Package 'eaR'

October 14, 2019

Title Perception-based Music Analysis

Version 0.1.2

Author Marc Vidal [aut, cre],
Marc Leman [aut]

Maintainer Marc Vidal <Marc.VidalBadia@UGent.be>

Description Analysis of music content and automatic extraction and description of musical features from Auditory Images, calculated from the acoustical signal.

License GPL (>= 2)

URL https://github.com/m-vidal/eaR

Depends R (>= 3.5.0)

Imports seewave, signal, stats, tuneR, utils

Suggests cowplot, ggplot2, reshape2

Encoding UTF-8

LazyData true

NeedsCompilation no

RoxygenNote 6.1.1

R topics documented:

`eaR-package` *Perception-based Music Analysis*

Description

Analysis of music content and automatic extraction and description of musical features from Auditory Images, calculated from the acoustical signal.

Some functions were ported from earlier versions of IPEM Package in Matlab.

Details

Package:	eaR
Type:	Package
Version:	0.1.2
Date:	2019-mm-dd
License:	GPL-2
URL:	`https://github.com/m-vidal/pv01`

Transformation of acoustic signals to Auditory Nerve Images (ANI) can be realized with function `CalcANI`. To make sure that this function works correctly, check the installation of the auditory model using `InstallAuditoryModel`. Results of the model (a time/period image) can be accessed through plot function `PlotImage`. Some functions are intended for the creation of auditory stimuli, for its subsequent transformation and analysis (`AdaptLevel`, `CalcNoteFrequency`, `ShepardTone`, `ShepardToneComplex`).

Further transformations from auditory nerve images can be taken using `PeriodicityPitch` and `LeakyIntegration` functions. Inferences on pitch images can be performed on `ContextualityIndex`. This function calculates the goodness of fit of tone patterns (*probe*) for a given tonal context.

Additionally, `CalcOnsetsFromANI` finds onsets of sound events and `RoughnessFFT` analyses the roughness (or equivalently: the sensory dissonance) of a sound. Generally, plots of the analysis can be accessed on the generated object.

The rest of the functions have an accessory character, and can eventually be used in certain circumstances in combination to the main ones.

Author(s)

Marc Vidal and Marc Leman; credits are also due to Koen Tanghe, Micheline Lesaffre.

References

Langner, G. (1992). Periodicity coding in the auditory system. *Hearing Research*, 60, pp.115–142.

Langner, G., Benson, C. (2015). *The Neural Code of Pitch and Harmony*. Cambridge University Press.

Leman, M. (2000). An auditory model of the role of short-term memory in probe-tone ratings. *Music Perception*, 17(4), pp.481-509.

Leman, M. (2000). Visualization and calculation of the roughness of acoustical musical signals using the synchronization index model (SIM). *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00)*, Verona, Italy.

Leman, M., Lesaffre, M., Tanghe, M. (2001). Introduction to the IPEM Toolbox for Perception-based Music Analysis. *Proceedings of the XIII Meeting of the FWO Research Society on Foundations of Music Research*, Ghent.

Ligges, U., Krey, S., Mersmann, O., Schnackenberg, S. (2018). tuneR: Analysis of Music and Speech. R package version 1.3.3.

Smith, L. (1996). Onset-based Sound Segmentation. *Advances in neural information processing systems*, vol.9, pp.729-735.

Sueur, J. (2018). *Sound analysis and synthesis with R*. Heidelberg, Germany: Springer.

Van Immerseel, L. Van and Martens, J. (1992). Pitch andvoiced/unvoiced determination with an auditory model. *The Journal of the Acoustical Society of America*, vol. 91, pp.3511-3526.

`AdaptLevel` *Adapt RMS Power Level*

Description

This function adapts the Root Mean Square (RMS) power level of the signal $s(t)$ to the specified dB level. The RMS is calculated in the time domain for the sampled signal as:

$$RMS\{s(t)\} = \sqrt{\frac{1}{N}\sum_t s^2(t)}$$

where N is the sample size. To calculate the factor to adapt the signal,

$$A = 10 * \log_{10}(\mathrm{P}/RMS)$$

where RMS is the reference to which P is being compared.

Usage

```
AdaptLevel(inSignal, indB)
```

Arguments

`inSignal`	a numeric vector representing the input signal.
`indB`	a scalar indicating the wanted level (0 is maximum level without clipping, -20 is reasonable).

Details

The reference value of 0 dB is the level of a square wave with amplitude 1. Thus, a sine wave with amplitude 1 yields -3.01 dB.

Value

A numeric vector representing the adapted input signal.

Note

The multi-channel version of `AdaptLevel` is deprecated and only is available in **IPEM** Toolbox.

Author(s)

Marc Vidal (`R` version). Based on the original code from **IPEM** Toolbox.

Examples

```
AdaptLevel(SchumannKurioseGeschichte, -20)
```

`CalcANI` *Auditory Nerve Image*

Description

This function calculates the Auditory Nerve Image (ANI) for the given signal $s(t)$ using an adapted version of Van Immerseel and Martens (1992) model of the auditory periphery. The model is implemented in c-code and stored in the dependencies of the package operating in parallel to `R`.

