Package ‘phonTools’

November 20, 2023

Title Tools for Phonetic and Acoustic Analyses

Version 0.2-2.2

Date 2023-11-14

Author Santiago Barreda

Maintainer Santiago Barreda <sbarreda@ucdavis.edu>

Depends R (>= 2.15)

Description Contains tools for the organization, display, and analysis of the sorts of data frequently encountered in phonetics research and experimentation, including the easy creation of IPA vowel plots, and the creation and manipulation of WAVE audio files.

License BSD_2_clause + file LICENSE

NeedsCompilation no

Repository CRAN

Date/Publication 2023-11-20 17:20:02 UTC

R topics documented:

`a96` *Aronson et al. (1996) Hebrew Vowel Data*

Description

Formant frequency information for vowels averaged across 6 male speakers.

Usage

```
data (a96)
```

Format

A data frame with the following columns:

`sex` - A factor indicating speaker sex.

`vowel` - The vowel category in x-sampa.

`f1` - A numeric vector indcating the vowel F1 in Hz.

`f2` - A numeric vector indcating the vowel F2 in Hz.

`f3` - A numeric vector indcating the vowel F3 in Hz.

`f4` - A numeric vector indcating the vowel F4 in Hz.

References

Aronson, L., Rosenhouse, J. Rosenhouse, G. & Podoshin, L. (1996). An acoustic analysis of modern Hebrew vowels and voiced consonants. Journal of Phonetics 24. 283-193.

Examples

```
#data(a96)
#vowelplot (a96$f1, a96$f2, a96$vowel, logaxes = 'xy', xsampa = TRUE)
```

`b95` *Bradlow (1995) Spanish Vowel Data*

Description

Formant frequency information for vowels averaged across 4 male speakers.

Usage

```
data (b95)
```

Format

A data frame with the following columns:

`sex` - A factor indicating speaker sex.

`vowel` - The vowel category in x-sampa.

`f1` - A numeric vector indcating the vowel F1 in Hz.

`f2` - A numeric vector indcating the vowel F2 in Hz.

References

Bradlow, A. R. (1995). A comparative acoustic study of English and Spanish vowels. Journal of the Acoustical Society of America 97. 1916-1924.

Examples

```
#data(b95)
#vowelplot (b95$f1, b95$f2, b95$vowel, logaxes = 'xy', xsampa = TRUE)
```

`combocalc` *Combinations and Permutations*

Description

Calculate the number of combinations or permutations for a number of objects.

Usage

```
combocalc (objects, choose, order = FALSE, repetition = TRUE)
```

Arguments

`objects`	The number of different kinds of objects available for selection.
`choose`	The number of objects selected at a given time.
`order`	If TRUE, the order of the objects matters, for example aba != aab.
`repetition`	If TRUE, a sequence such as bbb is permissible.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
# combinations, no repetition
combocalc (10, 4, order = FALSE, repetition = FALSE)
# combinations, with repetition
combocalc (10, 4, order = FALSE, repetition = TRUE)
# permutations, no repetition
combocalc (10, 4, order = TRUE, repetition = FALSE)
# permutations, with repetition
combocalc (10, 4, order = TRUE, repetition = TRUE)
```

`createtemplate` *Create an LDA Template*

Description

Create a linear discriminant analysis template.

Usage

```
createtemplate (features, classes)
```

Arguments

`features`	A matrix of features in which each row represents a single token and each column represents a different 'feature' used to classify the token. For vowel sounds, each column should represent different single formant frequency.
`classes`	A vector indicating the category of each token described in each row of 'features'. The length of this vector must equal the number of rows in 'features'.

Details

This function finds the location of the mean of each class in an n-dimensional space, where each dimension corresponds to one of n columns in 'features'. In addition, the pooled, within-category covariance matrix is found.

The name for each vowel category is stored as the rownames of the 'means' element.

The function plot() is defined for template objects and allows the user to view the location of and expected variation around the different vowel categories in the formant space.

This information may be used in conjunction with the PSTM() function, included in this package. The mean and covariance matrices provided by this function may also be useful for the ldclassify() function provided in this package.

Value

A 'template' object, a list containing the elements:

`classes`	The category labels.
`means`	A matrix containing the mean location for each vowel category within the formant-space. Each row represents a different category while each column represents a different formant.
`covariance`	The pooled, within-category covariance matrix for the formant frequencies provided.
`ranges`	A matrix of dimension ranges, one row for each dimension.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Nearey, T. M. & P. F. Assmann. (2007). Pobabilistic 'sliding template' models for indirect vowel normalization. in Experimental Approaches to Phonology, edited by M.J. Sole, P. S., Beddor, and M. Ohala (Oxford University Press, Oxford), pp. 246-269.

Examples

```
#data (pb52)            ## load the Peterson and Barney vowels.
## normalize them.
#normdvowels = normalize (pb52[,7:9], pb52$speaker, pb52$vowel)
#formants = normdvowels[,1:3]
#vowels = pb52$vowel

## create a vowel template with the normalized formant frequencies
## and information about the vowel category.
#template = createtemplate (formants, vowels)

## and inspect with plot()
#plot (template, xsampa = TRUE)
```

`errorbars` *Error bars.*

Description

This functions adds error bars to a plot or, optionally, plots points and added error bars.

Usage

```
errorbars (x, y, top, bottom = top, length = .2, add = TRUE, ...)
```

Arguments

`x`	X axis positions.
`y`	Y axis positions.
`top`	Top error bar length.
`bottom`	Bottom error bar length.
`length`	The length of the horizontal bars.
`add`	If TRUE, error bars are added to existing plot.
`...`	Additional arguments go to the internal call of arrows() which draws the error bars.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Inspired by a similar function first seen here:

http://monkeysuncle.stanford.edu/?p=485

Examples

```
## add bars to existing plots
#plot (1:10, 1:10, col = 4, pch = 18)
#errorbars (1:10, 1:10, .5, col = 4)

## or create a new plot
#errorbars (1:10, 1:10, 2, add = FALSE)
```

`f73` *Fant (1973) Swedish Vowel Data*

Description

Formant frequency information for vowels averaged across 50 male speakers.

Usage

```
data (f73)
```

Format

A data frame with the following columns:

`sex` - A factor indicating speaker sex.

`vowel` - The vowel category in x-sampa.

`f1` - A numeric vector indcating the vowel F1 in Hz.

`f2` - A numeric vector indcating the vowel F2 in Hz.

`f3` - A numeric vector indcating the vowel F3 in Hz.

`f4` - A numeric vector indcating the vowel F4 in Hz.

References

Fant, G. (1973). Speech sounds and features. Cambridge, MA: MIT Press.

Examples

```
#data(f73)
#vowelplot (f73$f1, f73$f2, f73$vowel, logaxes = 'xy', xsampa = TRUE)
```

`f99` *Fourakis et al. (1999) Greek Vowel Data*

Description

Formant frequency information for vowels averaged across 5 male speakers.

Usage

```
data (f99)
```

Format

A data frame with the following columns:

`sex` - A factor indicating speaker sex.

`vowel` - The vowel category in x-sampa.

`f1` - A numeric vector indcating the vowel F1 in Hz.

`f2` - A numeric vector indcating the vowel F2 in Hz.

References

Fourakis, M., Botinis, A. & Katsaiti, M. (1999). Acoustic characteristics of Greek vowels. Phonetica, 56. 28-43.

Examples

```
#data(f99)
#vowelplot (f99$f1, f99$f2, f99$vowel, logaxes = 'xy', xsampa = TRUE)
```

`fastacf` *Fast Autocorrelation*

Description

Compute the ACF of a signal.

Usage

```
fastacf  (signal, lag.max = length(signal), window = 'hann',
          show = TRUE, correct = FALSE)
```

Arguments

`signal`	The signal, a numeric vector.
`lag.max`	The maximum lag value to be returned.
`window`	The type of window to be applied to the signal. Uses the windowfunc() function in this package. For no window select 'rectangular'.
`show`	If TRUE, the results are plotted.
`correct`	If TRUE, the output is corrected based on the window length and the window function that is applied.

Details

The autocorrelation function is calculated using the inverse Fourier transform applied to the power spectrum. This leads to much faster calculation times than the acf() function included in the typical R installation. Corrections for window type and length are carried out as described in Boersma (1993).

Value

A dataframe with the following columns:

`lag`	Indicates the lag value.
`acf`	Indicates the ACF value at the lag.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Boersma, P., (1993). Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. Proc. Instit. Phon. Sci. 17: 97-110.

Examples

```
## Uncomment and run the code below to see the speed advantage.
## Raising the n makes the difference even more pronounced.
#n = 25000
#system.time ({
#acf (rnorm (n), plot = F, lag.max = n)
#})

#system.time ({
#fastacf (rnorm (n), plot = F, lag.max = n)
#})
```

Ffilter *Formant Filtering*

Description

Perform cascade formant filtering of sounds.

Usage

```
Ffilter (sound, ffs, bwp = 0.06, minbw = 60, fs = 22050, verify = FALSE)
```

Arguments

sound	A numeric vector representing a waveform, or a 'sound' object created with the loadsound() or makesound() functions.
ffs	A list of vectors of initial and final center frequencies for each formant. Each vector should contain formant frequencies in order of lowest to highest frequency. If only a single vector is provided, formants will remain stable throughout.
bwp	A vector of formant bandwidths, one for each formant.
minbw	The minimum permissible formant bandwidth, in Hertz.
fs	The sampling frequency of the sound. If a 'sound' object is passed, this does not need to be specified.
verify	If TRUE, before and after spectra are plotted to allow the user to visually verify the process.

Details

This function allows the user to specify one or more formant filters and to pass a signal through said filters. This may be used to create synthetic speech sounds or to modify existing sounds as desired. The given signal is passed through the formant filters in reverse order. Filter bandwidths specify the distance between formant center frequencies and the point at which the output will be 3 dB below peak energy.

Filter bandwidths may be provided in Hz, or as a percentage of the formant frequencies. To set these as a percent of formant frequencies, all values must be less than 1. If these are not provided they are set to 6 percent of the formant center frequencies by default. If only one value is provided, this is assumed to be the desired value for all formants.

Value

A vector representing the filtered sound.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Klatt, D. H. (1980). Software for a cascade/parallel formant synthesizer. Journal of the Acoustical Society of America 67(3): 971-995.

http://www.fon.hum.uva.nl/praat/manual/Sound__Filter__one_formant____.html

Examples

```
## uncomment and run
## Generate half a second of white noise
#sound = rnorm (11025, 0, 1)

## pass this through some formant filters
## two static formants
#filtered1 = Ffilter (sound, ffs = c(4000,7000), bw = 500)
## a single moving formant
#filtered2 = Ffilter (sound, ffs = list(3000,8000), bw = 500)
## two moving formants


## inspect the results
#par (mfrow = c(1,2), mar = c(4,4,1,1))
#spectrogram (filtered1, maxfreq = 11025)
#spectrogram (filtered2, maxfreq = 11025)
```

`findformants` *Find Formants*

Description

Find formants given a sound or set of LPC coefficients.

Usage

```
findformants (sound, fs = 10000, coeffs = NULL, maxbw = 600,
minformant = 200, verify = TRUE, showbws = FALSE,
showrejected = TRUE)
```

Arguments

`sound`	A numeric vector representing a waveform, or a 'sound' object created with the loadsound() or makesound() functions.
`fs`	The sampling frequency of the sound. If a 'sound' object is passed this does not need to be specified.
`coeffs`	If a number is given, this many coefficients are used in the analysis. Alternatively, the LPC (AR filter) coefficients may be passed to the function directly using this parameter. For good results, two coefficients are required for each formants plus 2 or 3 'for the pot'.
`maxbw`	The maximum bandwidth for accepted formants.
`minformant`	Formants below this frequency are rejected.
`verify`	If TRUE, a plot is created which allows the user to visually inspect the process.
`showbws`	If TRUE, formant bandwidths are indicated on the plot.
`showrejected`	If TRUE, rejected formant locations are indicated.

Details

Formant frequencies are found analytically using the formulas provided in Snell (1993). If Verify = TRUE, the estimated frequency response, formant locations, and a pole-zero plot of the estimated filter are presented. Accepted formants are presented in 5 colors (which are reused if there are more than 5 formants), while rejected formants are presented in black.

Value

A dataframe with the following elements is returned:

`frequency`	The frequencies of formants, in ascending order.
`bandwidth`	The corresponding formant bandwidth.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Snell, R.(1993). "Formant location from LPC analysis data", IEEE Transactions on Speech and Audio Processing, 1(2), pp. 129-134.

Examples

