---

**Description**

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

**Usage**

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

**Arguments**

object1, object2	Object(s) of class <a href="#">Wave</a> .
------------------	---

**Value**

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

**Author(s)**

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

**See Also**

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

---

extractWave

*Extractor for Wave objects*

---

**Description**

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

**Usage**

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

**Arguments**

object	Object of class <a href="#">Wave</a> .
from	Sample number or time in seconds (see xunit) at which to <i>start</i> extraction.
to	Sample number or time in seconds (see xunit) at which to <i>stop</i> 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 ( <a href="#">plot-methods</a> ), if interact = TRUE.

## Details

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

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

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

## Value

An object of class `Wave`.

## Author(s)

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

## See Also

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

## Examples

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

## Not run:
# or interactively:
Wobj2 <- extractWave(Wobj)

## End(Not run)
```

## Description

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

**Usage**

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

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

**Arguments**

object	An object of class <a href="#">Wspec</a> .
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: <ol style="list-style-type: none"> <li>1. The first peak larger than peakheight * 'largest_peak' is taken.</li> <li>2. Its frequency is multiplied by 1+search.par[1] Now, any larger peak between the old peak and that value is taken, if (a) it exists and if (b) it is above the search.par[2]-th Fourier-Frequency.</li> <li>3. Within the interval of frequencies 'current peak' * search.par[3:4], another high peak is looked for. If any high peak exists in that interval, it can be assumed we got the wrong partial and the 'real' fundamental frequency can be re-estimated from the next two partials.</li> </ol>

**Details**

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

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

### Value

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

### Note

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

### Author(s)

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

### See Also

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

---

freqconv

*Frequency scale conversion*

---

### Description

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

### Usage

```
bark2hz(z)
hz2bark(f)
hz2mel(f, htk = FALSE)
mel2hz(z, htk = FALSE)
```

### 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**

Daniel P. W. Ellis: <http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/>, Malcolm Slaney: Auditory Toolbox

**Examples**

```
hz2bark(440)
bark2hz(hz2bark(440))
hz2mel(440, htk = TRUE)
mel2hz(hz2mel(440, htk = TRUE), htk = TRUE)
hz2mel(440, htk = FALSE)
mel2hz(hz2mel(440, htk = FALSE), htk = FALSE)
```

---

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

---

**Description**

S4 generic for length.

**Methods**

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

**See Also**

For the primitive: [length](#)

---

lifter	<i>Liftering of cepstra</i>
--------	-----------------------------

---

**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 *xi<sup>l</sup>ift*,  $i = 1, \dots, nrow(x)$  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**

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

**Examples**

```
testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
m <- melfcc(testsound, frames_in_rows=FALSE)
unlm <- lifter(m, inv=TRUE)
```

---

lilyinput

*Providing LilyPond compatible input*


---

**Description**

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

**Usage**

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

**Arguments**

- X                    A data frame containing 4 named components (columns):
- note: Integer - the notes' pitch in halftones from diapason (a), i.e. 0 for diapason a, 3 for c', ...
  - length: Integer - denominator of lengths of the notes, e.g. 8 for a quaver.
  - punctate: Logical - whether to punctate a note.

- `slur`: Logical - TRUE indicates to start a slur, or to end it. That means that the first, third, ... occurrences of TRUE start slurs, while the second, fourth, ... occurrences end slurs. Note that it is only possible to draw one slur at a time.

<code>file</code>	The file to be written for <i>LilyPond</i> 's input.
<code>Major</code>	Logical indicating major key (if TRUE) or minor key.
<code>key</code>	Keynote, necessary to set sharps/flats.
<code>clef</code>	Integer indicating the kind of clef, supported are 'treble' (default), 'bass', 'alto', and 'tenor'.
<code>time</code>	Character indicating which meter to use, examples are: "3/4", "4/4".
<code>endbar</code>	Logical indicating whether to set an ending bar at the end of the sheet music.
<code>midi</code>	Logical indicating whether Midi output (by <i>LilyPond</i> ) is desirable.
<code>tempo</code>	Character specifying the tempo to be used for the Midi file if <code>midi = TRUE</code> . The default, "2 = 60" indicates: 60 half notes per minute, whereas "4 = 90" indicates 90 quarters per minute.
<code>textheight</code>	Textheight of the sheet music to be written by <i>LilyPond</i> .
<code>linewidth</code>	Linewidth of the sheet music to be written by <i>LilyPond</i> .
<code>indent</code>	Indentation of the sheet music to be written by <i>LilyPond</i> .
<code>fontsize</code>	Fontsize of the sheet music to be written by <i>LilyPond</i> .

### Details

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

### Value

Nothing is returned, but a file is written.

### Note

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

### Author(s)

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

### References

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

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

**See Also**

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

---

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**

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

**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)
```

