Package ‘rPraat’

April 4, 2020

Type Package
Title Interface to Praat
Version 1.3.1
Encoding UTF-8
Maintainer Tomas Boril <borilt@gmail.com>
URL https://github.com/bbTomas/rPraat/
BugReports https://github.com/bbTomas/rPraat/issues
License MIT + file LICENSE
LazyData TRUE
Depends R (>= 3.4.0)
Imports graphics (>= 3.1.0), dplyr (>= 0.8.5), stringr (>= 1.4.0), readr(>= 1.3.1), dygraphs (>= 1.1.1.6), tuneR (>= 1.3.3)
RoxygenNote 7.1.0
Suggests testthat
NeedsCompilation no
Author Tomas Boril [aut, cre]
Repository CRAN
Date/Publication 2020-04-04 17:10:02 UTC

R topics documented:

  as.formant ................................................................. 4
  as.it ................................................................. 4
  as.pitch ............................................................... 5
  as.pt ................................................................. 6
  as.snd ................................................................. 6
  as.tg ................................................................. 7
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>col.read</td>
</tr>
<tr>
<td>col.write</td>
</tr>
<tr>
<td>detectEncoding</td>
</tr>
<tr>
<td>formant.cut</td>
</tr>
<tr>
<td>formant.cut0</td>
</tr>
<tr>
<td>formant.plot</td>
</tr>
<tr>
<td>formant.read</td>
</tr>
<tr>
<td>formant.sample</td>
</tr>
<tr>
<td>formant.toArray</td>
</tr>
<tr>
<td>formant.toFrame</td>
</tr>
<tr>
<td>formant.write</td>
</tr>
<tr>
<td>ifft</td>
</tr>
<tr>
<td>isInt</td>
</tr>
<tr>
<td>isLogical</td>
</tr>
<tr>
<td>isNum</td>
</tr>
<tr>
<td>isString</td>
</tr>
<tr>
<td>it.cut</td>
</tr>
<tr>
<td>it.cut0</td>
</tr>
<tr>
<td>it.interpolate</td>
</tr>
<tr>
<td>it.legendre</td>
</tr>
<tr>
<td>it.legendreDemo</td>
</tr>
<tr>
<td>it.legendreSynth</td>
</tr>
<tr>
<td>it.plot</td>
</tr>
<tr>
<td>it.read</td>
</tr>
<tr>
<td>it.sample</td>
</tr>
<tr>
<td>it.write</td>
</tr>
<tr>
<td>pitch.cut</td>
</tr>
<tr>
<td>pitch.cut0</td>
</tr>
<tr>
<td>pitch.plot</td>
</tr>
<tr>
<td>pitch.read</td>
</tr>
<tr>
<td>pitch.sample</td>
</tr>
<tr>
<td>pitch.toArray</td>
</tr>
<tr>
<td>pitch.toFrame</td>
</tr>
<tr>
<td>pitch.write</td>
</tr>
<tr>
<td>pt.cut</td>
</tr>
<tr>
<td>pt.cut0</td>
</tr>
<tr>
<td>pt.Hz2ST</td>
</tr>
<tr>
<td>pt.interpolate</td>
</tr>
<tr>
<td>pt.legendre</td>
</tr>
<tr>
<td>pt.legendreDemo</td>
</tr>
<tr>
<td>pt.legendreSynth</td>
</tr>
<tr>
<td>pt.plot</td>
</tr>
<tr>
<td>pt.read</td>
</tr>
<tr>
<td>pt.sample</td>
</tr>
<tr>
<td>pt.write</td>
</tr>
<tr>
<td>round2</td>
</tr>
<tr>
<td>seqM</td>
</tr>
<tr>
<td>snd.cut</td>
</tr>
</tbody>
</table>
R topics documented:

snd.cut0 .......................................................... 46
snd.plot .......................................................... 47
snd.read .......................................................... 48
snd.sample ......................................................... 49
snd.write ........................................................ 50
str.Trim ........................................................... 51
str_contains ....................................................... 51
str_find ........................................................... 52
str_find1 ........................................................... 53
tg.boundaryMagnet ............................................... 53
tg.checkTierInd .................................................... 54
tg.countLabels ..................................................... 55
tg.createNewTextGrid .......................................... 56
tg.cut ............................................................... 57
tg.cut0 ............................................................. 58
tg.duplicateTier .................................................. 59
tg.duplicateTierMergeSegments ............................... 60
tg.findLabels ....................................................... 61
tg.getEndTime ..................................................... 62
tg.getIntervalDuration ......................................... 63
tg.getIntervalEndTime .......................................... 64
tg.getIntervalIndexAtTime .................................... 65
tg.getIntervalStartTime ........................................ 65
tg.getLabel .......................................................... 66
tg.getNumberOfIntervals ........................................ 67
tg.getNumberOfPoints ............................................ 67
tg.getNumberOfTiers ............................................. 68
tg.getPointIndexHigherThanTime ............................. 69
tg.getPointIndexLowerThanTime ............................... 69
tg.getPointIndexNearestTime .................................. 70
tg.getPointTime .................................................... 71
tg.getStartInterval .............................................. 71
tg.getTierName .................................................... 72
tg.getTotalDuration ............................................. 73
tg.insertBoundary ................................................. 73
tg.insertInterval .................................................. 75
tg.insertNewIntervalTier ....................................... 76
tg.insertNewPointTier ............................................ 77
tg.insertPoint ..................................................... 78
tg.isIntervalTier .................................................. 79
tg.isPointTier ..................................................... 80
tg.plot .............................................................. 80
tg.read ............................................................. 82
tg.removeIntervalBothBoundaries ............................. 82
tg.removeIntervalLeftBoundary ................................ 84
tg.removeIntervalRightBoundary ............................... 85
tg.removePoint ..................................................... 86
tg.removeTier ....................................................... 86
<table>
<thead>
<tr>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Renames the <code>class(t)[&quot;name&quot;]</code> attribute and sets <code>class(t)[&quot;type&quot;] &lt;- &quot;IntensityTier&quot;</code> (if it is not already set)</td>
<td><code>as.it(it, name = &quot;&quot;)</code></td>
</tr>
<tr>
<td>Renames the <code>class(formant)[&quot;name&quot;]</code> attribute and sets <code>class(formant)[&quot;type&quot;] &lt;- &quot;Formant 2&quot;</code> (if it is not already set)</td>
<td><code>as.formant(formant, name = &quot;&quot;)</code></td>
</tr>
</tbody>
</table>

**Examples**

```r
class(formant.sample())
class(as.formant(formant.sample(), name = "New Name"))
```
as.pitch

Arguments

- `it` : IntensityTier object
- `name` : New name

Value

IntensityTier object

Examples

class(it.sample())
class(as.it(it.sample(), name = "New Name"))

description

Renames the class(pitch)["name"] attribute and sets class(pitch)["type"] <-"Pitch 1" (if it is not already set)

Usage

as.pitch(pitch, name = "")

Arguments

- `pitch` : Pitch 1 object
- `name` : New name

Value

Pitch 1 object

Examples

class(pitch.sample())
class(as.pitch(pitch.sample(), name = "New Name"))
Description

Renames the class(pt)["name"] attribute and sets class(pt)["type"] <-"PitchTier" (if it is not already set)

Usage

as.pt(pt, name = "")

Arguments

pt PitchTier object
name New name

Value

PitchTier object

Examples

class(pt.sample())
class(as.pt(pt.sample(), name = "New Name"))

Description

Renames the class(snd)["name"] attribute and sets class(snd)["type"] <-"Sound" (if it is not already set)

Usage

as.snd(snd, name = "")

Arguments

snd snd object
name New name
Details

At least, $\text{sig}$ and $\text{fs}$ members must be present in $\text{snd}$ list.
If not present, it calculates $\text{t}$, $\text{nChannels}$, $\text{nBits}$ (default: 16), $\text{nSamples}$, and $\text{duration}$ members of $\text{snd}$ list.