```
## make a synthetic vowel with a known set of formant frequencies
## and bandwidths
#sound = vowelsynth (ffs = c(500,1500,2500,3500,4500),
#                    fbw = c(30, 90, 150, 210, 270), f0 = 100)

## compare different plotting options
#findformants (sound)
#findformants (sound, showrejected = FALSE)
#findformants (sound, showbws = TRUE)
```

`FIRfilter` *Perform Digital Filtering*

Description

Finite Impulse Response (FIR) filtering of vectors.

Usage

```
FIRfilter (sound, from = 0, to = fs/2, fs = 22050, order = 200,
verify = FALSE, impulse = NULL, pad = TRUE)
```

Arguments

`sound`	A numeric vector representing a waveform, or a 'sound' object created with the loadsound() or makesound() functions.
`from`	The low cutoff point for the filter. Frequencies higher than this are not attenuated. Minimum allowed value is 0 Hz.
`to`	The high cutoff point for the filter. Frequencies lower than this are not attenuated. Maximum allowed value is fs/2 Hz.
`fs`	The sampling frequency of the sound. If a 'sound' object is passed, this does not need to be specified.
`order`	The number of taps included in the filter. The order of the filter may not exceed the number of samples that make up the sound.
`verify`	If TRUE, a plot comparing the spectrum of the input sound is compared the the filtered sound.
`impulse`	If a filter impulse response is specified, this is used to filter the signal.
`pad`	If TRUE, the signal is padded with zero so that the original length is maintained.

Details

This function performs lowpass, highpass and bandpass filtering using a windowed-sinc filter. Increasing the filter order decreases the transition region between the passband and the stopband. The magnitude of frequencies in the stopband is usually attenuated by about about 45 dB.

If verify is TRUE, a plot is created which allows the user to inspect the performance of the function.

Value

If a vector is given, the filtered vector is returned.

If a 'sound' object is given, a sound object containing the filtered sound is returned.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/Sinc_filter

Lyons, R. G. (2004). Understanding Digital Signal Processing (2nd ed.). Prentice Hall.

Examples

```
## generate random noise
#sound = rnorm (5000, 0, 100)

## implement low, high and band-pass filters
#par (mfrow = c(3,1), mar = c(4,4,1,1))
#snd1 = FIRfilter (sound, from = 400, fs = 1000, verify = T)
#snd2 = FIRfilter (sound, to = 400, fs = 1000, verify = T)
#snd3 = FIRfilter (sound, from = 400, to = 100, fs = 1000, verify = T)

## use filters of different orders (i.e., differing number of taps)
#par (mfrow = c(2,1), mar = c(4,4,1,1))
#snd1 = FIRfilter (sound, to = 400, fs = 1000, order = 50, verify = T)
## higher order filters lead to narrower transition regions
#snd2 = FIRfilter (sound, to = 400, fs = 1000, order = 2000, verify = T)
```

`formanttrack` *Formant Tracking*

Description

Create a formant track for a sound.

Usage

```
formanttrack (sound, timestep = 5, windowlength = 30,
formants = 5, cutoff = 5000, minformant = 200, maxbw = 600,
fs = 22050, show = TRUE, periodicity = .5, returnbw = FALSE)
```

Arguments

`sound`	A numeric vector representing a waveform, or a 'sound' object created with the loadsound() or makesound() functions.
`timestep`	How far the analysis window will advance, in milliseconds. If this value is set to zero, the function is applied to the entire signal.
`windowlength`	The length of the analysis window. Longer windows will result in increased accuracy with decreased time-resolution.
`formants`	The desired number of formants to be tracked. Depending on the sound and time point, fewer than this many may be found.
`cutoff`	The maximum analysis frequency. May not be higher than the Nyquist frequency.
`minformant`	Formants below this frequency are rejected.
`maxbw`	The maximum bandwidth for accepted formants.
`fs`	The sampling frequency of the sound. If a 'sound' or 'ts' object is passed, this does not need to be specified.
`show`	If TRUE, a plot is created which allows the user to visually inspect the process.
`periodicity`	A value between 0 and 1. Signal sections with corrected ACF values lower than this are not analyzed. Allows voiceless sections to be excluded from analysis.
`returnbw`	If TRUE, estimated formant bandwidths are returned.

Details

This function works by repeatedly calling findformants(), and periodicity is established using pitchtrack(), both of which are included in this package. When no formants are found, or if less than the desired number of formants are found, a value of zero is returned for that formant, at that time point.

Tracked formants are presented over a greyscale spectrogram if show is TRUE. When plotting, different colors are used for each formant to allow the user to distinguish these.

Value

A dataframe with the following elements is returned:

`time`	The time, in milliseconds, of the middle of the analysis window.
`f#`	The formant frequency for formant number #, one column for each formant.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
#data (sound)
#par (mfrow = c(2,1), mar = c(4,4,1,1))
#formanttrack (sound)
#formanttrack (sound, periodicity = 0)
```

---

freqresponse *Frequency Response*

Description

Find the frequency response of a digital filter

Usage

```
freqresponse (b, a, fs = 0, add = FALSE, show = TRUE,
              steps = 1000,...)
```

Arguments

b	The moving-average (MA), numerator coefficients of the filter.
a	The autoregressive (AR), denominator coefficients of the filter. Please note that the leading 1 at a[0] is not assumed.
fs	The sampling frequency of the sound. If this is not given calculations are presented as if fs = 1.
add	If TRUE, the frequency response plot is added to an existing plot.
show	If TRUE, the frequency response of the estimated filter is plotted.
steps	The number of steps between zero and the Nyquist frequency.
...	Additional arguments are passed to internal plotting functions.

Details

This function plots (and optionally returns) the frequency response for the digital filter whose transfer function is determined by the numerator and denominator filter coefficients given in b and a.

Value

A dataframe with two columns (frequency and response) that can be used to redraw the frequency response if required. The 'response' value corresponds to dB. magnitude below peak.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Lyons, R. G. (2004). Understanding Digital Signal Processing (2nd ed.). Prentice Hall.

Examples

```
## make a synthetic vowel with a known set of formant frequencies
#sound = vowelsynth (ffs = c(500,1500,2500,3500,4500),
#fbw = c(30, 80, 150, 200, 220),f0 = 100, dur = 100)

#plot (sound)

## let the LPC function estimate the filter used to generate the vowel
#coeffs = lpc (sound, show = FALSE)

## compare frequency response of estimated filter to vowel spectrum
#spectralslice (sound, col = 4, preemphasisf = 50)
#freqresponse (1, coeffs, add = TRUE, fs = 10000)

## generate a sinc function
#filt = sinc (seq (-15,15,1/2), normalized = TRUE)
## treat it as a low-pass FIR filter and inspect its frequency response
#freqresponse (filt, 1)
```

`h95` *Hillenbrand et al. (1995) Vowel Data*

Description

Formant frequency, f0 and duration information for vowels collected from 139 speakers in the Hillenbrand et al. (1995) data. Speaker numbers have been modified to be uniquely identifying numbers. Data has been simplified so that only "steady state" formant frequency measures are given. Missing F2 values (n = 10) and F3 values (n = 41) have been imputed using the imputeformants() function included in this package.

Usage

```
data (h95)
```

Format

A data frame with 1668 observations on the following 9 variables:

`type` - A factor with levels `b` `g` `m` `w` representing speaker type: boy, girl, man and woman.

`speaker` - A numeric vector indicating a uniquely identifying speaker number.

`vowel` - The vowel category in x-sampa

`dur` - A numeric vector indicating the duration of the vowel in milliseconds.

`f0` - A numeric vector indcating the vowel f0 in Hz.

`f1` - A numeric vector indcating the vowel F1 in Hz.

`f2` - A numeric vector indcating the vowel F2 in Hz.

`f3` - A numeric vector indcating the vowel F3 in Hz.

Source

The data was created from data provided on Dr. Hillenbrand's personal website:

http://homepages.wmich.edu/~hillenbr/voweldata.html

References

Hillenbrand, J.M., Getty, L.A., Clark, M.J., and Wheeler, K. (1995). "Acoustic characteristics of American English vowels," Journal of the Acoustical Society of America, 97, 3099-3111.

Examples

```
#data(h95)
#vowelplot (h95$f1, h95$f2, h95$vowel, logaxes = 'xy', ellipses = TRUE,
#xsampa = TRUE)
```

`hotelling.test` *Hotelling's T2 Test*

Description

Hotelling's T2 test for one and two samples.

Usage

```
hotelling.test(matrix1, matrix2 = NULL)
```

Arguments

`matrix1`	A numeric matrix or dataframe in which each row represents an observation of a multivariate random variable, and each column represents a dimension of that variable.
`matrix2`	An optional second numeric matrix or dataframe of the same column rank as 'matrix1'.

Details

If a single matrix is provided, this function tests the alternative hypothesis that all column means are not equal to zero. If a second matrix is provided, the alternative hypothesis to be tested is that the group means are not all equal. The statistic is tested using an F-distribution which assumes that the matrices represent (roughly) multivariate normal variables.

This function is only designed for multivariate tests of location. If a univariate test is desired, please use a t-test.

Value

An object of class 'Hotelling.test', a list containing the elements:

`f.value`	The value of the test statistic.
`df1`	The numerator degrees of freedom for the F statistic.
`df2`	The denominator degrees of freedom for the F statistic
`p.value`	The p-value for the test.
`samples`	The number of independent samples involved in the test.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Hotelling, H. (1931). The generalization of Student's ratio. Annals of Mathematical Statistics 2 (3): 360-378.

http://en.wikipedia.org/wiki/Hotelling's_T-squared_distribution

Examples

```
## load Peterson & Barney data
#data (pb52)

## separate the Peterson & Barney vowels by speaker
## gender and age (child vs. adult)
#men = pb52[pb52$sex == 'm' & pb52$type == 'm',]
#women = pb52[pb52$sex == 'f' & pb52$type == 'w',]
#boys = pb52[pb52$sex == 'm' & pb52$type == 'c',]
#girls = pb52[pb52$sex == 'f' & pb52$type == 'c',]

## fit 4 separate models which predict F1 frequency
## on the basis of vowel category.
#men = rcr (f1 ~ vowel, men$speaker, men)
#women = rcr (f1 ~ vowel, women$speaker, women)
#boys = rcr (f1 ~ vowel, boys$speaker, boys)
#girls = rcr (f1 ~ vowel, girls$speaker, girls)

## A Hotelling T2 test indicates that there are
## significant differences in F1 frequency
## based on vowel category between males and females
#hotelling.test (men$coefficients, women$coefficients)

## but no significant differences based on the same
## criteria between boys and girls.
#hotelling.test (boys$coefficients, girls$coefficients)
```

`imputeformants` *Impute Missing Formant Values*

Description

Impute missing formant values using a least-squares approximation.

Usage

```
imputeformants (ffs, speaker, vowel)
```

Arguments

`ffs`	A numeric vector containing formant frequency measurements for a single formant. Values should be in Hz, and missing values need to be set to 0.
`speaker`	A vector indicating which speaker produced each formant.
`vowel`	A vector indicating which vowel category each formant belongs to.

Details

This function finds the least-squares approximation for each missing value based on the assumption that each formant for a given vowel differs between-speakers solely on the basis on a speaker-specific multiplicative parameter. This assumption is well supported in the literature (Nearey 1978, Nearey & Assmann 2007, Turner et al. 2009). This parameter would be most closely related to gross speaker vocal-tract length.

Value

A vector containing each original formant frequency and imputed formant frequencies where appropriate.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Nearey, T. M. (1978). Phonetic Feature Systems for Vowels. PhD thesis, Indiana University Linguistics Club.

Nearey, T. M. & P. F. Assmann. (2007). Pobabilistic 'sliding template' models for indirect vowel normalization. in Experimental Approaches to Phonology, edited by M.J. Sole, P. S., Beddor, and M. Ohala (Oxford University Press, Oxford), pp. 246-269.

Turner, R. E., Walters, T. C., Monaghan, J. J. M., & Patterson, R. D. (2009). A statistical, formant-pattern model for segregating vowel type and vocal-tract length in developmental formant data. The Journal of the Acoustical Society of America, 125(4), 2374. doi:10.1121/1.3079772

Examples

```
#data (h95)

## Select F2 values from the tenth speaker in H95 data
## set the first 5 values to "missing"
#ffs = h95$f2
#ffs[h95$speaker == 10][1:5] = 0
#speaker = h95$speaker
#vowel = h95$vowel

## impute these missing values
#imputedf2 = imputeformants (ffs, speaker, vowel)

## resulting in a very close approximation of the original values
#plot (imputedf2[h95$speaker == 10], h95$f2[h95$speaker == 10])
#abline (0, 1, col = 2)
```

`interpolate` *Interpolation*

Description

Piece-wise cubic or linear spline interpolation.