---

melfcc *MFCC Calculation*


---

**Description**

Calculate Mel-frequency cepstral coefficients.

**Usage**

```
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 class 'Wave'.
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 -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 > 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
pspectrum	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', 'NUMCHANS = 20', 'CEPLIFTER = 22', 'NUMCEPS = 12', 'WINDOWSIZE = 250000.0', 'USEHAMMING = T', 'TARGETKIND = MFCC\_0').

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

**Author(s)**

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

**References**

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

**Examples**

```

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

#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**

```

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 <a href="#">Wspec</a> .
observed	Observed notes, probably as a result from <a href="#">noteFromFF</a> (or a smoothed version). This should correspond to the <a href="#">Wspec</a> 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.

<code>xlab, ylab</code>	Annotation of x/y-axes.
<code>xlim, ylim</code>	Range of x/y-axis, where <code>ylim</code> must be an integer that represents the range of note heights that should be displayed.
<code>observedtype</code>	Type (either "p" for points or "l" for lines) used for representing observed notes. "l" (the default) is not sensible for polyphonic representations.
<code>observedcol</code>	Colour for the observed melody.
<code>expectedcol</code>	Colour for the expected melody.
<code>gridcol</code>	Colour of the grid.
<code>lwd</code>	Line width, see <a href="#">par</a> for details.
<code>las</code>	Orientation of axis labels, see <a href="#">par</a> for details.
<code>cex.axis</code>	Size of tick mark labels, see <a href="#">par</a> for details.
<code>mar</code>	Margins of the plot, see <a href="#">par</a> for details.
<code>notenames</code>	Optionally specify other notenames (character) for the y axis.
<code>thin</code>	Amount of thinning of notenames, i.e. only each thinth notename is displayed on the y-axis.
<code>silence</code>	Character string for label of the 'silence' (default) axis.
<code>plotenergy</code>	Logical (default: TRUE), whether to plot energy values in the bottom part of the plot.
<code>...</code>	Additional graphical parameters to be passed to underlying plot function.
<code>axispar</code>	A named list of three other lists ( <code>ax1</code> , <code>ax2</code> , and <code>ax4</code> ) containing parameters passed to the corresponding <a href="#">axis</a> calls for the three axis time ( <code>ax1</code> ), notes ( <code>ax2</code> ), and energy ( <code>ax4</code> ).
<code>boxpar</code>	A list of parameters to be passed to the box generating functions.
<code>energylabel</code>	A list of parameters to be passed to the energy-label generating <a href="#">mtext</a> call.
<code>energypar</code>	A list of parameters to be passed to the <a href="#">lines</a> function that draws the energy curve.
<code>expectedpar</code>	A list of parameters to be passed to the <a href="#">rect</a> function that draws the rectangles for expected values.
<code>gridpar</code>	A list of parameters to be passed to the <a href="#">abline</a> function that draws the grid lines.
<code>observedpar</code>	A list of parameters to be passed to the <a href="#">lines</a> function that draws the observed values.

**Author(s)**

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

**See Also**

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

MFCC

*DEPRECATED: Mel Frequency Cepstral Coefficients***Description**

DEPRECATED: Computation of MFCCs — this has been replaced by [melfcc](#) already and is just a wrapper! Will be removed shortly.

**Usage**

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

**Arguments**

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

**Note**

This function was always documented to be in development and highly EXPERIMENTAL!!!

**See Also**

[melfcc](#)

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

**Value**

An object of class [Wave](#).

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

**Author(s)**

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

**See Also**

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

**Examples**

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

---

normalize

*Rescale the range of values*

---

**Description**

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

**Usage**

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

**Arguments**

object	Object of class <a href="#">Wave</a> .
unit	Unit to rescale to. "1" (default) for rescaling to real values in [-1,1], "8" (i.e. 8-bit) for rescaling to integers in [0, 254], "16" (i.e. 16-bit) for rescaling to integers in [-32767, 32767], 24 (i.e. 24-bit) for rescaling to integers in [-8388607, 8388607], 32 (i.e. 32-bit) for rescaling to integers in [-2147483647, 2147483647], and "0" for not rescaling (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).

**Value**

An object of class [Wave](#).

**Author(s)**

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

**See Also**

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

---

noSilence

*Cut off silence from a Wave object*

---

**Description**

Cut off silence or low noise at the beginning and/or at the end of an object of class [Wave](#).