We consider the input signal $s(t), t \in \mathcal{T} = \{1, 2, 3 \ldots n\}$ to be transformed to a multidimensional signal (an image) which is a matrix. The output of the model is known as Auditory Nerve Image or *Primary Image*. This can be noted as $\mathbf{A} = A(k, t)$, where $\{\mathbf{c}_k(t)\}_{k=1,2,\ldots,m}$ are frequency bands (or channels) at time t. Thus, for a fixed $\mathbf{c}_k$ we have a discrete time series representing the rate-code of neural discharge in a channel k.

Before-mentioned transformation processes are described in more detail in Immerseel and Martens (1992) but prim/rily consist of a simulation of the cochlear mechanical filtering using an array of overlapping band-pass filters. Then, the musical signal is decomposed in different subbands and represented as neural patterns. The frequency range covered by the auditory filters depends on the center frequencies of the filters, the number of chosen channels and the chosen distance between the channels. In most of our simulations, we chose forty channels ($m = 40$), half a critical band apart from each other. This covers a range from 140 Hz to 8877 Hz.

To let the auditory model process the acoustical signal, note that the function `CalcANI` resamples all signal inputs automatically to a sampling rate of 22050 Hz. The outcome of the auditory model has a sample frequency of 11025 Hz, but for most calculations, this can be downsampled to a lower value ($f_A = 11025/4$ is the default).

Usage

```
CalcANI(inSignal, inSampleFreq,
        inDownsamplingFactor = 4, inNumOfChannels = 40,
        inFirstCBU = 2, inCBUStep = 0.5)
```

Arguments

`inSignal`	the sound signal to be processed. It can either be a `"WAVE"` object (**tuneR**), a numeric vector or a `"*.wav"` file stored in the Working Directory.
`inSampleFreq`	the sample frequency of the input signal (in Hz).
`inDownsamplingFactor`	the integer factor by which the outcome of the auditory model is downsampled (use 1 for no downsampling). If empty or not specified, 4 is used by default.
`inNumOfChannels`	number of channels to use. If empty or not specified, 40 is used by default.
`inFirstCBU`	frequency of first channel (in critical band units). If empty or not specified, 2 is used by default.
`inCBUStep`	frequency difference between channels (in cbu). If empty or not specified, 0.5 is used by default.

Value

An object of class `"AI"`, which is a list with the following elements:

`AuditoryNerveImage`	a matrix of size *n* x *m* representing the auditory nerve image, where *n* is the number of channels and *m* is the number of samples.
`ANIFreq`	sample frequency of ANI (in Hz).
`ANIFilterFreqs`	center frequencies used by the auditory model (in Hz).

Author(s)

Marc Vidal (`R` version). Based on the original code from **IPEM** Toolbox.

References

Van Immerseel, L. Van and Martens, J. (1992). Pitch and voiced/unvoiced determination with an auditory model. *The Journal of the Acoustical Society of America*, vol. 91, pp.3511-3526.

Examples

```
data(SchumannKurioseGeschichte)
s <- SchumannKurioseGeschichte
ANIs <- CalcANI(s, 22050)
```

`CalcNoteFrequency` *Note Frequency*

Description

This function calculates any note frequency for an equal-tempered scale within a range of eight octaves (0-7). The used formula is,

$$f_n = f_0 * (a)^n$$

where f_0 is a fixed frequency defined at 440 Hz (A_4), and the coefficient $a = \sqrt[12]{2}$ allows to calculate the note frequency through the steps $n \in \mathbf{Z}$ in a bounded interval, being $n_{A_4} = 0$.

Usage

```
CalcNoteFrequency(inNote, inOctave)
```

Arguments

`inNote`	a note string A-G with or without accidental (e.g. "Bb"). Both sharps (#) and flats (b) can be used.
`inOctave`	the octave number of the note if empty or not specified, `inNoteNr` is assumed to contain the octave specification.

Details

`CalcNoteFrequency` is a shorter version of `IPEMCalcNoteFrequency` (**IPEM** Toolbox).

Value

A numeric giving the frequency (Hz) of the indicated note.

Author(s)

Marc Vidal.

Examples

```
CalcNoteFrequency("A")
```

`CalcOnsetsFromANI` *Calculates Onsets from an ANI*

Description

This function calculates the onsets for a given signal represented by its auditory nerve image, using an integrate-and-fire neural net layer.

Usage

```
CalcOnsetsFromANI(inANIObj)
```

Arguments

`inANIObj`	an object of class "`AI`" which must contain an auditory nerve image, its sample frequency and the filterbank frequencies used by the auditory model.

Value

`OnsetSignal`	A signal having a non-zero value for an onset, and zero otherwise (the higher the non-zero value, the more our system is convinced that the onset is really an onset).
`OnsetFreq`	sample frequency of the result (same as `inSampleFreq`).

Author(s)

Marc Vidal (R version). Based on the original code from **IPEM** Toolbox.