Usage

```
interpolate (y, x = 1:length(y), steps = 20, increment = -1,
show = FALSE, type = 'cubic', ...)
```

Arguments

`y`	A vector of 'knots', between which the function will interpolate points.
`x`	The 'x' coordinates corresponding to each knot. If not specified, the knots are assumed to be equally spaced.
`steps`	The number of interpolating steps between each knot. Increasing this number will result in a smoother interpolation. If the knots are not equally spaced along the x-axis, the interpolated points will not be equally spaced across the entire curve.
`increment`	If this is greater than 0, interpolated points are separated along the x-axis by this value. Note that if the knot locations are not multiples of this increment, there will be irregularities in the spacing of the interpolated points.
`show`	If TRUE, the result of the interpolation is shown in a plot.
`type`	If 'cubic', a natural cubic spline interpolation is performed. If 'linear', a linear interpolation is performed.
`...`	Additional arguments are passed to the internal call of plot() if show = TRUE.

Details

By default, this function performs a ’natural’ cubic spline interpolation between the points provided by the user. Optionally, a linear interpolation between the points may be carried out.

Value

A dataframe with columns corresponding to the x and y dimensions of the interpolated points is returned.

x	The x-axis coordinates of the interpolated points.
y	The y-axis coordinates of the interpolated points.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/Spline_interpolation

Examples

```
## generate ten random points
#y = rnorm (10, 0, 5)
#interpolate (y, show = TRUE)  ## plot a cubic interpolation
#linear = interpolate (y, type = 'linear')
## and compare to a linear interpolation
#lines (linear, col = 2)
```

---

ldboundary *Linear Discriminant Boundary*

Description

Given two mean vectors and a covariance matrix (and optional prior probabilities), this function will return the slope and intercept of the boundary line between the two categories.

Usage

```
ldboundary (mean1, mean2, covariance, prior1 = .5, prior2 = .5, add = F, ...)
```

Arguments

`mean1`	A mean vector for category 1, must contain 2 elements.
`mean2`	A mean vector for category 2, must contain 2 elements.
`covariance`	A 2x2 covariance matrix for both distributions.
`prior1`	The prior probability of category 1.
`prior2`	The prior probability of category 2.
`add`	If TRUE, the boundary line is added top the plot.
`...`	Additional parameters are passed to the internal call of the line plotting function, in the event that add = T.

Value

The slope and intercept of the boundary line are returned.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/Linear_discriminant_analysis https://onlinecourses.science.psu.edu/stat557/book/export/html/3

Examples

```
## create two groups with the same covariance patterns
#group1 = rmvtnorm (200, means= c(0,0), k=2, sigma = -.4)
#group2 = rmvtnorm (200, means= c(3,3), k=2, sigma = -.4)
#covariance = (var (group1) + var (group2)) / 2

## plot groups and boundary line between categories.
#plot (group1, col = 2, pch = 16, ylim = c(-2,5), xlim = c(-2,5))
#points (group2, col = 4, pch = 16)
#ldboundary (c(0,0), c(3,3), covariance, add = TRUE)
```

`ldclassify` *Linear Discriminant Classification*

Description

Classify items using linear discriminant analysis.

Usage

```
ldclassify (data, means, covariance, template = NULL, posterior = 'no')
```

Arguments

`data`	A matrix in which each row represents an item to be classified, and each column represents an observation from a variable describing the item.
`means`	A matrix of means where each row is a mean vector specifying a candidate category. The number of columns must equal the number of columns in data.
`covariance`	The pooled within-groups covariance matrix to be used for classification.
`template`	A 'Template' object may be passed instead of a mean and covariance.
`posterior`	If 'winner', the posterior probability of the winning category is returned. If 'all', the posterior of every category is returned.

Details

This function classifies the items described in the data matrix by comparing them to the reference patterns for the different candidate categories represented in the means matrix. The category with the minimum Mahalanobis distance to the observed pattern (i.e., a given row of the data matrix) is selected as the winner. Mahalanobis distances are found with using the covariance matrix provided to the function.

Mean and covariance matrices can be made easily for data using the createtemplate() function included in this package.

Value

A vector of winning categories is returned. If winning posteriors are desired, these are returned in a second column. All posteriors are returned in separate columns for each category.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## load Peterson & Barney vowel data
#data (pb52)

## normalize vowel formant frequencies
#normdvowels = normalize (pb52[,7:9], pb52$speaker, pb52$vowel)
#formants = normdvowels[,1:3]
#vowels = pb52$vowel

## make a vowel template based on these frequencies
#template = createtemplate (formants, vowels)

## classify vowels
#answers = ldclassify (formants, template$means, template$covariance)
## compare to known categories
#table (answers, vowels)
```

`loadsound` *Load WAV files into R*

Description

A function which allows WAV files to be loaded into R.

Usage

```
loadsound (filename)
```

Arguments

`filename`	A string indicating the file name of the WAV file to be loaded. If no filename is provided, a dialog box will open allowing the user to select a file.

Details

The function is only compatible with 8 and 16 bit mono WAV files. The function returns a 'sound object'. Many of the functions included in this package interact with 'sound' objects.

Value

An object of class 'sound', a list containing the elements:

`filename`	A vector containing the filename of the WAV file.
`fs`	The sampling frequency of the sound.
`duration`	The duration of the sound, in milliseconds.
`sound`	A vector of numeric values representing the sampled sound.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

https://ccrma.stanford.edu/courses/422/projects/WaveFormat/

Examples

```
## Use the command below to select a WAV file to load into R
## sound = loadsound ()

## sound
## plot (sound)
## spectrogram (sound)
```

`loadtable` *Load Table*

Description

Load text table data quickly.

Usage

```
loadtable (...)
```

Arguments

`...`	Arguments are passed to the internal call of read.table().

Details

This function is a wrapper for read.table() for those times when you just don't feel like typing a filename. The function opens up a file selection dialog box allowing the user to select the file containing the data.

Value

Returns the output of read.table().

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## uncomment and run
# data = loadtable ()
# head (data)
```

`lpc` *Linear Predictive Coding*

Description

Predict autoregressive filter coefficients.

Usage

```
lpc (sound, order = round(fs/1000)+3, fs = 10000, show = FALSE,
add = FALSE, preemph = TRUE)
```

Arguments

`sound`	Either a numeric vector representing a sequence of samples taken from a sound wave or a sound object created with the loadsound() or makesound() functions.
`order`	The number of LPC coefficients to be estimated. By default there is 2 per kHz below the Nyquist frequency plus 3 extra coefficients.
`fs`	The sampling frequency in Hz. If a sound object is passed this does not need to be specificed.
`show`	If TRUE, the frequency response of the estimated filter is plotted.
`add`	If TRUE, the frequency response plot is added to an existing plot.
`preemph`	If TRUE, preemphasis of 3 dB per octave is applied to the sound before analysis.

Details

LPC coefficients are estimated using the autocorrelation method. The signal is windowed with a Hanning window prior to analysis.

Value

A vector containing the LPC coefficients is returned.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## make a synthetic vowel with a known set of formant frequencies
#sound = vowelsynth (ffs = c(500,1500,2500,3500,4500),
#fbw = c(30, 80, 150, 200, 220),f0 = 100, dur = 250)

## let the LPC function estimate the filter used to generate the vowel
#coeffs = lpc (sound, show = TRUE)
```

`makeFIR` *Create a Digital Filter*

Description

Design a Finite Impulse Response (FIR) Filter.

Usage

```
makeFIR (frequency, dB, order = 200, signal = NULL, window = 'hann',
         verify = FALSE, interpolation = 'linear')
```

Arguments

`frequency`	The frequencies at which the frequency response of the filter will be specified. The first frequency specified must be equal to 0. The final frequency specified is assumed to be equal to fs/2.
`dB`	The power (in decibels) of the filter at each specified frequency.
`order`	The order of the signal.
`signal`	If a signal is provided, it is filtered and returned.
`window`	The type of window to be applied to the filter impulse response. Uses the windowfunc() function included in this package.
`verify`	If TRUE, a series of plots are created to allow the user to verify that the filter is appropriate.
`interpolation`	Determines 'linear' or 'cubic' interpolation between the specified points. Uses the interpolate() function included in this package.

Details

Designs Type I FIR filters of odd length (even order). If an odd order is provided, 1 is added to the order. Filters are designed using the window-design method. The filter frequency response is defined at evenly-spaced locations determined by the filter order and the sampling frequency. If frequency specifications do not fall exactly on these points, the nearest appropriate location is used. This design method may lead to 'undesirable' behaviour between specified frequencies. This can be minimized by increasing the filter order and selecting an appropriate window function.

Value

If output = TRUE, the impulse response of the filter specified by the user is returned.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Lyons, R. G. (2004). Understanding Digital Signal Processing (2nd ed.). Prentice Hall.

Examples

```
## specify a filter with an arbitrary response
#frequency = c(0, 500, 502, 5000, 5002, 7000, 7002, 11025)
#dB = c(0, 0, -50,  -50, -10,  -10, -70, -70)

## create the filter and verify that the frequency response is as desired
#testfilter = makeFIR (frequency, dB, verify = TRUE, order = 1500)
#spectralslice (testfilter, padding = 1000)


## create the filter and verify that the frequency response is as desired
```

```
#makeFIR (frequency, dB, verify = TRUE, order = 300, signal = rnorm (1400))
```

`makesound` *Make a 'sound' object*

Description

Create a 'sound' object from a numeric vector.

Usage

```
makesound (sound, filename, fs = 22050)
```

Arguments

`sound`	A numeric vector representing a sound wave.
`filename`	A string indicating the desired file name associated with this object.
`fs`	The desired sampling frequency of the sound object.

Details

This function can make a vector into a 'sound' object. If a filename is not set, the filename defaults to 'sound.wav' where 'sound' indicates the name of the sound variable that was passed to the function. The benefit of working with 'sound' objects is that they carry their sampling frequency and filename (as well as some other information) with them.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

https://ccrma.stanford.edu/courses/422/projects/WaveFormat/

Examples

```
data (sound)
## take only the first 10000 samples from a 'sound' object
tmp = sound$sound[1:10000]
## and make a new 'sound' object
tmp = makesound (tmp, fs = 22050)
tmp

## get ready to make two plots with thin margins
#multiplot (2); par (mar = c(4,4,1,1));
## and show a spectrogram of the original
```

```
#spectrogram (sound)

## and the new, truncated version
#spectrogram (tmp)
```

---

multiplot *Plot with variable panel sizes*

Description

Create plots with columns or rows of unequal sizes.

Usage

```
multiplot (n = 2, type = 'r', sizes = rep (1/n, n), show = FALSE)
```

Arguments

`n`	The number of figure panels to be created.
`type`	If 'r', the panels will be arranged in rows, if 'c', the panels will be arranged in columns.
`sizes`	A vector indicating the percentage of the total width/height taken up by each row/column.
`show`	If TRUE, the resulting panel layout is shown to the user.

Details

This function is essentially a wrapper for the layout() function, which allows the user to create multi-panel figures in which each row or column is of a varying height/width.

Please note that small rows or columns might result in an error related to figure margins being too large when you try to create a plot. These may be reduced with the 'mar' parameter for the par() function, which sometimes solves the problem.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## uncomment and run

data (sound)
## run this instead to select your own sound for this demo.
# sound = loadsound ()

#par (mar = c(4,4,1,1))
```

```
#multiplot (n = 3, sizes = c(.25, .5, .25))

#plot (sound)
#spectrogram (sound, dynamicrange = 50, maxfreq = 7000)
#spectralslice (sound)
```

normalize *Normalize Vowels*

Description

Function to normalize vowels using one of several methods.

Usage

```
normalize(formants, speakers, vowels, method = 'neareyE',
          corners = NULL)
```

Arguments

`formants`	A matrix or dataframe containing formant frequency information for vowels. Each row is assumed to indicate data from a single vowel. At least two columns (indicating information regarding at least two formants) are required.
`speakers`	A vector indicating which speaker produced each vowel in 'formants'. The length of this vector must equal the number of rows in 'formants'.
`vowels`	A vector indicating the vowel category of each vowel in 'formants'. The length of this vector must equal the number of rows in 'formants'.
`method`	A string indicating the desired method. Choices are 'neareyE', 'neareyI', 'lobanov' and 'wandf'. See details for more information.
`corners`	For the 'wandf' method, a vector of two strings indicating the lowest F1-highest F2 vowel, and the highest F1-intermediate F2 vowel for the vowel system. In most vowel systems these are an /i/-like vowel, and an /a/-like vowel, respectively. Vowels must be provided in that order.