**Usage**

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

**Arguments**

object	Object of class <a href="#">Wave</a> .
zero	The zero level (default: 0) at which ideal cut points are determined (see Details). A typical alternative would be 127 for 8 bit <a href="#">Wave</a> objects. If zero=NA, the mean of the left <a href="#">Wave</a> channel 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 <a href="#">Wave</a> object for concatenation.

**Details**

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

**Value**

An object of class [Wave](#).

**Author(s)**

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

**See Also**

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

---

noteFromFF

*Deriving notes from frequencies*

---

**Description**

Deriving notes from given (fundamental) frequencies.

**Usage**

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

**Arguments**

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 <a href="#">round</a> . 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  $\text{round}(12 * \log(x / \text{diapason}, 2) + \text{roundshift})$ .

**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 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 interger values vector, where 0 corresponds to a', notes below and above have to be specified in the corresponding half-tone 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**

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

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

---

panorama

*Narrow the Panorama of a Stereo Sample*

---

**Description**

Narrow the panorama of a stereo Wave object.

**Usage**

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

**Arguments**

object	Object of class <a href="#">Wave</a> .
pan	Value in [-1,1] to narrow the panorama, see the Details below. The default (1) does not change anything.

**Details**

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

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

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

**Value**

Wave object with the transformed panorama.

**Author(s)**

Uwe Ligges, based on code by Matthias Heymann

**See Also**[Wave-class](#), [Wave](#)


---

periodogram-methods     *Periodogram (Spectral Density) Estimation on Wave objects*


---

**Description**

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

**Usage**

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

**Arguments**

<code>object</code>	An object of class <a href="#">Wave</a> or a character string pointing to a <a href="#">Wave</a> file.
<code>width</code>	A window of width 'width' running through the time series selects the samples from which the periodograms are to be calculated.
<code>overlap</code>	The window can be applied by each overlapping <code>overlap</code> samples.
<code>starts</code>	Start number (in samples) for a window. If not given, this value is derived from argument <code>ends</code> , or will be derived width and <code>overlap</code> .
<code>ends</code>	End number (in samples) for a window. If not given, this value is derived from argument <code>starts</code> , or will be derived from width and <code>overlap</code> .
<code>taper</code>	proportion of data to taper. See <a href="#">spec.pgram</a> for details.
<code>normalize</code>	Logical; if TRUE (default), two steps will be applied: (i) the input signal will be normalized to amplitude $\max(\text{abs}(\text{amplitude})) = 1$ , (ii) the resulting spec values will be normalized to sum up to one for each periodogram.
<code>frqRange</code>	Numeric vector of two elements indicating minimum and maximum of the frequency range that is to be stored in the resulting object. This is useful to reduce memory consumption.
<code>from</code>	Where to start reading in the <a href="#">Wave</a> file, in units.
<code>to</code>	Where to stop reading in the <a href="#">Wave</a> 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 <a href="#">downsample</a> .
channel	Character, indicating whether the “left” or “right” channel should be extracted (see <a href="#">mono</a> 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 <a href="#">spec.pgram</a> .

### Value

An object of class [Wspec](#) is returned containing the following slots.

freq	Vector of frequencies at which the spectral density is estimated. See <a href="#">spectrum</a> for details. (1)
spec	List of vectors or matrices of the spec values returned by <a href="#">spec.pgram</a> 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 <a href="#">Wave</a> object.
kernel	The kernel argument, or the kernel constructed from spans returned by <a href="#">spec.pgram</a> . (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	Whether the underlying <a href="#">Wave</a> object was stereo or not. (1)
samp.rate	Sampling rate of the underlying <a href="#">Wave</a> object. (1)
variance	The variance of samples in each window, corresponding to amplitude / loudness of sound.
energy	The “energy” $E$ , also an indicator for the amplitude / loudness of sound:

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

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

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

**Note**

Support for processing of stereo [Wave](#) objects has not yet been implemented.

**Author(s)**

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

**See Also**

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