Examples

```
data(SchumannKurioseGeschichte)
s <- SchumannKurioseGeschichte
ANIs <- CalcANI(s, 22050)
CalcOnsetsFromANI(ANIs)
```

CalcRMS *Calculate RMS*

Description

This function calculates the running RMS value of the given signal: a RMS value is generated every `inFrameInterval` seconds over a period of `inFrameWidth` seconds. Thus, for each frame F in each channel $\mathbf{c}_k(t)$ of an auditory nerve image $\mathbf{A}$, an RMS value is calculated according to:

$$RMS\{F_i(\mathbf{c}_k(t))\} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} F_i^2}$$

where N is frame width in samples, and F_i is the *i*-th sample of frame F.

Usage

```
CalcRMS(inSignal, inSampleFreq, inFrameWidth, inFrameInterval)
```

Arguments

`inSignal`	the input signal (if this is a matrix, RMS values are calculated for each channel (ie. row) in the signal).
`inSampleFreq`	the sample frequency of the input signal (in Hz).
`inFrameWidth`	the period over which the RMS is calculated (in s).
`inFrameInterval`	the period between two successive frames (in s).

Value

An object of class `"AI"` that contains the running RMS value (`RMSSignal`) of the given signal and its sample frequency (`RMSFreq`).

Author(s)

Marc Vidal (R version). Based on the original code from **IPEM** Toolbox.

Examples

```
data(SchumannKurioseGeschichte)
s <- SchumannKurioseGeschichte
ANIs <- CalcANI(s, 22050)
RMS <- CalcRMS(ANIs$AuditoryNerveImage, ANIs$ANIFreq, 0.029, 0.0058)
```

`Clip` *Clip a Multi-channel Signal*

Description

This function clips the incoming (multi-channel) signal at the given limits. Specify empty values for either one of the limits if you don't want clipping at that side of the signal range. Signal channels are represented by rows.

Usage

```
Clip(inSignal, inLowLimit = NULL, inHighLimit = NULL,
     inClipLowTo = NULL, inClipHighTo = NULL)
```

Arguments

`inSignal`	a matrix representing the (multi-channel) signal to be clipped.
`inLowLimit`	specifies the low level clipping value: values lower than this are replaced by either the limit itself or `inClipLowTo`. If empty or not specified, no clipping occurs.
`inHighLimit`	specifies the high level clipping value: values higher than this are replaced by either the limit itself or `inClipHighTo`. If empty or not specified, no clipping occurs.
`inClipLowTo`	if non-empty, this is a replacement value for too low values. If empty or not specified, `inLowLimit` is used.
`inClipHighTo`	if non-empty, this is a replacement value for too high values. If empty or not specified, `inHighLimit` is used.

Value

A matrix representing the clipped signal.

Author(s)

Marc Vidal (R version). Based on the original code from **IPEM** Toolbox.

Examples

```
signal <- matrix(runif(64), ncol=8)
Clip(signal, 0.5 ,0.7, 0, 1)
```

`ContextualityIndex` *Tone Contextuality Index*

Description

Contextuality measures the pitch commonality between two running pitch images of the same sound, each having a possible different echo T. In Leman (2000), contextuality is used to measure the pitch commonality of local (short echo) images versus global (long echo) images.

This function uses a pitch image $\mathbf{P}$ to determine two echoed images with a different T using `LeakyIntegration`. Then, we calculated the Pearson correlation coefficient concerning the following methods:

- *Inspection*: compares a fixed local image (or *probe*) with running local images and running global images. In this case, $\mathbf{P}^T$ refers to both running local and global images while the fixed local image is the last sample of it; thus, the correlation coefficient can be expressed as

$$\rho_i = \frac{1}{m-1}\sum_{k=1}^{m}\left(\frac{\mathbf{P}_{ik}^{T}-\mu_i}{\sigma_i}\right)\left(\frac{\mathbf{P}_{nk}^{T}-\mu_n}{\sigma_n}\right).$$

- *Comparison*: compares a running local image $\mathbf{P}^{T_1}$ with running global image $\mathbf{P}^{T_2}$. Then the correlation coefficient between corresponding columns is

$$\rho_i = \frac{1}{m-1}\sum_{k=1}^{m}\left(\frac{\mathbf{P}_{ik}^{T_1}-\mu_i}{\sigma_i}\right)\left(\frac{\mathbf{P}_{ik}^{T_2}-\mu_i}{\sigma_i}\right).$$

Usage

```
ContextualityIndex(inANIObj, inSnapShot = NULL, inHalfDecayChords = 0.1,
                   inHalfDecayToneCenters = 1.5, inEnlargement = 0)
```

Arguments

`inANIObj`	an object of class "AI". It must contain the output of the `PeriodicityPitch` function: a periodicity pitch image, its sample frequency (in Hz) and the periods of periodicity analysis (in s).
`inSnapShot`	time where the snapshot should be taken (in s). If negative, the time is taken at `abs(inSnapShot)` from the end of the sample. If empty or not specified, the time of the last sample is used by default.
`inHalfDecayChords`	half decay time for leaky integration into chord image (local image). If empty or not specified, 0.1 is used by default.
`inHalfDecayToneCenters`	half decay time for leaky integration into tone center image (global image). If empty or not specified, 1.5 is used by default.
`inEnlargement`	time by which the input signal is enlarged (in s). If `-1`, `2*inHalfDecayToneCenters` is used. If empty or not specified, 0 is used by default.