Details

This function normalizes vowels based on provided formant frequencies (FFs). The available methods are:

Nearey formant-extrinsic log-mean ('neareyE'): This method finds the logarithmic-mean FF across all vowels produced by a speaker, and subtracts this value from the log-transformed FFs representing each vowel.

Nearey formant-intrinsic log-mean ('neareyI'): This method finds the logarithmic-mean for each formant independently across all vowels produced by a speaker. The log-mean for each individual formant is then subtracted from the log-transformed FF representing each vowel.

Lobanov ('lobanov'): This method finds the mean and standard deviation for each formant. FFs are then standardized (in the statistical sense) using these estimated parameters for each speaker, for each formant.

Watt and Fabricius ('wandf'): This method requires the user to provided point vowels representing the frontmost and highest vowel, and the lowest (and, ideally central) vowel in a vowel system. An estimate of the centroid of the vowel system is calculated based on these values. Normalized FFs are then expressed as the ratio of observed FFs to the estimated centroids, independently for F1 and F2.

For both Nearey methods, and the Lobanov method, the average is found for each vowel category within-speaker before calculating the overall mean. As a result, the data from each speaker may contain unequal numbers of each vowel category. However, all speakers must be represented by the same vowel categories or the result will be (possibly) subtle differences in normalized vowel spaces dues to the possibly differing estimates of means and stadard deviations of the different formants.

Value

A dataframe with the same numbers of rows as the formant data provided and the following columns:

`formants`	A column corresponding to each formant provided. These are named 'fn' where n corresponds to the formant number.
`speakers`	A column indicated which speaker produced each vowel.
`vowels`	A column indicating which vowel is represented in each row.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Lobanov, B. M. (1971). Classification of Russian vowels spoken by different listeners. Journal of the Acoustical Society of America 49:606-08.

Nearey, T. M. (1978). Phonetic Feature Systems for Vowels. PhD thesis, Indiana University Linguistics Club.

Watt, D. and Fabricius, A. (2002). Evaluation of a technique for improving the mapping of multiple speakers' vowel spaces in the F1 ~ F2 plane. In D. Nelson, Leeds Working Papers in Linguistics and Phonetics 9:159-73.

Examples

```
## normalize all Peterson & Barney (1952) vowels using each method.
#data (pb52)

#neareyE = normalize (pb52[,7:9], pb52$speaker, pb52$vowel,
#method = 'neareyE')
#neareyI = normalize (pb52[,7:9], pb52$speaker, pb52$vowel,
#method = 'neareyI')
#lobanov = normalize (pb52[,7:9], pb52$speaker, pb52$vowel,
#method = 'lobanov')
```

```
#wandf = normalize (pb52[,7:9], pb52$speaker, pb52$vowel,
#method = 'wandf', corners =  c('i','A'))

## compare normalization methods using vowelplot().
#par (mfrow = c(2,2), mar = c(4,4,3,1))
#vowelplot (neareyE[,1], neareyE[,2], neareyE$vowel, alternateAxes = TRUE,
#  pointType = 16, main = 'neareyE', ellipses = TRUE)
#vowelplot (neareyI[,1], neareyI[,2], neareyI$vowel, alternateAxes = TRUE,
#  pointType = 16, main = 'neareyI', ellipses = TRUE)
#vowelplot (lobanov[,1], lobanov[,2], lobanov$vowel, alternateAxes = TRUE,
#  pointType = 16, main = 'lobanov', ellipses = TRUE)
#vowelplot (wandf[,1], wandf[,2], wandf$vowel, alternateAxes = TRUE,
#  pointType = 16, main = 'wandf', ellipses = TRUE)
```

`normalize.compare` *Compare Normalization Methods*

Description

Compare the effectiveness of different normalization methods.

Usage

```
normalize.compare (normd)
```

Arguments

`normd`	A list of dataframes containing the different formant data to be compared. Each dataframe must contain columns called "f1", "f2", "speaker", and "vowel", in any order.

Details

This function provides a relatively straightforward way to compare the effectiveness of different normalization methods based on the assumption that a good normalization method maximizes the separation between different vowel categories and minimizes the variation within a single vowel category. Minimizing the variation within a single vowel category means that the vowel spaces of different speakers are maximally similar.

This function provides two measures to compare the performance of normalization methods:

1) The square root of the average Mahalanobis distance between vowel categories is found for all pairs of vowel categories. This value indicates the average separation of vowel categories with respect to within-category error and the covariance patterns of the formant frequencies. A higher value indicates a better performing normalization algorithm.

2) The percentage of correct classifications using a linear discriminant model trained on the normalized formant-frequencies using the given category-labels.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## load the Peterson and Barney vowels
#data (pb52)

## normalize using several different methods
#neareyE = normalize (pb52[,7:9], pb52$speaker, pb52$vowel,
#method = 'neareyE')
#neareyI = normalize (pb52[,7:9], pb52$speaker, pb52$vowel,
#method = 'neareyI')
#lobanov = normalize (pb52[,7:9], pb52$speaker, pb52$vowel,
#method = 'lobanov')
#wandf = normalize (pb52[,7:9], pb52$speaker, pb52$vowel,
#method = 'wandf', corners =  c('i','A'))
#normd = list (pb52, neareyE, neareyI, lobanov, wandf)

## compare outcome of methods (and unnormalized vowels)
## uncomment to run
#normalize.compare (normd)
```

---

ntypes *Number of Unique Elements*

Description

Find the number of unique elements in a vector.

Usage

```
ntypes (vector)
```

Arguments

`vector`	The vector of interest.

Details

A simple function that converts a vector to a factor, and finds the number of levels. This provides the number of unique elements in a vector, something I find I frequently need.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
data (pb52)
## the number of unique vowel categories.
ntypes (pb52$vowel)
```

---

p73 *Pols et al. (1973) Dutch Vowel Data*

Description

Formant frequency information for vowels averaged across 24 male speakers.

Usage

```
data (p73)
```

Format

A data frame with the following columns:

`sex` - A factor indicating speaker sex.

`vowel` - The vowel category in x-sampa.

`f1` - A numeric vector indcating the vowel F1 in Hz.

`f2` - A numeric vector indcating the vowel F2 in Hz.

`f3` - A numeric vector indcating the vowel F3 in Hz.

References

Pols, L. C. W., Tromp, H. R. C., & Plomp, R. (1973). Frequency analysis of Dutch vowels from 50 male speakers. Journal of the Acoustical Society of America, 53. 1093-1101.

Examples

```
data(p73)
#vowelplot (p73$f1, p73$f2, p73$vowel, logaxes = 'xy', ellipses = TRUE,
#xsampa = TRUE)
```

pb52 *Peterson & Barney (1952) Vowel Data*

Description

Formant frequency and f0 information for vowels collected from 76 speakers in the Peterson & Barney (1952) data.

Usage

```
data (pb52)
```

Format

A data frame with 1520 observations on the following 9 variables:

`type` - A factor with levels `c m w` representing speaker type: child, man and woman.

`sex` - A factor with levels `f m` representing speaker gender: male and female.

`speaker` - A numeric vector indicating a uniquely identifying speaker number.

`vowel` - The vowel category in x-sampa.

`repetition` - A numeric vector indicating the repetition number.

`f0` - A numeric vector indcating the vowel f0 in Hz.

`f1` - A numeric vector indcating the vowel F1 in Hz.

`f2` - A numeric vector indcating the vowel F2 in Hz.

`f3` - A numeric vector indcating the vowel F3 in Hz.

Source

The data was created from tables provided within Praat:

http://www.fon.hum.uva.nl/praat/

References

Peterson, G.E. & Barney (1952). Control methods used in a study of the vowels. Journal of the Acoustical Society of America 24: 175-184.

Boersma, Paul & Weenink, David (2012). Praat: doing phonetics by computer [Computer program]. Version 5.3.19, retrieved 24 June 2012 from http://www.praat.org/

Examples

```
data(pb52)
#vowelplot (pb52$f1, pb52$f2, pb52$vowel, logaxes = 'xy', ellipses = TRUE,
#xsampa = TRUE)
```

`peakfind` *Find the Peaks*

Description

Locate the peaks in a numeric vector.

Usage

```
peakfind (x, show = TRUE)
```

Arguments

`x`	A vector whose peaks are to be located.
`show`	If TRUE, the vector is plotted and peaks are indicated with red triangles.

Details

This function looks for peaks by finding elements whose value is greater than both the elements that surround it. If no peaks are found, a value of zero is returned.

Value

A vector indicating the location (position in the vector) of peaks in the vector.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
#data (sound)
#sound2 = sound$sound[10000:11000]
#spectrum = spectralslice (sound2, padding = 0, output = TRUE, show = TRUE)
#peakfind (spectrum[,2])
```

`phasor` *Plot Phasors*

Description

Plot phasors representing one or more complex-valued variables.

Usage

```
phasor (num, scaled = TRUE, add = FALSE, circle = FALSE,
        xlim, ylim, ...)
```

Arguments

`num`	A complex-valued variable to be plotted as a phasor, or a vector of such variables.
`scaled`	If TRUE, phasor magnitudes are scaled to 1.
`add`	If TRUE, phasors are plotted on existing figure. If FALSE, a new plot is created.
`circle`	If TRUE and scaled is TRUE, the unit circle is drawn.
`xlim`	x-axis range.
`ylim`	y-axis range.
`...`	Additional arguments are passed to the internal call of 'plot' and 'arrows'.

Details

Complex-valued variables may be created using the complex() function.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
#noise = rnorm (100)
#phasors = fft(noise)

#par (mfrow = c(1,3))
#phasor (phasors)
#phasor (phasors, circle = TRUE)
#phasor (phasors, scaled = FALSE)
```

pickIPA *Pick IPA Symbols*

Description

Select IPA symbols that you wish to include in a plot.

Usage

```
pickIPA (vowels, n = 0, xsampa = FALSE, description = FALSE,
         verify = TRUE)
```

Arguments

`vowels`	An optional vector of vowel labels that you would like to plot using IPA symbols.
`n`	If no vowel vector is provided, the number of symbols desired.
`xsampa`	If TRUE, x-sampa versions of the symbols are returned.
`description`	If TRUE, description of the place, manner and voicing of the symbols are also returned.
`verify`	If TRUE, the selected symbols are plotted in the order indicated by the user, allowing visual confirmation of the selected symbols.

Details

This is an interactive function that allows the user to select IPA symbols for plotting using a chart. The values returned by this function may only be passed to the 'pch' parameter within plotting functions. At the moment it has only been implemented for vowel sounds.

If a vowels vector is given, the function finds the number of categories in the vector. The user is then prompted to select the IPA symbol corresponding to each category by clicking on the correct location on the plot.

If a vowels vector is not provided, the function allows the user to select any desired number of symbols, and returns these in the same order as indicated by the user.

There may be issues when exporting figures to PDF using IPA font. Exporting plots directly as images works 'out of the box'

Value

A list with the following columns (some of which are optional):

`IPA`	The symbol which should be passed to 'pch' to plot IPA symbols.
`xsampa`	The xsampa representation of each IPA character.
`description`	If description = TRUE, a description of each sound.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/X-SAMPA

Examples

```
##uncomment to run
#vowels = pickIPA (n = 3)
#plot (c(1,2,3), c(1,2,3), pch = vowels)

## select vowels in the order displayed in the console
## to obtain symbols and descriptions of the vowel categories
## in the Peterson and Barney data.
```

```
# data (pb52)
# tmp = pickIPA (pb52$vowel, description = TRUE, xsampa = TRUE)
# tmp
```

pitchtrack *Pitch Tracking*

Description

Create a pitch track for a sound.

Usage

```
pitchtrack (sound, f0range = c(60,400), timestep = 2, fs = 22050, minacf = .5,
correction = TRUE, show = TRUE, windowlength = 50, addtospect = FALSE)
```

Arguments

`sound`	A numeric vector representing a waveform, or a 'sound' object created with the loadsound() or makesound() functions.
`f0range`	A numeric vector of length two where the first value corresponds to the minimum f0 Hz to be considered, and the second represents the maximum to be considered.
`timestep`	How far the analysis window will advance, in milliseconds. If this value is set to zero, the function is applied to the entire signal.
`fs`	The sampling frequency of the sound. If a 'sound' or 'ts' object is passed, this does not need to be specified.
`minacf`	Autocorrelation values below this are ignored.
`correction`	If TRUE, ACF values are corrected for lag value.
`show`	If TRUE, a plot displaying the pitch track is created.
`windowlength`	The length of the analysis window, in milliseconds. This should be approximately three-times longer than the wavelength of the lowest pitch. The default value is appropriate for a floor of 60 Hz.
`addtospect`	If TRUE, the pitch track is added to a spectrogram created with the spectrogram() function included in this package. The track is scaled up by a factor of 10 (e.g., 100 Hz will be plotted at 1000 Hz on the spectrogram) so that it will fit nicely in the typical spectrogram range of 0-5000 Hz.