**Examples**

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

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

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

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

---

play-methods

*Playing Waves*

---

**Description**

Plays wave files and objects of class `Wave`.

**Usage**

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

**Arguments**

object	Either a filename pointing to a Wave file, or an object of class <a href="#">Wave</a> . If the latter, it is written to a temporary file by <a href="#">writeWave</a> , 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 exist 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](#), [Wave](#), [writeWave](#), [setWavPlayer](#)

---

plot

*S4 generic for plot*

---

**Description**

S4 generic for plot.

**Methods**

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

**See Also**

For the S3 generic: [plot.default](#)

---

plot-Wave

*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 = 1500, ...)

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

**Arguments**

x	Object of class <a href="#">Wave</a> .
info	Logical, whether to include (written) information on the <a href="#">Wave</a> object within the plot.
xunit	Character indicating which units are used for setting up user coordinates (see <a href="#">par</a> ) and x-axis labeling. If <code>xunit = "time"</code> , the unit is time in seconds, otherwise the number of samples.
ylim	The y (amplitude) limits of the plot.
main, sub	A main / sub title for the plot.
xlab, ylab	Label for x-/y-axis.
simplify	Logical, whether the plot should be “simplified”. If TRUE (default), not all (thousand/millions/billions) of points (samples) of the <a href="#">Wave</a> object are drawn, but the <code>nr</code> (see below) ranges (in form of segments) within <code>nr</code> windows of the time series. Plotting with <code>simplify = FALSE</code> may take several minutes (depending on the number of samples in the Wave) and output in any vector format may be really huge.
nr	Number of windows (segments) to be used <i>approximately</i> (an appropriate number close to <code>nr</code> is selected) to <code>simplify</code> (see above) the plot. Only used if <code>simplify = TRUE</code> and the number of samples of Wave object <code>x</code> is larger.
...	Further arguments to be passed to the underlying plot functions.

**Details**

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

**Author(s)**

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

**See Also**

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

---

plot-Wspec

*Plotting Wspec objects*

---

**Description**

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

**Usage**

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

**Arguments**

<code>x</code>	Object of class <a href="#">Wspec</a> .
<code>which</code>	Integer indicating which of the periodograms contained in object <code>x</code> to plot. Default is to plot the first one.
<code>type</code>	The default is to plot horizontal lines, rather than points. See <a href="#">plot.default</a> for details.
<code>xlab</code> , <code>ylab</code>	Label for x-/y-axis.
<code>log</code>	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.
<code>...</code>	Further arguments to be passed to the underlying plot functions. See <a href="#">plot.default</a> for details.

**Author(s)**

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

**See Also**

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

**Description**

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

**Usage**

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

**Arguments**

x	Object of class <a href="#">WspecMat</a> (for plot) or <a href="#">Wspec</a> (for image).
xlab, ylab	Label for x-/y-axis.
xunit	Character indicating which units are used to annotate the x axis. If xunit = "time", the unit is time in seconds, otherwise the number of samples.
log	Character - "z" if the z values are to be logarithmic.
...	Further arguments to be passed to the underlying <a href="#">image</a> function. See <a href="#">image</a> 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**

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

**Arguments**

x	Matrix of spectra (output of <a href="#">audspec</a> ).
fmax	Maximum frequency in 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.
eq1	Vector of the equal loudness curve.

**Author(s)**

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

**References**

Daniel P. W. Ellis <http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/>, Hynek Hermansky

**See Also**

[audspec](#), [dolpc](#)

**Examples**

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

---

powspec

*Powerspectrum*

---

### 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: <http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/>

### See Also

[specgram](#)

### Examples

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

---

 prepComb

*Preparing the combination/concatenation of Wave objects*


---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

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

**Value**

An object of class `Wave`.

**Note**

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

**Author(s)**

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

**See Also**

`bind`, `Wave-class`, `Wave`, `extractWave`, and `noSilence` to cut off silence

**Examples**