Details

Sample frequency of output signals is the same as `inSampleFreq`.

Value

An object of class "`AI`", which is a list with the following elements:

`ChordsImage`	local integration of `inPeriodicityPitch` into chord image.
`ToneCentersImage`	global integration of `inPeriodicityPitch` into tone center image.
`LocalInspection`	correspondence between chord taken at snapshot position and running chord image.
`GlobalInspection`	correspondence between chord taken at snapshot position and running tone center image
`Comparison`	correspondence between running chord image and running tone center image.

Note

Sample frequency of output signals is the same as `inSampleFreq`.

Author(s)

Marc Vidal (R version). Based on the original code from **IPEM** Toolbox.

References

Leman, M. (2000). An auditory model of the role of short-term memory in probe-tone ratings. *Music Perception*, 17(4), pp.481-509.

Examples

```
C  <- ShepardToneComplex(c(1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0),0.75,22050,1,-20)
F  <- ShepardToneComplex(c(1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0),0.75,22050,1,-20)
G  <- ShepardToneComplex(c(0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1),0.75,22050,1,-20)
s<-c(C, F, G, C)
ANIs <- CalcANI(s, 22050)
PPs <- PeriodicityPitch(ANIs)
CIs <- ContextualityIndex(PPs)
```

`FindAllPeaks` *Find All Peaks in a Signal*

Description

This function finds the indices of all peaks in the given signal vector. A peak is taken to occur whenever a rising part in the signal is followed by a falling part.

Usage

```
FindAllPeaks(inSignal, inFlatPreference = "center")
```

Arguments

`inSignal`	the signal that is scanned for peaks (a `numeric` vector).
`inFlatPreference`	preference for choosing exact position in case of flat peaks: • if "left", the leftmost point of a flat peak is chosen • if "center", the center of a flat peak is chosen • if "right", the rightmost point of a flat peak is chosen. If empty or not specified, "center" is used by default.

Value

A numeric vector with the indices of the peaks (could be empty!).

Author(s)

Marc Vidal (`R` version). Based on the original code from **IPEM** Toolbox.

Examples

```
data(SchumannKurioseGeschichte)
Ps <- FindAllPeaks(SchumannKurioseGeschichte)
```

`FindNearestMinima` *Find Nearest Minima*

Description

Finds the nearest minima to the left and the right of the specified peak. Peaks can be found with `FindAllPeaks`.

Usage

```
FindNearestMinima(inSignal, inPeakIndex)
```

Arguments

`inSignal`	the index of the nearest miminum to the left of the peak if no real minimum was found, 1 is returned.
`inPeakIndex`	the index of the nearest miminum to the right of the peak. If no real minimum was found, `length(inSignal)` is returned.

Value

A list of two elements:

`LeftIndex`	the index of the nearest miminum to the left of the peak. If no real minimum was found, 1 is returned.
`RightIndex`	the index of the nearest miminum to the right of the peak. If no real minimum was found, `length(inSignal)` is returned.

Author(s)

Marc Vidal (R version). Based on the original code from **IPEM** Toolbox.

Examples

```
data(SchumannKurioseGeschichte)
Ps <- FindAllPeaks(SchumannKurioseGeschichte)
FindNearestMinima(SchumannKurioseGeschichte, Ps)
```

`InstallAuditoryModel` *Install the Auditory Model*

Description

Running the auditory model requires giving permissions to execute a compiled version of the auditory modelling library in c-code. Before start working with **eaR** use this function.

Usage

```
InstallAuditoryModel(inOs = NULL)
```

Arguments

`inOs`	On which system do you want to check the installation of the auditory model? It can be a character vector (selecting from "mac", "linux", "win") indicating your operating system.

Details

This function is for MAC and Linux users, while Windows users can verify if the model is installed correctly.

Value

Depending on the system you will get a specific message about the installation process. The window where the files are located will open (only Windows).

Author(s)

Marc Vidal.

Examples

```
InstallAuditoryModel("win")
```

`LeakyIntegration` *Leaky Integration*

Description

An echo can be defined as a leaky integration with a given half decay time. At each moment in time, a leaky integrator adds the incoming signal value to an attenuated version of the previously calculated value to form the newly calculated value. The half decay time specifies the time it takes for an impulse signal to reach half its original value.

This function takes an image as input and gives the leaky integrated image as output. Let us suppose that we have a Periodicity Pitch Image $\mathbf{P}$ with a factor of enlargement $k = 0$, then the new echoed image is calculated as

$$\widehat{\mathbf{P}}_{*,j} = \mathbf{P}_{*,j} + \widehat{\mathbf{P}}_{*,j-1} \cdot 2^{\frac{-1}{T}} \quad \forall j > 1$$

where T denotes the echo in seconds and the columns of both matrices are indexed with the subscript j. Leaky integration had been used in Leman (2000) to construct the pitch images of an echoic memory.