Details

Pitch tracking is carried out using a simplified version of the algorithm described in Boersma (1993), including corrections for lag value and window function. When plotting pitch tracks, the points sizes are proportional to autocorrelation values.

Value

A dataframe with the following columns:

`time`	The analysis window centre point, in milliseconds.
`f0`	The calculated f0 (pitch).
`acf`	The value of the autocorrelation function corresponding to the winning f0.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Boersma, P., (1993). Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. Proc. Instit. Phon. Sci. 17: 97-110.

Examples

```
#data (sound)               ## use the example 'sound' object provided
#sound = loadsound()       ## or run this line to use you own sound

## to generate a pitch track
#output = pitchtrack (sound)

## to add a pitch to a spectrogram
#spectrogram (sound)
#pitchtrack (sound, addtospect = TRUE)
```

`playsound` *Play Sounds*

Description

Play sounds in R using VLC Player.

Usage

```
playsound  (sound, path = 'default', fs = 10000, erase = TRUE)
```

Arguments

`sound`	Either a sound object, or a numeric vector to be played.
`path`	The location of VLC.exe on your computer. 'default' provides the default for a standard windows installation of VLC player. If this is set to 'pick', a dialog box opens allowing the user to specify the path.
`fs`	The sampling frequency in Hz. If a 'sound' or 'ts' object is passed this does not need to be specified.
`erase`	If TRUE, the temporary WAV file created is deleted after playing.

Details

The sound is written out as a .wav file and VLC is called from the command line to play the file. The file is optionally erased after the fact. If the path is selected by the user, this is returned for use in future calls. Obviously, this function relies on VLC being installed. With the appropriate path, it seems like this function should work on Linux and OSX, though it has only been tested on a Windows computer.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
#  ## make a 1 second, 100 Hz tone.
#  tone = sinusoid (freqs = 250, dur = 1000, fs = 1000)[,2]
#  ## play it in VLC player
#  playsound (q[,2], fs = 1000, erase = FALSE)
```

`polezero` *Pole-zero Plots*

Description

Generate a Pole-zero plot from filter coefficients.

Usage

```
polezero (b, a, ...)
```

Arguments

`b`	The filter moving-average (MA) coefficients.
`a`	The filter autoregressive (AR) coefficients.
`...`	Additional parameters are passed to the internal call of plot().

Details

This function plots filter poles (x) and zeros (o) based on the given coefficients.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/Pole

Examples

```
## example of a typical single-zero preemphasis filter
#a = 1
#b = c(1, -.94)
#polezero (b, a)

#example of a complex-pole formant-style filter
#a = c(1, -.3, .2)
#b = c(1)
#polezero (b, a)
```

`powertrack` *Power tracking*

Description

Create a power track for a sound.

Usage

```
powertrack (sound, timestep = 5, windowlength = 30,
            fs = 22050, show = TRUE, zeromax = TRUE, ...)
```

Arguments

`sound`	A numeric vector representing a waveform, or a 'sound' object created with the loadsound() or makesound() functions.
`timestep`	Determines how far the window will be moved for each adjacent analysis, in milliseconds.
`windowlength`	Determines how much of the signal is included for each analysis step, in milliseconds. If this is too small, pitch-synchronous ripples will be seen in the track.
`fs`	The sampling frequency of the sound. If a 'sound' or 'ts' object is passed, this does not need to be specified.
`show`	If TRUE, the track is plotted.

`zeromax`	If TRUE, the maximum dB value is set to zero.
`...`	Additional parameters are passed to the internal call of plot(), and used to create the figure.

Details

Returns the average power in the signal in a section as determined by the parameters of the function. A Hann window is applied to each section prior to analysis. Sections with zero power are ignored and not returned.

Value

A dataframe with the following columns:

`time`	The time, in milliseconds, of each point of analysis.
`power`	The power, in decibels, at each analysis point.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## plot the waveform and power of a given sound.
#data (sound)
# sound = loadsound()        ## run this line to use you own sound

## compare waveform and power
#par (mfrow = c(2,1), mar = c(4,4,1,1));
#plot (sound)
#powertrack (sound)
```

`preemphasis` *Add Preemphasis*

Description

Single-zero preemphasis filter.

Usage

```
preemphasis (input, cutoff = 50, fs = 22050, verify = FALSE,
coeff = 0)
```

Arguments

`input`	Either a numeric vector representing a sequence of samples taken from a sound wave or a sound object created with the loadsound() or makesound() functions.
`cutoff`	The spectral slope is increased by 6 dB. per octave above this frequency.
`fs`	The sampling frequency of the sound. If a 'sound' object is passed, this does not need to be specified.
`verify`	If TRUE, before and after spectra are plotted to allow the user to visually verify the process.
`coeff`	Optionally, the single coefficient used by the filter may be specified.

Value

The modified sound is returned. If a 'sound' object another 'sound' object is returned.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://www.fon.hum.uva.nl/praat/manual/Sound__Filter__pre-emphasis____.html

Examples

```
#signal = sinusoid (c(100, 200, 400, 800, 1600), fs = 4000,
#dur = 100, sum = TRUE)[,7]

#preemphasis (signal, verify = TRUE, fs = 4000, cutoff = 50)
```

`PSTM` *Probabilistic Sliding Template Model*

Description

Classify Vowels Using the Probabilistic Sliding Template Model

Usage

```
PSTM (ffs, f0, template, winner = TRUE)
```

Arguments

`ffs`	A matrix of formant frequencies in which each row represents a single vowel token, and each column represents a formant frequency. At least 3 formants need to be specified for every vowel.
`f0`	A vector containing the average f0 measured for each vowel to be classified. The length of this vector must equal the number of rows in ffs.
`template`	A 'template' object created with the createtemplate() function provided in this package. If no template is specified, vowels are classified relative to vowels from Edmonton English.
`winner`	If TRUE, only the winner of each classification is returned. If FALSE, information regarding all candidate vowels is returned. See 'value' subsection for details.

Details

The classic log-mean normalization method of Nearey (1978) helps compare the vowels produced by different speakers by controlling for the log-mean formant frequency (FF) produced by a speaker. This approach to normalization assumes that variation in the vowel spaces of speakers of the same dialect is primarily according to a single multiplicative parameter. When this speaker-specific scaling parameter is controlled for, differences in the vowel spaces of different speakers are minimized.

The Probabilistic Sliding Template Model (PSTM) of Nearey and Assmann (2007) attempts to predict perceived vowel quality by 'guessing' an appropriate speaker-specific scaling parameter and normalizing vowels using this estimated parameter. 'Method 6' of the PSTM (described in Nearey & Assmann, 2007) is used to estimate the necessary parameter. After normalization, vowels are classified by comparing them to a provided reference template, which can be created using the createtemplate() function included in this package. Normalized or unnormalized vowels may be classified, as long as the same transformations are performed on the data used to create the template and the data being classified.

If no template is passed, the model identifies vowels relative to the vowel system of Edmonton English speakers. For in-depth details regarding the specifics of this model, please see Nearey & Assmann (2007).

Value

If winner = TRUE:

A dataframe with the following columns:

`vowel`	The label for each winning vowel.
`psi`	The optimal psi determined for the winning vowel.
`postprob`	The posterior probability of observing the winning vowel, given the formants and the psi.

If winner = FALSE:

A list of dataframes, each of which contains information for every candidate vowel category for each token to be classified. Each dataframe has the following columns:

`vowel`	The label for each candidate vowel category.
`psi`	The optimal psi determined for each vowel category.
`postprob`	The posterior probability of observing each vowel category, given the formants and the psi.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Nearey, T. M. (1978). Phonetic Feature Systems for Vowels. PhD thesis, Indiana University Linguistics Club.

Nearey, T. M. & P. F. Assmann. (2007). Pobabilistic 'sliding template' models for indirect vowel normalization. in Experimental Approaches to Phonology, edited by M.J. Sole, P. S., Beddor, and M. Ohala (Oxford University Press, Oxford), pp. 246-269.

Examples

```
## load Peterson & Barney vowel data
#data (pb52)

## normalize vowel formant frequencies
#normdvowels = normalize (pb52[,7:9], pb52$speaker, pb52$vowel)
#formants = normdvowels[,1:3]
#vowels = pb52$vowel

## make a vowel template based on these frequencies
#template = createtemplate (formants, vowels)

## first classify only the first three vowels
#ffs = pb52[1:3,c(7:9)]
#f0 = pb52[1:3,6]

## outputting only the winners, and then the full posterior probabilities
#PSTM (ffs, f0, template)
#PSTM (ffs, f0, template, winner = FALSE)

## now classify all vowels
## uncomment to run
#ffs = pb52[,c(7:9)]
#f0 = pb52[,6]
#winner = PSTM (ffs, f0, template)
## with a good degree of accuracy
#table (winner$vowel, pb52$vowel)
```

pwelch *Welch's Power Spectral Density Estimate*

Description

Calculates a power spectral density estimate using Welch's method.

Usage

```
pwelch (sound, points = 0, overlap = 0, padding = 0,
window = 'hamming', show = TRUE, fs = 1, preemphasisf = 0,
zeromax = TRUE, type,...)
```

Arguments

`sound`	A vector representing a sound wave, or a 'sound' object.
`points`	The number of points to be included in each window. If not specified, the sound is divided into 10 equally-sized windows.
`overlap`	The amount of overlap between adjacent segments, in points.
`padding`	The amount of zero padding to be applied to each window.
`window`	The window to be applied to each segment of the signal. See windowfunc() for available options.
`show`	If FALSE, no plot is created.
`fs`	The sampling frequency of the sound. If a 'sound' object is passed this does not need to be specified.
`preemphasisf`	Preemphasis of 3 dB per octave is added to frequencies above the specified frequency.
`zeromax`	If TRUE, the maximum dB is set to 0.
`type`	The line type to be used for plotting, passes its value to the 'lty' parameter.
`...`	any additional arguments will be passed to the internal calls of 'plot'.

Details

This function divides the signal into a number of equally-sized windows, finds the power spectrum of each one, and then finds the average across all windowed sections.

Value

The following columns:

`hz`	The center frequency of the analysis bins.
`dB`	The dB magnitude/power for the analysis bin.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## make a sine wave
#sinewave = sinusoid (f = 300, fs = 1000, dur = 2000, sum = FALSE)

## add noise
#sinewave = sinewave[,2] + rnorm (length (sinewave[,2]), 0, 3)

## compare the results of pwelch() and spectralslice()
#par (mfrow = c(2,1), mar = c(4,4,1,1))
#spectralslice (sinewave, fs = 1000)
#pwelch (sinewave, points = 400, fs = 1000)
```

`rcr` *Random Coefficients Regression*

Description

Carry out a random coefficients regression (rcr) using repeated calls to glm, individually for the data from each participant/data cluster.

Usage

```
rcr(formula, participants, dataframe, ...)
```

Arguments

`formula`	A symbolic description of the model to be fitted.
`participants`	A vector indicating which row in the dataframe belongs to which participant. Length must equal the number of rows in the dataframe.
`dataframe`	The dataframe containing the data for the model.
`...`	Additional arguments to be passed to the internal glm() function call. For example, family, if not 'gaussian', should be specified.

Details

This function fits a model to the data from each participant individually using repeated calls to glm(). Significance testing is then carried out on the coefficients fit for each participant using the methods established in Gumpertz & Pantula (1989) and Lorch & Myers (1990).

In perceptual experiments there is frequently a high number of data points collected from each participant, and the data collected from each participant is balanced by design. In these situations rcr performs comparably to mixed-effects models. In the event that only a small number of observations are made from each listener, or the data is not balanced, rcr may not be appropriate.

A call to summary() on an rcr object performs a one-sample t-test on each coefficient to test whether it is significantly different from zero.

A call to anova() on an rcr object performs a one-sample t-test in the case of single coefficients, and a one-sample Hotelling T2 test in the event that multiple coefficients are associated with a single factor, to test that they are not all equal to zero.

A call to plot() on an rcr object displays the density corresponding to the distribution of all fitted coefficients. These are compared to a normal distribution with the same mean and standard deviation.

Value

An object of class 'rcr', a list containing the elements:

`formula`	The formula used to call and create the rcr object.
`call`	The exact call used to create the rcr object.
`participants`	A vector indicating the labels used to identify each individual participant as indicated by 'participants'.
`factors`	A vector indicating the grouping of the explanatory variables.
`factor.names`	A vector containing the names of each group of coefficients.
`coefficients`	A dataframe containing the coefficients fit individually for each participant.
`coefficient.means`	The mean value of each coefficient across all participants.
`coefficient.names`	The name of each individual coefficient.
`varExp`	The percent of variance or deviance explained by the model for each participant.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Gumpertz, M., & Pantula, S. G. (1989). A Simple Approach to Inference in Random Coefficient Models. The American Statistician, 43(4), 203-210.

Lorch, T. F. & Myers, J. L. (1990). Regression analyses of repeated measures data in cognitive research. J. Exp. Psychol. Learn. Mem. Cogn. 16: 149-157.

Examples

```
#data (pb52)

## runs an rcr model on the Peterson & Barney (1952) vowels to test
## for the predictive value of the speaker's f0 and F3
#rcr.model = rcr (f1 ~ f0 * f3, pb52$speaker, pb52)
```

```
#rcr.model

## test for the significance of each individual coefficient
#summary (rcr.model)

## a similar analysis can be run using vowel category as the predictor
#rcr.model = rcr (f1 ~ vowel, pb52$speaker, pb52)
#rcr.model

## here, summary() tests each coefficient individually
#summary (rcr.model)

## while anova() tests associated coefficients together
#anova (rcr.model)
```

`reduce.fraction` *Reduce Fractions*

Description

Reduce fractions to lowest terms using Euclid's Algorithm.

Usage

```
reduce.fraction (ratio)
```

Arguments

`ratio`	A vector with two integers. The first element is the numerator of a ratio, and the second element is the denominator.

Value

A vector containing the elements of the reduced fraction. The first element is the numerator of a ratio, and the second element is the denominator.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## an easy one
reduce.fraction (c(100, 200))

## irreducible
reduce.fraction (c(140, 201))

## a hard one
reduce.fraction (c(140, 203))
```

`resample` *Resample a Sound*

Description

Resample using sinc interpolation.