Value

$\text{snd}$ object

Examples

class(snd.sample())
class(as.snd(snd.sample(), name = "New Name"))

Description

Renames the $\text{class(tg)["name"]}$ attribute and sets $\text{class(tg)["type"]} \leftarrow \"\text{TextGrid}\"$ (if it is not already set)

Usage

as.tg(tg, name = "")

Arguments

tg TextGrid object
name New name

Value

TextGrid object

Examples

class(tg.sample())
class(as.tg(tg.sample(), name = "New Name"))
col.read

Description

Loads Collection from Praat in Text or Short text format. Collection may contain combination of TextGrids, PitchTiers, Pitch objects, Formant objects, and IntensityTiers.

Usage

col.read(fileName, encoding = "UTF-8")

Arguments

- fileName: Input file name
- encoding: File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

Collection object

See Also

tg.read, pt.read, pitch.read, formant.read, it.read

Examples

```r
## Not run:
coll <- col.read("coll_text.Collection")
length(coll)  # number of objects in collection
class(coll[[1]])["type"]  # 1st object type
class(coll[[1]])["name"]  # 1st object name
it <- coll[[1]]  # 1st object
it.plot(it)

class(coll[[2]])["type"]  # 2nd object type
class(coll[[2]])["name"]  # 2nd object name
tg <- coll[[2]]  # 2nd object
tg.plot(tg)
length(tg)  # number of tiers in TextGrid
tg$word$label

class(coll[[3]])["type"]  # 3rd object type
class(coll[[3]])["name"]  # 3rd object type
pitch <- coll[[3]]  # 3rd object
names(pitch)
pitch$n  # number of frames
pitch$t[4]  # time instance of the 4th frame
pitch$frame[[4]]  # 4th frame: pitch candidates
```
Description

Saves Collection of objects to a file (in UTF-8 encoding). col is list of objects, each item col[[i]] must contain class(col[[i]])["type"] ("TextGrid", "PitchTier", "IntensityTier", "Pitch 1", or "Formant 2") and class(col[[i]])["name"] (name of the object) parameters set. These parameters can be created easily using "as.something()" functions: as.tg(), as.pt(), as.it(), as.pitch(), as.formant()

Usage

```r
col.write(col, fileNameCollection, format = "short")
```

Arguments

- `col` Collection object = list of objects (col[[1]], col[[2]], etc.) with class(col[[i]])["type"] and class(col[[i]])["name"] parameters set
- `fileNameCollection` file name to be created
- `format` Output file format ("short" (short text format) or "text" (a.k.a. full text format))

Details

Sound objects in col.read() and col.write() are not supported at this moment because they would occupy too much disc space in text format.

See Also

- col.read
Examples

```r
col <- list(as.tg(tg.sample(), "My textgrid"), as.pt(pt.sample(), "My PitchTier 1"),
            as.pt(pt.Hz2ST(pt.sample()), "My PitchTier 2"), as.it(it.sample(), "My IntensityTier"),
            as.pitch(pitch.sample(), "My Pitch"), as.formant(formant.sample(), "My Formant"))
col.write(col, "my_collection.Collection")
```

## End(Not run)

---

`detectEncoding` `detectEncoding`

## Description

Detects unicode encoding of Praat text files

## Usage

`detectEncoding(fileName)`

## Arguments

`fileName`  
Input file name

## Value

detected encoding of the text input file

## Examples

```r
detectEncoding("demo/H.TextGrid")
detectEncoding("demo/H_UTF16.TextGrid")
```

## End(Not run)

---

`formant.cut` `formant.cut`

## Description

Cut the specified interval from the Formant object and preserve time

## Usage

`formant.cut(formant, tStart = -Inf, tEnd = Inf)`
formant.cut0

Arguments

formant  Formant object (either in Frame or Array format)
tStart    beginning time of interval to be cut (default -Inf = cut from the xmin of the Formant)
tEnd      final time of interval to be cut (default Inf = cut to the xmax of the Formant)