```

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

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

```

---

quantize

*Functions for the quantization of notes*


---

**Description**

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

**Usage**

```

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

```

**Arguments**

notes	Series of notes, a vector of integers such as returned by <a href="#">noteFromFF</a> . 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 <a href="#">Wspec</a> object.
parts	Number of outgoing parts. The notes vector is divided into parts bins, the outcome is a vector of the modes of all bins.
minlength	1/(length of the shortest note). Example: if the shortest note is a quaver (1/8), set minlength=8.
barsize	One bar contains barsize number of notes of length minlength.
bars	We expect bars number of bars.

**Value**

quantize returns a list with components:

notes	Vector of length parts corresponding to the input data. The data is binned and modes corresponding to the data in those bins are returned.
energy	Same as notes, but for the energy argument.

quantMerge returns a data.frame with components:

note	integer representation of a note (see Arguments).
duration	1/duration of a note (see minlength in Section Arguments), if punctuation=FALSE.
punctuation	Whether the note should be punctuated. If TRUE, the real duration is 1.5 times the duration given in duration.
slur	currently always FALSE, sensible processing is not yet implemented. It is supposed to indicate the beginning and ending positions of slurs.

### Author(s)

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

### See Also

to get the input: [noteFromFF](#), for plotting: [quantplot](#), for further processing: [lilyinput](#), to get notenames: [notenames](#); for an example, see the help in [tuneR](#).

---

quantplot

*Plotting the quantization of a melody*

---

### Description

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

### Usage

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

**Arguments**

observed	Either a vector of observed notes resulting from some quantization, or a list with components notes (observed notes) and energy (corresponding energy values), e.g. the result from a call to <a href="#">quantize</a> .
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 <a href="#">par</a> for details.
las	Orientation of axis labels, see <a href="#">par</a> for details.
cex.axis	Size of tick mark labels, see <a href="#">par</a> for details.
mar	Margins of the plot, see <a href="#">par</a> 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 <a href="#">axis</a> 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 <a href="#">mtext</a> call.
energypar	A list of parameters to be passed to the <a href="#">points</a> function that draws the energy values.
expectedpar	A list of parameters to be passed to the <a href="#">rect</a> function that draws the rectangles for expected values.
gridpar	A named list of two other lists (gridbar and gridinner) containing parameters passed to the <a href="#">abline</a> functions that draw the grid lines (for bar separators and inner bar (note) separators).
observedpar	A list of parameters to be passed to the <a href="#">lines</a> 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](#).

---

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>

**References**

The decoder source code is taken from the MAD library, see <http://www.underbit.com/products/mad/>.

**See Also**

[Wave](#)

## Examples

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

---

show-WaveWspec-methods

*Showing objects*

---

## Description

Showing Wave, Wspec, and WspecMat objects.

## Methods

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

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

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

## Author(s)

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

## See Also

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

---

smoother *Meta Function for Smoothers*

---

### Description

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

### Usage

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

### Arguments

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

```
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**

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

**See Also**

[lpc2cep](#)