Usage

```
LeakyIntegration(inSignal, inSampleFreq, inHalfDecayTime = 0.1,
                 inEnlargement = 0)
```

Arguments

`inSignal`	input signal (multi-dimensional), each row representing a channel leaky integration is performed for each channel.
`inSampleFreq`	sample frequency of inSignal (in Hz).
`inHalfDecayTime`	time (in s) at which an impulse would be reduced to half its value. If empty or not specified, 0.1 is used by default.
`inEnlargement`	time (in s) to enlarge the input signal. If -1, `2*inHalfDecayTime` is used. If empty or not specified, 0 is used by default.

Details

Sample frequency of `outLeakyIntegration` is the same as `inSampleFreq`.

Value

An object of `matrix` class representing the leaky integration of the input signal.

Author(s)

Marc Vidal (`R` version). Based on the original code from **IPEM** Toolbox.

References

Leman, M. (2000). An auditory model of the role of short-term memory in probe-tone ratings. Music Perception, 17(4), 481-509.

Examples

```
s <- c(SchumannKurioseGeschichte)
ANIs <- CalcANI(s, 22050)
LIs <- LeakyIntegration (ANIs$AuditoryNerveImage, ANIs$ANIFreq, 0.1, -1)
```

`OnsetPattern` *Onset Pattern*

Description

Integrate-and-fire neural net for onset-detection. Based on an article by Leslie S. Smith (1996). Neuron dynamics are given by: `dA/dt` = `Input(t) -diss*A`, being `A` the neurons' accumulated value and I the dissipation

Usage

```
OnsetPattern(inSignal, inSampleFreq)
```

Arguments

`inSignal`	a matrix of size *n* x *m* where *n* is the number of channels and *m* is the length in samples.
`inSampleFreq`	the sample frequency of the incoming signal (in Hz).

Value

`OnsetPattern`	a matrix of size *n* x *m* where an element is 1 if an onset was detected.
`OnsetPatternFreq`	sample frequency of result (same as `inSampleFreq`).

Author(s)

Marc Vidal (`R` version). Based on the original code from **IPEM** Toolbox.

References

Smith, L. (1996). Onset-based Sound Segmentation. *Advances in neural information processing systems*, vol.9, pp.729-735.

Examples

```
data(SchumannKurioseGeschichte)
s <- SchumannKurioseGeschichte
ANIs <- CalcANI(s, 22050)
OPs <- OnsetPattern(ANIs$AuditoryNerveImage, ANIs$ANIFreq)
```

`OnsetPatternFilter` *Filter* `OnsetPattern` *Output*

Description

This function takes an (multi-channel) onset pattern generated by `OnsetPattern`, and outputs a (one-dimensional) signal in which a non-zero value means an onset occurred (the value is a measurement of the likeliness for the onset). Currently, the following (simple) strategy is followed: an onset is detected at a certain moment, if a certain fraction of channels have an onset within a specific period of time and, the moment falls at least a minimum period behind the last detected onset.

Usage

```
OnsetPatternFilter(inOnsetPattern, inSampleFreq)
```

Arguments

`inOnsetPattern`	a matrix of size *n* x *m* where *n* is the number of channels and *m* is the length in samples.
`inSampleFreq`	the sample frequency of the incoming signal (in Hz).

Value

`OnsetSignal`	a 1 dimension signal having a non-zero value at detected onset-times (the clearer the onset, the higher the value).
`OnsetSignalFreq`	sample frequency of the result (same as `inSampleFreq`).

Author(s)

Marc Vidal (R version). Based on the original code from **IPEM** Toolbox.

Examples

```
data(SchumannKurioseGeschichte)
s <- SchumannKurioseGeschichte
ANIs <- CalcANI(s, 22050)
OPs <- OnsetPattern(ANIs$AuditoryNerveImage, ANIs$ANIFreq)
OnsetPatternFilter(OPs$OnsetPattern, OPs$OnsetPatternFreq)
```

`OnsetPeakDetection` *Onset Peak Detection*

Description

Returns a pattern having a non-zero value on moments of possible onset peaks.

Usage

```
OnsetPeakDetection(inSignal, inSampleFreq)
```

Arguments

`inSignal`	a matrix of size *n* x *m* where *n* is the number of channels and *m* is the length in samples.
`inSampleFreq`	the sample frequency of the incoming signal (in Hz).

Value

A list of two arrays:

`OnsetResults`	a matrix of size *n* x *m* with a value between 0 and 1 (non-zero where an onset occurs, and proportional to the importance of the onset).
`OnsetResultsFreq`	sample frequency of the result (same as `inSampleFreq`).

Author(s)

Marc Vidal (`R` version). Based on the original code from **IPEM** Toolbox.