Value

Formant object

See Also

formant.cut0, tg.cut, tg.cut0, formant.read, formant.plot

Examples

formant <- formant.sample()
formant2 <- formant.cut(formant, tStart = 3)
formant2_0 <- formant.cut0(formant, tStart = 3)
formant3 <- formant.cut(formant, tStart = 2, tEnd = 3)
formant3_0 <- formant.cut0(formant, tStart = 2, tEnd = 3)
formant4 <- formant.cut(formant, tEnd = 1)
formant4_0 <- formant.cut0(formant, tEnd = 1)
formant5 <- formant.cut(formant, tStart = -1, tEnd = 1)
formant5_0 <- formant.cut0(formant, tStart = -1, tEnd = 1)
## Not run:
formant.plot(formant)
formant.plot(formant2)
formant.plot(formant2_0)
formant.plot(formant3)
formant.plot(formant3_0)
formant.plot(formant4)
formant.plot(formant4_0)
formant.plot(formant5)
formant.plot(formant5_0)
## End(Not run)

Description

Cut the specified interval from the Formant object and shift time so that the new xmin = 0

Usage

formant.cut0(formant, tStart = -Inf, tEnd = Inf)
formant.plot

Arguments

formant Formant object (either in Frame or Array format)
tStart beginning time of interval to be cut (default -Inf = cut from the xmin of the Formant)
tEnd final time of interval to be cut (default Inf = cut to the xmax of the Formant)

Value

Formant object

See Also

formant.cut, tg.cut, tg.cut0, formant.read, formant.plot

Examples

formant <- formant.sample()
formant2 <- formant.cut(formant, tStart = 3)
formant2_0 <- formant.cut0(formant, tStart = 3)
formant3 <- formant.cut(formant, tStart = 2, tEnd = 3)
formant3_0 <- formant.cut0(formant, tStart = 2, tEnd = 3)
formant4 <- formant.cut(formant, tEnd = 1)
formant4_0 <- formant.cut0(formant, tEnd = 1)
formant5 <- formant.cut(formant, tStart = -1, tEnd = 1)
formant5_0 <- formant.cut0(formant, tStart = -1, tEnd = 1)
## Not run:
formant.plot(formant)
formant.plot(formant2)
formant.plot(formant2_0)
formant.plot(formant3)
formant.plot(formant3_0)
formant.plot(formant4)
formant.plot(formant4_0)
formant.plot(formant5)
formant.plot(formant5_0)

## End(Not run)

Description

Plots interactive Formant object using dygraphs package.

Usage

formant.plot(formant, scaleIntensity = TRUE, drawBandwidth = TRUE, group = "")
formant.read

Arguments

formant           Formant object
scaleIntensity    Point size scaled according to relative intensity
drawBandwidth     Draw formant bandwidth
group             [optional] character string, name of group for dygraphs synchronization

See Also

formant.read, formant.sample, formant.toArray, tg.plot

Examples

## Not run:
formant <- formant.sample()
formant.plot(formant, drawBandwidth = TRUE)

## End(Not run)

formant.read formant.read

Description


Usage

formant.read(fileNameFormant, encoding = "UTF-8")

Arguments

fileNameFormant file name of Formant object
encoding        File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

A Formant object represents formants as a function of time.
f$xmin ... start time (seconds)
fxmax ... end time (seconds)
f$nx ... number of frames
f$dx ... time step = frame duration (seconds)
f$x1 ... time associated with the first frame (seconds)
formant.sample

f$t \ldots$ vector of time instances associated with all frames
f$maxnFormants \ldots$ maximum number of formants in frame
f$frame[[1]]$ to f$frame[[f$nx]] \ldots$ frames
f$frame[[1]]$intensity \ldots$ intensity of the frame
f$frame[[1]]$nFormants \ldots$ actual number of formants in this frame
f$frame[[1]]$frequency \ldots$ vector of formant frequencies (in Hz)
f$frame[[1]]$bandwidth \ldots$ vector of formant bandwidths (in Hz)

See Also

pitch.read, pt.read, tg.read, it.read, col.read

Examples

```r
## Not run:
f <- formant.read("demo/maminka.Formant")
names(f)
f$nx
f$t[4] \# time instance of the 4th frame
## End(Not run)
```

formant.sample

Description

Returns sample Formant object.