**Examples**

```
testsound <- normalize(sine(400) + sine(1000) + square(250), "16")
pspectrum <- powspec(testsound@left, testsound@samp.rate)
aspectrum <- audspec(pspectrum, 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 showed, and an additional summary of the `Wave`-object's (one or two) channels is given.
- object = "Wspec"** The `Wspec` object is being showed, and as an additional output is given: `df`, `taper` (see `spectrum`), and for the underlying `Wave` object the number of channels and its sampling rate.
- object = "WspecMat"** The `WspecMat` object is being showed, and as an additional output is given: `df`, `taper` (see `spectrum`), and for the underlying `Wave` object the number of channels and its sampling rate.

**Author(s)**

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

**See Also**

For the S3 generic: [summary.default](#), [plot-methods](#), [Wave-class](#), [Wave](#), [Wspec](#), [WspecMat](#)

---

 tuneR

*tuneR*


---

**Description**

tuneR, a collection of examples

**Functions in tuneR**

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

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

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

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

**Author(s)**

Uwe Ligges, <[ligges@statistik.tu-dortmund.de](mailto:ligges@statistik.tu-dortmund.de)>

**Examples**

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

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

## End(Not run)

tmpfile <- file.path(tempdir(), "testfile.wav")
# write the Wave object into a Wave file (can be played with any player):
```

```

writeWave(Wobj, tmpfile)
# reading it in again:
Wobj2 <- readWave(tmpfile)

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

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

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

# apply some quantization (into 8 parts):
qnotes <- quantize(snotes, WspecObject@energy, parts = 8)
# an plot it, 4 parts a bar (including expected values):
quantplot(qnotes, expected = rep(c(0, -12), each = 4), bars = 2)
# now prepare for LilyPond
qlily <- quantMerge(snotes, 4, 4, 2)
qlily

```

**Description**

Constructors and coercion for class Wave objects

**Usage**

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

**Arguments**

```
left, right, samp.rate, bit
    See Section "Slots".
...
    Further arguments to be passed to the default method Wave.default.
```

**Value**

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

**Author(s)**

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

**See Also**

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

**Examples**

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

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

---

Wave-class

*Class Wave*

---

**Description**

Class "Wave"

**Objects from the Class**

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

**Slots**

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

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

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

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

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

**Author(s)**

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

**See Also**

[Wave](#)

---

Waveforms

*Create Wave Objects of Special Waveforms*

---

**Description**

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

**Usage**

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

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

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

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

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

**Arguments**

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

**Value**

A [Wave](#) object.

**Author(s)**

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

**See Also**

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

**Examples**

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

---

WaveIO *Reading and writing Wave files*

---

**Description**

Reading and writing Wave files.

**Usage**

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

**Arguments**

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

**Value**

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

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

**Author(s)**

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

**See Also**

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

## Examples

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

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

---

WavPlayer

*Getting and setting the default player for Wave files*

---

## Description

Getting and setting the default player for Wave files

## Usage

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

## Arguments

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

## Value

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

## Author(s)

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

## See Also

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

---

Wspec-class

*Class Wspec*

---

### Description

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

### Details

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

### Objects from the Class

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

### Slots

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

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

### Author(s)

Uwe Ligges, <[ligges@statistik.tu-dortmund.de](mailto:ligges@statistik.tu-dortmund.de)>

**See Also**

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

---

WspecMat-class

---

Class *WspecMat*


---

**Description**

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

**Details**

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

**Objects from the Class**

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

**Slots**

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

freq: Object of class "numeric"

spec: Object of class "matrix"

kernel: Object of class "ANY"

df: Object of class "numeric"

taper: Object of class "numeric"

width: Object of class "numeric"

overlap: Object of class "numeric"

normalize: Object of class "logical"

starts: Object of class "numeric"

stereo: Object of class "logical"

samp.rate: Object of class "numeric"

variance: Object of class "numeric"

energy: Object of class "numeric"

**Author(s)**

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

**See Also**

the show, plot and summary methods

---

*[-methods*

*Extract or Replace Parts of an Object*

---

**Description**

Operators act on objects to extract or replace subsets.

**See Also**

[Extract](#) for the S3 generic.