Examples

```
data(SchumannKurioseGeschichte)
s <- SchumannKurioseGeschichte
ANIs <- CalcANI(s[1:22050], 22050)
OnsetPeakDetection(ANIs$AuditoryNerveImage, ANIs$ANIFreq)
```

`OnsetPeakDetection1Channel` *Onset Peak Detection in One Channel*

Description

Returns the indices of the "important" peaks in the input signal, together with an indication of the "importance" of the peak.

Usage

```
OnsetPeakDetection1Channel(inSignal, inSampleFreq)
```

Arguments

`inSignal`	the signal to be scanned for peaks.
`inSampleFreq`	the sample frequency of the signal (in Hz).

Value

A list of two arrays:

`PeakIndices`	the indices within the input signal where the peaks are found.
`Importances`	a value between 0 and 1 showing the "importance" of a peak.

Author(s)

Marc Vidal (R version). Based on the original code from **IPEM** Toolbox.

Examples

```
data(SchumannKurioseGeschichte)
OnsetPeakDetection1Channel(SchumannKurioseGeschichte, 22050)
```

PeriodicityPitch *Periodicity Pitch Image*

Description

This function calculates the periodicity pitch of an auditory nerve image containing neural firing patterns in different auditory channels. It is based on the idea that periodicity pitch is calculated as the summed autocorrelation over bandpass filtered fluctuations in auditory channels.

We consider the auditory nerve image $\mathbf{A}$ and its sample frequency f_A, from the output of the function `CalcANI`. The channels (rows) of $\mathbf{A}$ are filtered between 80 and 1250 Hz. Due to the fact that the output of the auditory model gives the envelopes of the neural firing probabilities (< 1250Hz), it suffices to first apply a low-pass filter and then subtract that from the original signal in order to obtain the pitch. Thus, let F be a second order Butterworth filter with a cutoff frequency of 80 Hz, then,

$$\widehat{\mathbf{A}} = \mathbf{A} - F[\mathbf{A}].$$

The lower limit of 80 Hz accounts for the fact that for smaller frequencies, the sensation of pitch becomes more a sensation of textural properties. The higher limit of 1250 Hz is related to the limits of neural synchronization. Beyond about 1250 Hz, the neurons are no longer able to follow the exact period of the signal very accurately, and periodicity pitch becomes unreliable. Only the lowest frequency can be changed by `inLowFrequency`.

A frame-based auto-correlation analysis is performed on the filtered channels. A frame width FW and step size FS are chosen and then for each frame and each channel $\mathbf{c}_k$ of $\widehat{\mathbf{A}}$, we perform an auto-correlation for each time-lag δ,

$$r_k^j(t) = \int_t^{t+FW} \mathbf{c}_k^j(\tau)\mathbf{c}_k^j(\tau+\delta)d\tau$$

where $\delta \in [0, FW]$, τ denotes the periods of the periodicity analysis and $j \in \lfloor 1, \mathsf{int}(t/FS) \rfloor$. A coincidence mechanism is fulfilled by the summation of the auto-correlation results over all channels, so that

$$\mathbf{P}_{*,j} = \sum_{k=1}^{m} r_k^j(t)^T$$

where $\mathbf{P}$ is the pitch image, that is related to the notion of virtual pitch pattern. It gives an account of the common periodicity along the auditory neurons in the frequency region of 80 Hz to 1250 Hz.

Usage

```
PeriodicityPitch(inANIObj, inLowFrequency = 80,
                 inFrameWidth = 0.064, inFrameStepSize = 0.010)
```

Arguments

`inANIObj`	an object of class "`AI`". It must contain the output of `CalcANI` function: • a matrix of size *n* x *m* where *n* is the number of auditory channels (<= 40) and *m* is the number of samples • the sample frequency of the input signal (in Hz).
`inLowFrequency`	cutoff frequency (in Hz) of a first order lowpass filter applied before calculating the autocorrelation. If empty or not specified, 80 is used by default.
`inFrameWidth`	width of the frame used for the accumulation of the autocorrelation (in s). If empty or not specified, 0.064 is used by default.
`inFrameStepSize`	stepsize or time interval between two `inFrameWidth` (in s). If empty or not specified, 0.010 is used by default.

Details

As for any frame-based function, the first value of the output signal is the value calculated for the first complete frame in the input signal. This means the following: if you have an input signal of length 1 s at a sample frequency of 1000 Hz, a frame width of 0.050 s, and a frame step size of 0.010 s, then there will be `ceiling(((1 -0.050)*1000 + 1)/(0.010*1000)) = 96` values in the output signal, where the first value corresponds to the first complete frame (the interval 0 to 0.050 s).

Value

An object of class "`AI`", which is a list with the following elements:

`PeriodicityPitchImage`	periodicity pitch: a matrix of size `inFrameWidth * length(inMatrix) / outSampleFreq`
`SampleFreq`	sampling rate, equal to inSampleFreq/inFrameStepSize (in Hz).
`Periods`	analyzed periods (in s).
`BPANI`	bandpass filtered auditory nerve images (at the original sample frequency).