Usage

```
resample (sound, newfs, oldfs, precision = 50, filterorder = 200, synthfilter = FALSE)
```

Arguments

`sound`	Either a numeric vector representing a sequence of samples taken from a sound wave or a sound object created with the loadsound() or makesound() functions.
`newfs`	The new desired sampling frequency.
`oldfs`	The original sampling frequency. If a 'sound' object is provided, this does not need to be specified.
`precision`	The number of samples before and after the current point to be used for interpolation.
`filterorder`	The number of taps to be used for the low-pass FIR filters used, where appropriate.
`synthfilter`	If TRUE, synthfilter() is used for filtering.

Value

The resampled vector is returned. If a 'sound' object is passed, the resampled sound is returned as an object.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
#data (sound)
## downsample and then upsample the sound back to
## its original sampling frequency
#downsamped = resample (sound, 11025)
#upsamped = resample (downsamped, 22050)

# compare a part of the waveforms for all three sounds
#par (mfrow = c(3,1), mar = c(4,4,1,1))
#plot (sound$sound[1:14000], type = 'l')
#plot (upsamped$sound[1:14000], type = 'l', col = 2)
#plot (downsamped$sound[1:7000], type = 'l', col = 4)
```

rmvtnorm *Random multivariate normal variables*

Description

Draw vectors from a multivariate normal distribution.

Usage

```
rmvtnorm (n = 1, k = 2, means = rep (0, k), sigma = diag (k))
```

Arguments

n	The number of vectors to be drawn.
k	The dimension of the vectors to be drawn.
means	A vector of means, one for each dimension.
sigma	The covariance matrix of the distribution. If a number between 0 and 1 is provided, this is assumed to be the correlation parameter for a bivariate standard normal distribution.

Details

If means and sigma are not specified, a standard normal distribution is assumed along every dimensions, and dimensions are assumed to be uncorrelated. If the number of dimensions is not specified, a bivariate distribution is assumed.

Value

A matrix with rows equal to n and columns equal to k, where each row indicates a single observation, and each column represents a different dimension.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## Examples of draws from different bivariate normal distributions
## and standard deviation ellipses drawn to fit them.
#par (mfrow = c(2,2))
#draws = rmvtnorm (n = 1000, k = 2, sigma = .3)
#plot (draws)
#sdellipse (draws, stdev = 3, lwd = 3, col = 2)

#draws = rmvtnorm (n = 1000, k = 2, sigma = -.3)
#plot (draws)
#sdellipse (draws, stdev = 3, lwd = 3, col = 2)
```

```
#draws = rmvtnorm (n = 1000, k = 2, sigma = -.7)
#plot (draws)
#sdellipse (draws, stdev = 3, lwd = 3, col = 2)

#draws = rmvtnorm (n = 1000, k = 2, sigma = .7)
#plot (draws)
#sdellipse (draws, stdev = 3, lwd = 3, col = 2)
```

`rotate` *Rotate*

Description

Rotate 2D and complex-valued observations. The output is of the same kind as the input.

Usage

```
rotate (xy, angle, degrees = FALSE, origin = TRUE)
```

Arguments

`xy`	Either, a vector of complex-valued observations, or a matrix with 2 columns and any number of rows.
`angle`	The desired angle of rotation, in radians.
`degrees`	If angle of rotation is specified in degrees instead of radians, this should be set to TRUE.
`origin`	If TRUE, points are rotated about the origin. If FALSE, points are rotated 'in place' about the mean for each dimension (i.e. the central location of the distribution).

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## rotate points in a 2D space
#mat = cbind (1:100, 100:1)
#rotate (mat, pi/2)

## rotate complex-valued numbers
#complx = complex (real = mat[,1], imaginary = mat[,2])
#rotate (complx, pi/2)
```

`sdellipse` *Standard deviation Ellipse*

Description

Draw standard deviation ellipses around a group of observations.

Usage

```
sdellipse (points, stdev = 1.96, density = .01,
add = TRUE, show = TRUE, means = NULL, se = FALSE, ...)
```

Arguments

`points`	A matrix with two columns in which each row is a different observation from a bivariate normal distribution. Optionally, a 2 by 2 covariance matrix may be specified directly in conjunction with the means parameter.
`stdev`	The number of standard deviations to be enclosed by the ellipse.
`density`	The spacing between sampling points along the ellipse. A higher number results in a coarser sampling.
`add`	If TRUE, the ellipse is added to an existing plot. If FALSE, a new plot is created.
`show`	If FALSE, no ellipse is drawn.
`means`	A vector of 2 means, one for each dimension. If these are specified, points is assumed to be a covariance matrix rather than a sequence of observations.
`se`	If TRUE, a standard error (rather than standard deviation) ellipse is plotted.
`...`	Additional arguments are passed to the internal call of lines() or plot() as appropriate.

Details

This function may be used in 2 different ways: 1) To draw standard deviation ellipses around a set of observations (if 'means' is not specified) 2) To draw ellipses and circles on plots (if 'means' is specified).

Value

A matrix is returned where the first column indicate x-axis ellipse coordinates and the second column indicates y-axis ellipse coordinates.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## Examples of draws from different bivariate normal distributions
## and standard deviation ellipses drawn to fit them.
#par (mfrow = c(2,2))
#draws = rmvtnorm (n = 1000, k = 2, sigma = .3)
#plot (draws)
#sdellipse (draws, stdev = 3, lwd = 3, col = 2)

#draws = rmvtnorm (n = 1000, k = 2, sigma = -.3)
#plot (draws)
#sdellipse (draws, stdev = 3, lwd = 3, col = 2)

#draws = rmvtnorm (n = 1000, k = 2, sigma = -.7)
#plot (draws)
#sdellipse (draws, stdev = 3, lwd = 3, col = 2)

#draws = rmvtnorm (n = 1000, k = 2, sigma = .7)
#plot (draws)
#sdellipse (draws, stdev = 3, lwd = 3, col = 2)

## alternatively, a covariance matrix may be specified directly.
#par (mfrow = c(1,1))
#sdellipse (matrix(c(1,.5,.5,1),2,2), means = c(0,0),
#add = FALSE, stdev = 1)
#sdellipse (matrix(c(1,-.5,-.5,1),2,2), means = c(0,0),
#add = TRUE, stdev = 1)
```

`selectslice` *Select Slices*

Description

Select spectral slices from a spectrogram.

Usage

```
selectslice (specobject, n = 1, plot = TRUE, ...)
```

Arguments

`specobject`	A 'spectrogram' object created with the spectrogram() function included in this package.
`n`	The number of desired slices.
`plot`	If FALSE, the spectrogram is not plotted.
`...`	Additional arguments are passed to the internal call of plot.spectrogram().

Details

This function allows the user to select a given number of time points on a spectrogram, and to retrieve spectral slices at those time points. The spectrogram object is plotted, and the user must click on the spectrogram a given number of times at the desired points. The analysis parameters of the slices will reflect the parameters used when creating the spectrogram. Setting the 'plot' parameter to FALSE is useful if you want one set of analysis parameters for the visually presented spectrogram and another set for the spectral slices.

Value

A dataframe with one column for each spectral slice selected. Row names indicate frequencies, column names indicate times in milliseconds. Values are decibels below peak.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## uncomment to run

# sound = vowelsynth ()
# spect = spectrogram (sound)
# slices = selectslice (spect, n = 3)
```

`sinc` *Sinc Function*

Description

Sample the sinc function at given points

Usage

```
sinc (x, normalized = FALSE)
```

Arguments

`x`	A vector of x-axis points at which the sinc function will be sampled.
`normalized`	If TRUE, a normalized sinc function is returned.

Details

The formula for the unnormalized sinc function is y = sin(x)/x.

The formula for the normalized sinc function is y = sin(x*pi)/(x*pi).

Value

A vector containing y-axis values for each specified x-axis value is returned.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/Sinc_function

Lyons, R. G. (2004). Understanding Digital Signal Processing (2nd ed.). Prentice Hall.

Examples

```
#x = seq(-20,20,.1)
## generate both forms of the sinc function between -20 and 20
#y1 = sinc (x)
#y2 = sinc (x, normalized = TRUE)
## the unnormalized sinc function has zero crossing at
## integer multiples of pi
#plot (x,y1, type = 'b')
## the normalized sinc function has zero crossing at integers
#lines (x,y2, type = 'b', col = 4)
#abline (h = 0)
```

`sinusoid` *Create Sinusoids*

Description

Create and display one or more sinusoids.

Usage

```
sinusoid (freqs, amps = rep(1, length(freqs)), dur = 50,
phases = rep(0, length(freqs)), fs = 10000, sum = FALSE,
show = FALSE, colors = NULL)
```

Arguments

`freqs`	A vector of frequencies, one for each desired sinusoid.
`amps`	A vector of peak amplitudes, one for each desired sinusoid.
`dur`	The desired duration of the sinusoids, in milliseconds.
`phases`	A vector of initial phases, one for each desired sinusoid, expressed in radians.
`fs`	The desired sampling frequency of the sinusoids.
`sum`	If TRUE, the sum the generated sinusoids is also found.

`show`	If TRUE, the generated sinusoids are plotted. If sum is TRUE, a second plot is created to display the sum of the sinusoids.
`colors`	An optional vector of colors used to plot the individual sinusoids. If the number of colors given is less than the number of frequencies specified, the colors are repeated.

Details

A number of sinusoids are generated, and optionally plotted and/or returned. The number of frequencies specified must equal the number of amplitudes and initial phases.

Value

A dataframe with the following columns:

`time`	The time, in milliseconds, at which is sample is taken.
`waveN`	A series of columns, each indicating the amplitude of wave N at a given time.
`sum`	A column indicating the sum of all of the sinusoids.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## several waves, and the sum
#sum = sinusoid (freqs = c(100,200,300), amps = c(1,3,2),
#sum = TRUE, show = TRUE)

## no sum, different phase shifts
#nosum = sinusoid (freqs = c(100,200,300), amps = c(1,3,2),
#phases = c(pi/2, 0, pi/4), sum = FALSE)
```

`snip` *Snip/Zoom*

Description

Select a subsection of a sound or spectrogram object.

Usage

```
snip  (object, show = TRUE)
```

Arguments

`object`	A 'sound' or 'spectrogram' object to be truncated.
`show`	If TRUE, the selected subsection is displayed, resulting in zoom functionality.

Details

The input object is plotted and the user must click on two points on the plot. If show = TRUE, this function allows the user to zoom in on a subsection of the object. If the output is assigned to a variable, a new object is created and returned that contains only the data in between the two designated points.