# Index

- \*Topic **IO**
  - play-methods, 29
  - readMP3, 40
  - WaveIO, 49
- \*Topic **aplot**
  - plot, 30
  - plot-Wave, 31
- \*Topic **arith**
  - Arith-methods, 3
- \*Topic **classes**
  - Wave-class, 46
  - Wspec-class, 51
  - WspecMat-class, 52
- \*Topic **datagen**
  - Waveforms, 47
- \*Topic **documentation**
  - tuneR, 44
- \*Topic **error**
  - equalWave, 8
- \*Topic **file**
  - lilyinput, 14
  - readMP3, 40
  - WaveIO, 49
- \*Topic **hplot**
  - melodyplot, 19
  - plot, 30
  - plot-Wave, 31
  - plot-Wspec, 32
  - plot-WspecMat, 33
  - quantplot, 38
- \*Topic **interface**
  - lilyinput, 14
  - play-methods, 29
- \*Topic **iplot**
  - extractWave, 9
- \*Topic **manip**
  - bind, 4
  - channel, 5
  - downsample, 8
  - extractWave, 9
  - Mono-Stereo, 21
  - normalize, 22
  - noSilence, 23
  - panorama, 26
  - prepComb, 36
- \*Topic **methods**
  - [-methods, 53
  - Arith-methods, 3
  - length, 13
  - play-methods, 29
  - plot, 30
  - plot-Wave, 31
  - plot-Wspec, 32
  - plot-WspecMat, 33
  - show-WaveWspec-methods, 41
  - summary-methods, 43
  - Wave, 45
- \*Topic **misc**
  - smoother, 42
- \*Topic **print**
  - show-WaveWspec-methods, 41
  - summary-methods, 43
- \*Topic **ts**
  - FF, 10
  - melfcc, 17
  - MFCC, 21
  - periodogram-methods, 27
  - smoother, 42
- \*Topic **utilities**
  - bind, 4
  - channel, 5
  - downsample, 8
  - equalWave, 8
  - extractWave, 9
  - Mono-Stereo, 21
  - noSilence, 23
  - noteFromFF, 24
  - notenames, 25

- play-methods, 29
- prepComb, 36
- quantize, 37
- WavPlayer, 50
- [, ANY-method ([-methods), 53
- [, Wave-method (Wave), 45
- [, Wspec-method (Wspec-class), 51
- [, WspecMat-method (WspecMat-class), 52
- [-methods, 53
  
- abline, 20, 39
- Arith, numeric, Wave-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-methods, 3
- audspec, 3, 34
- axis, 20, 39
  
- bark2hz (freqconv), 12
- bind, 4, 10, 36, 44
  
- channel, 5, 10, 44
- coerce, data.frame, Wave-method (Wave), 45
- coerce, list, Wave-method (Wave), 45
- coerce, matrix, Wave-method (Wave), 45
- coerce, numeric, Wave-method (Wave), 45
- coerce, Wave, data.frame-method (Wave), 45
- coerce, Wave, list-method (Wave), 45
- coerce, Wave, matrix-method (Wave), 45
- coerce, Wspec, WspecMat-method (WspecMat-class), 52
  
- decmedian, 42
- deltas, 6
- dolpc, 7, 34
- downsample, 8, 28, 44
  
- equalWave, 4, 8
- Extract, 53
- extractWave, 5, 9, 24, 36, 44
  
- FF, 10, 20, 25, 40, 44
- FFpure, 44
- FFpure (FF), 10
- fft2barkmx, 4
- fft2melmx, 4
- freqconv, 12
  
- getWavPlayer (WavPlayer), 50
- groupGeneric, 3
  
- hz2bark (freqconv), 12
- hz2mel (freqconv), 12
  
- image, 33
- image, ANY-method (plot-WspecMat), 33
- image, Wspec-method (plot-WspecMat), 33
- image-methods (plot-WspecMat), 33
- image-Wspec (plot-WspecMat), 33
- interactive, 10
  
- length, 13, 13
- length, ANY-method (length), 13
- length, Wave-method (length), 13
- length-methods (length), 13
- levinson, 7
- lifter, 13
- lilyinput, 14, 38, 44
- lines, 20, 39
- lpc2cep, 16, 43
  