Author(s)

Marc Vidal (`R` version). Based on the original code from **IPEM** Toolbox.

Examples

```
probe <- ShepardTone(293.66, 1, indBLevel = -20)
s <- c(SchumannKurioseGeschichte, numeric(2205), probe)
ANIs <- CalcANI(s, 22050)
PPs <- PeriodicityPitch(ANIs)
```

`PlotImage` *Plot an Auditory Image*

Description

Auditory images reflect features of the sound as internal representations, in other words as brain activations so to speak. This function display Auditory Images from `CalcANI` (Auditory Nerve Image) and `PeriodicityPitch` (Pitch Image) or an "`AI`" data object (future versions).

Usage

```
PlotImage(inAIObj)
```

Arguments

`inAIObj`	A object of class "`AI`" which contains the matrix to be represented

Details

This function uses package **ggplot2** for representations.

Value

'done'

Author(s)

Marc Vidal.

Examples

```
data(SchumannKurioseGeschichte)
s <- SchumannKurioseGeschichte
ANIs <- CalcANI(s, 22050)
PPs <- PeriodicityPitch(ANIs)
PlotImage(PPs)
```

`RoughnessFFT` *Calculation of the Roughness of Acoustical Musical Signals*

Description

This function estimates the roughness (or equivalently: the sensory dissonance) of a sound, from its Auditory Nerve Image calculated in `CalcANI`.

Roughness is considered to be a sensory process highly related to texture perception. The visualization and calculation method is based on Leman (2000), where roughness is defined as the energy of the relevant beating frequencies in the auditory channels. The model is called the synchronization index model of roughness, and it is based on phase-locking to frequencies that are present in the neural patterns. The synchronization index model allows a straightforward visualization of the energy components underlying roughness, in particular concerning auditory channels and to phase-locking synchronization (the synchronization index for the relevant beating frequencies on a frequency scale).

Beating frequencies are those at which the sound oscillates in amplitude. Thus, for an amplitude modulated sine wave with carrier frequency f_c and modulated frequency f_m, its spectrum has only three frequencies: f_c, $f_c - f_m$ and $f_c + f_m$. The beating frequency is f_m. Now, in the auditory system, these frequencies are introduced as elective beating frequencies into the spectrum of the neural rate-code patterns. This is because to wave rectification in the cochlea, where the lower part of the modulated signal is cut off. As a result, the new frequencies are introduced of which the most important ones correspond with the beating frequency f_m and its multiples. Neurons may synchronize with these frequencies provided that they fall in the frequency range where synchronization is physiologically possible. This mechanism forms a physiological basis for the detection of beats and hence, the sensation of roughness.

The synchronization index model calculates roughness in terms of the energy of neural synchronization to the beating frequencies. The energy refers to a quantity which we derive from the magnitude spectrum. Since the beating frequencies are contained in the lower spectral area of the neuronal pattern $\mathbf{c}_k(t)$, the spectral part we are interested in is defined as:

$$\mathbf{b}_k(\omega) = \mathbf{f}_k(\omega)|\mathbf{d}_k(\omega)|$$

where $\mathbf{f}_k(\omega)$ is a filter whose spectrum is depending on the channel k, and $\mathbf{d}_k(\omega)$ is the short-term spectrum which we define as:

$$\mathbf{d}_k(\omega) = \int_{-\infty}^{+\infty} \mathbf{c}_k(t) w\,(t' - t)\, e^{-2\pi i \omega t'} dt'$$

where $w\,(t' - t)$ is a (hamming) window and $\{\mathbf{d}_k(\omega)\}_{k=1,2,\ldots,m}$ are the rows of $D_t(k, \omega)$. The *magnitude* spectrum is then defined as $|D_t(k, \omega)|$ and the *phase* spectrum as $\angle D_t(k, \omega)$. In order to be able to reproduce the psychoacoustical data on roughness, the filters $\mathbf{f}_k(\omega)$ should be more narrow at auditory channels where the center frequency is below 800 Hz, and the filters should be attenuated for high center frequencies as well.

The pattern $\{\mathbf{b}_k(\omega)\}_{k=1,2,\ldots,m}$ are the rows of $B_t(k, \omega)$ which represents the *spectrum* of the neural synchronization to the beating frequencies in channel k. The *synchronization index* (Javel et al., 1988) of the beating frequencies is then defined as the normalized magnitude:

$$I_t(k, \omega) = \left| \frac{B_t(k, \omega)}{D_t(k, 0)} \right|$$

where $I_t(k, \omega)$ is the normalized magnitude in channel k. $D_t(k, 0)$ is the DC-component of the whole signal in channel k. The short-term energy spectrum of the neural synchronization to beating frequencies in a particular auditory channel k is defined by:

$$\hat{B}_t(k, \omega) = I_t(k, \omega)^\alpha$$

where $(1 < \alpha < 2)$ is a parameter which can be related to the power law (Leman, 2000). For this experimental setup 1.6 is chosen. We then define the following relationships for the calculation of roughness:

$$\mathbf{r}_{\hat{B}_\omega}(t) = \int \hat{B}_t(\omega) df \quad = \int \sum_{k=1}^{m} \hat{B}_t(k, \omega) df$$

$$\mathbf{r}_{\hat{B}_k}(t) = \sum_{k=1}^{m} \hat{B}_t(k) = \sum_{k=1}^{m} \int \hat{B}_t(k, \omega) df$$

where $\mathbf{r}$ is the roughness at time t, obtained by an integration of the energy over all frequencies, as well as over all channels. This definition implies a proper visualization, one along the axis of auditory channels and one along the axis of the (beating) frequencies.