Value

A 'sound' or 'spectrogram' object which interacts with several functions included in this package.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
#data (sound)
## the example below is commented because examples cannot
## require user interaction. simply uncomment the lines below
## and select two points on the plot.
# clipped = snipsound (sound)
# spectrogram (clipped)
# spectralslice (clipped)
```

`sound` *Sound object*

Description

An example of a sound object, the phrase 'This is a spectrogram', produced by the author of this package. This sound object may be inspected with spectrogram() and plotted with plot(). Sections of the individual samples representing the waveform (found in the "sound" element of a sound object) can be passed to spectralslice() to see a spectral slice. Several functions included in this package may also be used to modify or manipulate 'sound' objects.

Usage

```
data (sound)
```

Format

This sound object has the same properties as all sound objects. These may be inspected by using the print() function.

Examples

```
## uncomment and run

#data (sound)
#par (mar = c(4,4,1,1))
#multiplot (n = 3, sizes = c(.25, .5, .25))

#plot (sound)
#spectrogram (sound, dynamicrange = 50, maxfreq = 7000)
```

`spectralslice` *Spectral Slice*

Description

A function to plot the power spectrum of a vector representing a sound wave.

Usage

```
spectralslice (sound, padding = length(sound) * 2, fs = 1,
show = TRUE, add = FALSE, window = "kaiser",
windowparameter = 3, zeromax = TRUE, preemphasisf = 0, type,
line = FALSE, removeDC = TRUE, ...)
```

Arguments

`sound`	A vector representing a sound wave, or a 'sound' object.
`padding`	The amount of zero-padding desired for the analysis, in number of samples.
`fs`	The sampling frequency of the sound. If a 'sound' object is passed this does not need to be specified.
`show`	If FALSE, no plot is created.
`add`	If TRUE, the spectrum is added to an existing plot. If FALSE a new one is created.
`window`	The window to be applied to the signal, applied by windowfunc(), provided in this package.
`windowparameter`	The parameter for the window to be applied to the signal, if appropriate. Passed to windowfunc().
`zeromax`	If TRUE, the maximum dB is set to 0.
`type`	The line type to be used for plotting, passes its value to the 'lty' parameter.
`preemphasisf`	Preemphasis of 3 dB per octave is added to frequencies above the specified frequency.
`line`	If TRUE, a line spectrum is created.
`removeDC`	If TRUE, the DC component is removed.
`...`	any additional arguments will be passed to the internal calls of 'plot' or 'lines'.

Value

A dataframe with the following elements is returned:

hz	The center frequency of the analysis bins.
dB	The dB magnitude/power for the analysis bin.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## synthesize schwa-like vowel
#vowel = vowelsynth (ffs = c(500,1500,2500,3500,4500))$sound

## compare window lengths
#par (mfrow = c(3,1))
#spectralslice (vowel[500:550], fs = 10000)
#spectralslice (vowel[500:1000], fs = 10000)

## line spectrum
#spectralslice (vowel[500:600], padding = 0, line = TRUE, fs = 10000)
```

---

spectrogram *Create Spectrograms*

Description

Create and display spectrograms.

Usage

```
spectrogram (sound, fs = 22050, windowlength = 5,
timestep = -1000, padding = 10,
preemphasisf = 50, maxfreq = 5000, colors = TRUE,
dynamicrange = 50, nlevels = dynamicrange, maintitle = "",
show = TRUE, window = 'kaiser', windowparameter = 3,
quality = FALSE)
```

Arguments

sound	Either a numeric vector representing a sequence of samples taken from a sound wave or a sound object created with the loadsound() or makesound() functions.
fs	The sampling frequency in Hz. If a sound object is passed this does not need to be specified.

`windowlength`	The desired analysis window length in milliseconds.
`timestep`	If a negative value is given, -N, then N equally-spaced time steps are calculated. If a positive number is given, this is the spacing between adjacent analyses, in milliseconds.
`padding`	The amount of zero padding for each window, measured in units of window length. For example, if the window is 50 points, and padding = 10, 500 zeros will be appended to each window.
`preemphasisf`	Preemphasis of 6 dB per octave is added to frequencies above the specified frequency. For no preemphasis, set to a frequency higher than the sampling frequency.
`maxfreq`	the maximum frequency to be displayed for the spectrogram up to a maximum of fs/2. This is set to 5000 Hz by default.
`colors`	If TRUE, a color spectrogram will be displayed. If FALSE, greyscale is used. If a vector of colors is provided, these colors are used to create the spectrogram.
`nlevels`	The number of divisions to be used for the z-axis of the spectrogram. By default it is set equal to the dynamic range, meaning that a single color represents 1 dB on the z-axis.
`dynamicrange`	Values greater than this many dB below the maximum will be displayed in the same color.
`maintitle`	A string indicating the spectrogram title if one is desired.
`show`	If FALSE, no spectrogram is plotted. This is useful if the user would like to perform an action on an existing spectrogram plot without having to redraw it.
`window`	the window to be applied to the signal, applied by the windowfunc function in this package.
`windowparameter`	the parameter for the window to be applied to the signal, if appropriate.
`quality`	If TRUE, a contour plot is created, which results in a high-quality image that may be slow to plot. If FALSE, a lower-quality image is created that plots much faster.

Details

This function is used to create and plot spectrograms. The user may specify all analysis parameters, in addition to the colors used to display the spectrogram.

The function optionally returns a spectrogram object for which the plot() function is defined.

Value

If output is set to TRUE, an object of class 'spectrogram', a list containing the elements:

`spectrogram`	a matrix containing the magnitude at each bin center. Frequencies differ across columns, while time varies between rows.
`fs`	the sampling frequency of the sound from which the spectrogram was made.
`windowlength`	the length of the analysis window used to create the spectrogram.

`timestep`	the timestep (in milliseconds) used to create the spectrogram.
`dynamicrange`	the dynamic range (in dB) of the spectrogram.
`color`	the colors used to create the spectrogram. This value corresponds to the 'color' parameter set when calling spectrogram().
`maxfreq`	the maximum desired frequency when plotting.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## uncoment and run.
#data (sound)             ## use the example 'sound' object provided
#sound = loadsound()       ## or run this line to use you own sound

#spectrogram (sound)
# spectrogram (sound, quality = TRUE)
```

---

synthfilter *Filtering by Synthesis*

Description

Use Fourier synthesis to recreate signal without undesired frequency components.

Usage

```
synthfilter (sound, band = c(0,fs/4), fs = 1, verify = FALSE, attenuation = 0)
```

Arguments

`sound`	A numeric vector representing a waveform, or a 'sound' object created with the loadsound() or makesound() functions.
`band`	A vector with exactly two elements, the first specifying the lowest frequency to be passed (must be > 0), and the second specifying the highest frequency to be passed (must be < fs/2).
`fs`	The sampling frequency of the sound. If a 'sound' object is passed, this does not need to be specified.
`attenuation`	Attenuation of stop band, in dBs. If left as 0, stopband frequency components are completely omitted.
`verify`	If TRUE, a plot comparing the spectrum of the input sound is compared the the filtered sound.

Details

This function performs lowpass, highpass and bandpass filtering by performing an FFT on the entire signal, zeroing out coefficients representing undesired frequency components, and performing an IFFT on the result. This approach may not be appropriate for some applications, but it is useful in some cases. This may be slow for long signals.

If verify is TRUE, a plot is created which allows the user to inspect the performance of the function.

Value

If a vector is given, the filtered vector is returned. If a 'sound' object is given, a sound object containing the filtered sound is returned.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## uncomment and run
##  white noise
#noise = rnorm(5000)

## low-pass filter
#synthfilter (noise, band = c(0, .25), verify = TRUE)

##  band-pass filter
#synthfilter (noise, band = c(.15, .25), verify = TRUE)

##  high-pass filter
#synthfilter (noise, band = c(.35, .5), verify = TRUE)
```

`t07` *Thomson (2007) Vowel Data*

Description

Mean formant frequency information for the vowels of male and female speakers from Edmonton English.

Usage

```
data (pb52)
```

Format

A data frame with 1520 observations on the following 9 variables:

`vowel` - The vowel category in x-sampa.

`gender` - A factor with levels `f` `m` representing speaker gender: male and female.

`f1` - A numeric vector indcating the vowel F1 in Hz.

`f2` - A numeric vector indcating the vowel F2 in Hz.

`f3` - A numeric vector indcating the vowel F3 in Hz.

References

Thomson, R. (2007). Modeling L1/L2 interactions in the perception and production of English vowels by Mandarin L1 speakers: A training study. PhD dissertation, University of Alberta.

Examples

```
#data(t07)
#vowelplot (t07$f1, t07$f2, t07$vowel, logaxes = 'xy', meansOnly = TRUE,
#xsampa = TRUE)
```

`voweldata` *Information about Vowel Data Sets*

Description

Contains a brief description and listing of all vowel data sets included in this package.

Usage

```
data (voweldata)
```

Format

A data frame with the following information:

`name` - The name of the data set.

`language` - The language represented in the data set.

`nspeakers` - The number of speakers.

`sexes` - The sexes of the speakers, m (male) and female (f).

`nformants` - The number of formants represented, between 2 and 4.

`individual` - If 'yes', individual speaker measurements are given, if 'no', measurements are averaged within gender.

References

At the moment 9 vowel data sets are included in this package. See individual data set pages for citation information.

Examples

```
#data(voweldata)
```

`vowelplot` *Plot Vowels*

Description

A flexible function that can create a wide variety of vowel plots (including IPA symbols).

Usage

```
vowelplot (f1s, f2s, labels = 0, xrange = NULL, yrange = NULL,
meansOnly = FALSE, ellipses = FALSE, ellipsesd = 1.96, add = FALSE,
pointType = 0, colors = NULL, logaxes = '', defaultPlot = TRUE,
alternateAxes = FALSE, xsampa = FALSE, ...)
```

Arguments

`f1s`	A numeric vector indicating vowel F1 frequencies.
`f2s`	A numeric vector indicating vowel F2 frequencies.
`labels`	A vector with labels for vowels. Must be provided for any category-dependent differences in plotting. If x-sampa labels are given IPA symbols may be plotted.
`xrange`	Allows the user to set the x axis range for the plot.
`yrange`	Allows the user to set the y axis range for the plot.
`meansOnly`	If TRUE, only category means are plotted (labels must be provided).
`ellipses`	If TRUE, standard deviation ellipses are drawn (one per category as indicated by label vector).
`ellipsesd`	A number indicating the number of standard deviations ellipses will enclose.
`add`	If TRUE, vowels are plotted on existing figure. If FALSE, a new one is created.
`pointType`	Kinds of points to use determined by 'pch' value. If specified it overrides text labels. IPA symbols may be plotted by finding appropriate values using the pickIPA() function included in this package.
`colors`	Colors to use for different categories. If specified this overrides automatic colors. It cycles through the list given if number of colors are less than number of categories.
`logaxes`	Linear axes are used by default. For log axes set to 'xy'.

`defaultPlot`	If TRUE, the function plots using pre-determined values. If FALSE, the user has almost complete control over the internal call of 'plot'.
`alternateAxes`	If TRUE, F1 is plotted on the y axis and F2 on the x axis with the origin in the top right corner. By default F1 is plotted on the x axis and F2 on the y axis with the origin in the bottom left corner.
`xsampa`	If TRUE, the labels vector given to the function is assumed to be specified in x-sampa and IPA symbols are used to plot using the xsampatoIPA() function included in this package. If this is set to TRUE and the 'labels' input is not in x-sampa, the symbols will be wrong.
`...`	Additional arguments are passed to the internal call of 'plot'.

Details

*** This function has been deprecated and is no longer being developed. I recommend using vplot() instead, which is more flexible and gives the user more control over plotting. ***

This function now includes functionality to easily generate vowel plots using IPA symbols. This relies on category labels being specified in x-sampa. Alternatively, the required plotting values for IPA symbols may be selected using the pickIPA() function included in this package, and then passed to the 'pointType' parameter.

There may be issues when exporting figures to PDF using IPA font. Exporting plots directly as images works 'out of the box'.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/X-SAMPA

Examples

```
## A few examples of some vowel plots.

#data (pb52)
#par (mfrow = c(1,4), mar = c(4.2,4.2,1,1))

# standard layout with linear axes
#vowelplot (pb52$f1, pb52$f2, pb52$vowel, xsampa = TRUE)

# alternate layout with log axes
#vowelplot (pb52$f1, pb52$f2, pb52$vowel, logaxes = 'xy',
#alternateAxes = TRUE, xsampa = TRUE)

# category means only
#vowelplot (pb52$f1, pb52$f2, pb52$vowel, logaxes = 'xy',
#meansOnly = TRUE, xsampa = TRUE)

# category means only with standard deviation ellipses
```

```
#vowelplot (pb52$f1, pb52$f2, pb52$vowel, logaxes = 'xy', meansOnly = TRUE,
#           ellipses = TRUE, xsampa = TRUE)
```

---

vowelsynth *Vowel Synthesis*

Description

Create synthetic vowels using a cascade formant synthesizer.