- mel2hz (freqconv), 12
- melfcc, 17, 21
- melodyplot, 19, 40, 44
- MFCC, 21
- mono, 5, 10, 28, 44
- mono (Mono-Stereo), 21
- Mono-Stereo, 21
- mtext, 20, 39
  
- noise (Waveforms), 47
- normalize, 22, 48, 49
- noSilence, 23, 36, 48
- noteFromFF, 12, 19, 20, 24, 37, 38, 40, 42, 44
- notenames, 25, 38
  
- options, 50
  
- panorama, 26
- par, 20, 31, 39
- periodogram, 12, 25, 32, 33, 41, 44, 51, 52
- periodogram (periodogram-methods), 27
- periodogram, character-method (periodogram-methods), 27

- periodogram, Wave-method
  - (periodogram-methods), 27
- periodogram-methods, 27
- play, 44, 50
- play (play-methods), 29
- play, character-method (play-methods), 29
- play, Wave-method (play-methods), 29
- play-methods, 29
- plot, 30
- plot, ANY, ANY-method (plot), 30
- plot, Wave, missing-method (plot-Wave), 31
- plot, Wspec, missing-method (plot-Wspec), 32
- plot, WspecMat, missing-method
  - (plot-WspecMat), 33
- plot-methods, 9, 10, 41, 44
- plot-Wspec, 29
- plot-methods (plot), 30
- plot-Wave, 31
- plot-Wspec, 32
- plot-WspecMat, 33
- plot.default, 30, 32
- plot.Wave.channel (plot-Wave), 31
- points, 39
- postaud, 34
- powspec, 3, 35
- prepComb, 5, 36
  
- quantize, 16, 37, 39, 40, 44
- quantMerge, 16, 44
- quantMerge (quantize), 37
- quantplot, 16, 20, 38, 38, 44
  
- readMP3, 40
- readWave, 44, 46
- readWave (WaveIO), 49
- rect, 20, 39
- round, 25
  
- sawtooth (Waveforms), 47
- setWavPlayer, 30
- setWavPlayer (WavPlayer), 50
- show, Wave-method
  - (show-WaveWspec-methods), 41
- show, Wspec-method
  - (show-WaveWspec-methods), 41
- show, WspecMat-method
  - (show-WaveWspec-methods), 41
- show-WaveWspec-methods, 41
  
- silence, 24
- silence (Waveforms), 47
- sine, 44
- sine (Waveforms), 47
- smoother, 42, 44
- spec.pgram, 27–29, 52
- spec2cep, 16, 42
- specgram, 35
- spectrum, 28, 43, 51, 52
- square (Waveforms), 47
- stereo, 5, 44
- stereo (Mono-Stereo), 21
- stop, 9
- summary, ANY-method (summary-methods), 43
- summary, Wave-method (summary-methods), 43
- summary, Wspec-method (summary-methods), 43
- summary, WspecMat-method
  - (summary-methods), 43
- summary-methods, 41
- summary-methods, 43
- summary.default, 44
  
- tuneR, 12, 16, 20, 25, 32, 33, 38, 40, 44
- tuneR-package (tuneR), 44
  
- Wave, 3–5, 8–10, 13, 21–24, 26–32, 36, 40, 41, 43, 44, 45, 46–52
- Wave, ANY-method (Wave), 45
- Wave, data.frame-method (Wave), 45
- Wave, list-method (Wave), 45
- Wave, matrix-method (Wave), 45
- Wave, numeric-method (Wave), 45
- Wave-class, 3, 5, 8–10, 22–24, 27, 29, 30, 32, 36, 41, 44, 46, 46, 48–50, 52
- Wave-methods (Wave), 45
- Waveforms, 47
- WaveIO, 49
- WavPlayer, 50
- writeWave, 23, 30, 44, 46
- writeWave (WaveIO), 49
- Wspec, 10–12, 19, 28, 29, 32, 33, 37, 41, 43, 44, 52
- Wspec (Wspec-class), 51
- Wspec-class, 51
- WspecMat, 33, 41, 43, 44, 52
- WspecMat (WspecMat-class), 52
- WspecMat-class, 52