Usage

```
RoughnessFFT(inObjANI, inFrameWidth = 0.2, inFrameStepSize = 0.02)
```

Arguments

`inObjANI`	an object of class `"AI"` which must contain an auditory nerve image, its sample frequency and the filterbank frequencies used by the auditory model.
`inFrameWidth`	the width of the window for analysing the signal (in s). If empty or not specified, 0.2 s is used by default.
`inFrameStepSize`	the stepsize or time interval between two `inFrameWidthInSampless` (in s).

Details

For now, the roughness values are dependend on the used frame width. So, to make usefull comparisons, only results obtained using the same frame width should be compared.

Value

`outFFTMatrix1`	visualisation of energy over channels.
`outFFTMatrix2`	visualisation of energy spectrum for synchronization (synchronisation index SI).
`outRoughness`	roughness over signal.
`outSampleFreq`	sampling rate of outRoughness (in Hz).
`PlotRoughness`	the plot of the roughness over signal.

Author(s)

Marc Vidal (`R` version). Based on the original code from **IPEM** Toolbox.

References

Javel, E., McGee, J., Horst, J., & Farley, G. (1988). Temporal mechanisms in auditory stimulus coding. In G. Edelman, W. Gall, & W. Cowan (Eds.), *Auditory function: neurobiological bases of hearing*. New York: John Wiley and Sons.

Leman, M. (2000). Visualization and calculation of the roughness of acoustical musical signals using the synchronization index model (SIM). In: *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00)*, Verona, Italy.

Examples

```
data(SchumannKurioseGeschichte)
s <- SchumannKurioseGeschichte
ANIs <- CalcANI(s, 22050)
Rs <- RoughnessFFT(ANIs)
```

SchumannKurioseGeschichte — *Schumann, "Kuriose Geschichte"*

Description

This is a sampled signal (22050) of a WAV file containing the first bars of the piece "Kuriose Geschichte" (Curious Story) from R. Schumann's op.15.

Usage

```
data("SchumannKurioseGeschichte")
```

Format

the sampled signal in `numeric` class.

Source

ISMPL - V. Horowitz (piano).

Examples

```
data(SchumannKurioseGeschichte)
```

ShepardTone — *Shepard Tone*

Description

This function generates a Shepard tone.

Usage

```
ShepardTone(inMainFreq, inDuration = 1, inSampleFreq = 22050,
            inPhaseFlag = 1, indBLevel = NULL)
```

Arguments

`inMainFreq`	the main frequency (in Hz).
`inDuration`	the duration (in s). If empty or not specified, 1 is used by default.
`inSampleFreq`	the desired sample frequency for the output signal (in Hz). If empty or not specified, 22050 is used by default.
`inPhaseFlag`	for choosing whether random phase is to be used or not (1 to use random phase, 0 otherwise). If empty or not specified, 1 is used by default.
`indBLevel`	dB level of generated tone (in dB) if empty or not specified, no level adjustment is performed.

Value

`outSignal`, the signal for the tone in `numeric` class.

Note

To convert the output to "Wave" class use `tuneR::Wave` and `tuneR::normalize` functions.

Author(s)

Marc Vidal (R version). Based on the original code from **IPEM** Toolbox.

Examples

```
# Note D
D <- ShepardTone(293.66, indBLevel = -20)
```

`ShepardToneComplex` *Shepard Tone Complex*

Description

This function generates a tone complex built up of Shepard tones.

Usage

```
ShepardToneComplex(inToneVector, inDuration = 1, inSampleFreq = 22050,
                     inPhaseFlag = 1, indBLevel = NULL)
```

Arguments

`inToneVector`	a 12 elements vector representing the amplitude for each tone `"C","C#","D"`, ... in the tone complex (0 = no tone, 1 = full amplitude).
`inDuration`	the duration (in s). If empty or not specified, 1 is used by default.
`inSampleFreq`	the desired sample frequency for the output signal (in Hz) if empty or not specified, 22050 is used by default.
`inPhaseFlag`	for choosing whether random phase has to be used or not (1 to use random phase, 0 otherwise). If empty or not specified, 1 is used by default.
`indBLevel`	dB level of generated tone complex (in dB) if empty or not specified, no level adjustment is performed.

Value

`outSignal`, the signal for the tone complex in `numeric` class.

Author(s)

Marc Vidal (R version). Based on the original code from **IPEM** Toolbox.

Examples

```
# Chord C
C  <- ShepardToneComplex(c(1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0), indBLevel = -20)
```

Index