Usage

```
vowelsynth (ffs = c(270, 2200, 2800, 3400, 4400), fbw = 0.06,
dur = 300, f0 = c(120,100),fs = 10000, verify = FALSE,
returnsound = TRUE, noise1 = .001, noise2 = .01, power = NULL)
```

Arguments

`ffs`	A vector of center frequencies for each formant. For moving formants, a list containing two vectors may be provided, where the first vector indicates intitial values, and the final vector indicates final values. Formant frequencies should be provided in order of lowest to highest frequency.
`fbw`	A vector of formant bandwidths for each formant. See details for more information.
`dur`	The desired duration of the sound, in milliseconds. Vowels must be at least 50 ms long.
`f0`	The desired f0 (pitch) of the sound. Optionally, a vector with initial and final f0 values may be provided.
`fs`	The desired sampling frequency of the output sound.
`verify`	If TRUE, the waveform and spectrogram of the created sound are plotted to allow the user to visually verify the process.
`returnsound`	If TRUE, the sound is returned as a sound object, which can be used with several other functions included in this package. If FALSE, only a vector representing the sound wave is returned.
`noise1`	Noise to be added to the source, before formant filtering, as a proportion of the source RMS amplitude.
`noise2`	Noise to be added to the output, as a proportion of the signal RMS amplitude.
`power`	A desired power contour can be specified. Must be of the same length as the output sound, or the sound is truncated to the length of this vector.

Details

This function is a Klatt-style cascade formant synthesizer that is intended to create synthetic vowel sounds. The voice source is generated using the KLGLOTT88 method described in Klatt (1988).

If the vowels sound too 'robotic', there may be formants too close to the Nyquist frequency. Conversely, if the vowels sound too 'muffled', you may need to add more formants, and make sure they extend closer to the Nyquist frequency. Vowels sound more natural when f0 is not static.

Formant bandwidths may be provided in Hz, or as a percentage of the formant frequencies. To set these as a percent of formant frequencies, all values must be less than 1. If these are not provided they are set to 6 percent of the formant center frequencies by default. If only one value is provided, this is assumed to be the desired value for all formants.

Value

A vector or 'sound' object representing the filtered sound.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

Klatt, D. H. (1980). Software for a cascade/parallel formant synthesizer. Journal of the Acoustical Society of America 67(3): 971-995.

Klatt, D. H. (1988). Klattalk: The conversion of English text to speech. Unpublished Manuscript. Massachusetts Institute of Technology, Cambridge, MA. Chapter 3.

Examples

```
## uncomment to run

## The following examples are based on my vowels
#i = vowelsynth (returnsound = FALSE, f0 = c(125,105))
#a = vowelsynth (ffs = c(700, 1300, 2300, 3400, 4400),
#returnsound = FALSE, f0 = c(125,105))
#e = vowelsynth (ffs = c(400, 2000, 2600, 3400, 4400),
#returnsound = FALSE, f0 = c(125,105))
#o = vowelsynth (ffs = c(400, 900, 2300, 3400, 4400),
#returnsound = FALSE, f0 = c(125,105))
#u = vowelsynth (ffs = c(300, 750, 2300, 3400, 4400),
#returnsound = FALSE, f0 = c(125,105))

#silence = rep(0, 1000)
#vowels = c(a, silence, e, silence, i, silence, o, silence, u)

#writesound (vowels, filename = 'vowels.wav', fs = 10000)

## an example of a synthetic diphthong
#ei = vowelsynth (ffs = list(c(400, 2000, 2600, 3400, 4400),
#c(270, 2200, 2800, 3400, 4400)), f0 = c(125,105))
```

```
#writesound (ei)
#spectrogram (ei, pause = FALSE)
```

---

vplot *Plot Vowels*

Description

A flexible function that can create a wide variety of vowel plots (including IPA symbols).

Usage

```
vplot (x, y, labels = NULL, colors = NULL, points = NULL,
meansonly = FALSE, ellipsesd = 0, add = FALSE, alternateaxes = FALSE,
xsampa = FALSE, logaxes = FALSE, ...)
```

Arguments

`x`	A numeric vector indicating formant frequencies to be plotted on the x axis.
`y`	A numeric vector indicating formant frequencies to be plotted on the y axis.
`labels`	A vector with labels for vowels. Must be provided for any category-dependent differences in plotting. If x-sampa labels are given IPA symbols may be plotted.
`colors`	Colors to use for different categories. If specified this overrides automatic colors. It cycles through the list given if number of colors are less than number of categories.
`points`	Kinds of points to use determined by 'pch' value. If specified it overrides text labels. IPA symbols may be plotted by finding appropriate values using the pickIPA() function included in this package.
`meansonly`	If TRUE, only category means are plotted (labels must be provided).
`ellipsesd`	If a number greater than zero is given, ellipses are drawn enclosing this many standard deviations (one per category as indicated by label vector).
`add`	If TRUE, vowels are plotted on existing figure. If FALSE, a new one is created.
`logaxes`	Linear axes are used by default, for log axes set to TRUE.
`alternateaxes`	If TRUE, the origin in the top right corner of the plot, resulting in a configuration like the IPA vowel quadrilateral if F1 and F2 are provided. By default the origin in the bottom left corner.
`xsampa`	If TRUE, the labels vector given to the function is assumed to be specified in x-sampa and IPA symbols are used to plot using the xsampatoIPA() function included in this package. If this is set to TRUE and the 'labels' input is not in x-sampa, the symbols will be wrong.
`...`	Additional arguments are passed to the internal call of 'plot'.

Details

*** This function replaces the older vowelplot() function, which has been deprecated. ***

A flexible vowel plotting function, including functionality to easily generate vowel plots using IPA symbols. This relies on category labels being specified in x-sampa (the required plotting values for IPA symbols may be selected using the pickIPA() function included in this package).

Default parameter values are set for the plot, but these may all be overridden using the standard plotting parameters.

There may be issues when exporting figures to PDF using IPA font. Exporting plots directly as images works 'out of the box'.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/X-SAMPA

Examples

```
## A few examples of some vowel plots.

## load the Peterson and Barney data
#data (pb52)
#pb52 = pb52[pb52$type=='m',]  ## use only the males

#par (mfrow = c(3,2), mar = c(4.2,4.2,1,1))

# standard layout with linear axes
#vplot (pb52$f1, pb52$f2, pb52$vowel, xsampa = TRUE)

# alternate layout with log axes
#vplot (pb52$f1, pb52$f2, pb52$vowel, logaxes = TRUE,
#           alternateaxes = TRUE, xsampa = TRUE)

# category means only
#vplot (pb52$f1, pb52$f2, pb52$vowel, logaxes = TRUE,
#           meansonly = TRUE, xsampa = TRUE, cex = 3)

# category means only with standard deviation ellipses
#vplot (pb52$f1, pb52$f2, pb52$vowel, logaxes = FALSE,
#       meansonly = TRUE, ellipsesd = 2, xsampa = TRUE)

# same as above, with alternate axes
#vplot (pb52$f1, pb52$f2, pb52$vowel, logaxes = TRUE,
#       meansonly = TRUE, ellipsesd = 2, xsampa = TRUE,
#    alternateaxes = TRUE)

# individual points with standard deviation ellipses
# and alternate axes
```

```
#vplot (pb52$f1, pb52$f2, pb52$vowel, logaxes = TRUE,
#    meansonly = FALSE, ellipsesd = 2, xsampa = TRUE,
#    alternateaxes = TRUE)
```

windowfunc *Window Function*

Description

Generates a window function of a given type and length.

Usage

```
windowfunc (npoints, type = 'hann', parameter = -1)
```

Arguments

npoints	The desired window length, in points. If a vector is given, the window will have the same length as the vector.
type	A string indicating the type of window desired. For the sake of simplicity, all window names are in lowercase. Supported types are: rectangular, hann, hamming, cosine, bartlett, gausian, and kaiser.
parameter	The parameter necessary to generate the window, if appropriate. At the moment, the only windows that require parameters are the Kaiser and Gaussian windows. By default, these are set to 2 for kaiser and 0.4 for gaussian windows.

Details

A window function is generated, of the type and length specified by the user. This is returned as a numeric vector.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/Window_function

Examples

```
#par (mfrow = c(1,4))
#plot (windowfunc (100, 'hann'))
#plot (windowfunc (100, 'bartlett'))
#plot (windowfunc (100, 'kaiser', parameter = 2))
#plot (windowfunc (100, 'kaiser', parameter = 6))
```

writesound *Write out a WAV file*

Description

Create a WAV file from a numeric vector or 'sound' object.

Usage

```
writesound (samples, filename = '', fs = 22050)
```

Arguments

samples	A numeric vector representing a sound wave.
filename	A string indicating the desired output file name.
fs	The desired output sampling frequency. If a sound object is passed this does not need to be specified.

Details

This function generates single channel (mono), 16-bit WAV sound files at a desired sampling frequency. If a 'sound' object is passed, the filename and sampling frequency do not need to be set. If a filename is not set, the filename defaults to 'samples.wav' where 'samples' indicates the name of the samples variable that was passed to the function.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

https://ccrma.stanford.edu/courses/422/projects/WaveFormat/

Examples

```
## generate a sine wave with a frequency of 1000 Hz
## sampled at a frequency of 10000 Hz
#x = seq (0,.1, 1/10000)
#snd = sin (2*pi*1000*x)
#plot (snd[1:100], type = 'b')

## write out sine wave as a WAV file
# writesound (snd, filename = '1khz.wav', fs = 10000)

## if no filename is provided, this sound will be called 'snd.wav'
# writesound (snd, fs = 10000)
```

`xsampatoIPA` *Convert X-Sampa to IPA*

Description

Convert a vector of x-sampa vowel descriptions to values that can be used to plot IPA.

Usage

```
xsampatoIPA (vowels, chart = FALSE, verify = FALSE)
```

Arguments

`vowels`	A vector representing vowel labels in x-sampa to be converted to values that can be used to plot the corresponding IPA symbols. If this is not provided a plot comparing IPA to x-sampa representations is displayed.
`chart`	If TRUE, a plot comparing IPA to x-sampa representations is displayed.
`verify`	If TRUE, the selected symbols are plotted to allow for verification.

Details

This function converts x-sampa to values that can be used to plot IPA symbols by passing them to the 'pch' parameter within plotting functions. At the moment it has only been implemented for vowel sounds. The function also generate a figure comparing IPA to x-sampa vowel representations that may be useful to some users.

There may be issues when exporting figures to PDF using IPA font. Exporting plots directly as images works 'out of the box'

Value

A vector of the same length as 'vowels', with the values required to plot the desired vowels.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

References

http://en.wikipedia.org/wiki/X-SAMPA

Examples

```
## compare x-sampa and IPA vpwel charts
#xsampatoIPA ()
## some examples
#IPA = xsampatoIPA (c('I','3','e','Q'))
#plot (1:5, pch = IPA, cex = 2)
```

y96 *Yang (1996) Korean Vowel Data*

Description

Formant frequency information for vowels averaged across 60 male and female speakers.

Usage

```
data (y96)
```

Format

A data frame with the following columns:

`sex` - A factor indicating speaker sex.

`vowel` - The vowel category in x-sampa.

`f1` - A numeric vector indcating the vowel F1 in Hz.

`f2` - A numeric vector indcating the vowel F2 in Hz.

`f3` - A numeric vector indcating the vowel F3 in Hz.

References

Yang, B. (1996). A comparative study of American English and Korean vowels produced by male and female speakers. Journal of Phonetics, 24. 245-261.

Examples

```
#data(y96)
#vowelplot (y96$f1, y96$f2, y96$vowel, logaxes = 'xy', xsampa = TRUE)
```

zeros *Zero Vector/Matrix*

Description

Returns a vector or 2D matrix of zeros of the same shape as the input.

Usage

```
zeros (x,y=0)
```

Arguments

`x`	A scalar, vector, or two dimensional matrix.
`y`	An optional parameter to create a 2D matrix.

Details

If a scalar value is given, a vector of that many zeros is returned. If the y parameter is also specified, a matrix is returned with x rows and y columns.

If a vector is given, a vector of zeros of the same length is returned.

If a 2D matrix or dataframe is given, a matrix of zeros with the name number of rows and columns is returned.

Author(s)

Santiago Barreda <sbarreda@ucdavis.edu>

Examples

```
## A vector
zeros (5)
## A matrix
zeros (5, 3)

## Copy an existing vector
zeros (1:10)

## Copy an existing matrix
samplematrix = 1:10
zeros (samplematrix)
```

Index