Usage

formant.sample()

Value

Formant

See Also

tg.sample, pt.sample, it.sample, pitch.sample

Examples

```r
formant <- formant.sample()
```
**formant.toArray**

**Description**

formant.toArray

**Usage**

formant.toArray(formant)

**Arguments**

formant  Formant object

**Value**

Formant object with frames converted to frequency and bandwidth arrays and intensity vector

**See Also**

formant.read, formant.plot

**Examples**

```r
formantArray <- formant.toArray(formant.sample())
formantArray$t[1:10]
formantArray$frequencyArray[, 1:10]
formantArray$bandwidthArray[, 1:10]
formantArray$intensityVector[1:10]
## Not run:
plot(formantArray$t, formantArray$frequencyArray[, 1]) # draw 1st formant track
## End(Not run)
```

---

**formant.toFrame**

**Description**

formant.toFrame

**Usage**

formant.toFrame(formantArray)
formant.write

Arguments

formantArray  Formant object (array format)

Value

Formant object with frames

See Also

formant.toArray, formant.read, formant.plot

Examples

formantArray <- formant.toArray(formant.sample())
formant <- formant.toFrame(formantArray)

formant.write(formant, "demo_output.Formant")

Description

Saves Formant to the file.

Usage

formant.write(formant, fileNameFormant, format = "short")

Arguments

formant  Formant object
fileNameFormant  Output file name
format  Output file format ("short" (default, short text format) or "text" (a.k.a. full text format))

See Also

formant.read, tg.read

Examples

## Not run:
formant <- formant.sample()
formant.write(formant, "demo_output.Formant")

## End(Not run)
### ifft

**Description**
Inverse Fast Fourier Transform (discrete FT), Matlab-like behavior.

**Usage**
```matlab
ifft(sig)
```

**Arguments**
- `sig` input vector

**Details**
This really is the inverse of the fft function, so `ifft(fft(x)) == x`.

**Value**
- output vector of the same length as the input vector

**See Also**
- `fft, Re, Im, Mod, Conj`

**Examples**
```matlab
ifft(fft(1:5))
```

### isInt

**Description**
Returns `TRUE / FALSE` whether it is exactly 1 integer number (in fact, the class can be numeric but the number must be integer), non-missing.

**Usage**
```matlab
isInt(num)
```

**Arguments**
- `num` variable to be tested
Description

Returns TRUE / FALSE whether it is exactly 1 logical value, non-missing

Usage

isLogical(logical)

Arguments

logical variable to be tested

Value

TRUE / FALSE

See Also

isNum, isInt, isString
Examples

isLogical(TRUE)
isLogical(FALSE)
isLogical(1)
isLogical(0)
isLogical(2)
isLogical(NA)
isLogical(NaN)
isLogical(logical(0))

isNum

isNum

Description

Returns TRUE / FALSE whether it is exactly 1 number (numeric or integer vector of length 1, non-missing)

Usage

isNum(num)

Arguments

num variable to be tested

Value

TRUE / FALSE

See Also

isInt, isLogical, isString

Examples

isNum(2)
isNum(2L)
isNum(-2)
isNum(-2L)
isNum(2.1)
isNum(-2.1)
isNum(1:5)
isNum(NA_real_)
isNum(numeric(0))
isString

Description
Returns TRUE / FALSE whether it is exactly 1 character string (character vector of length 1, non-missing)

Usage
isString(string)

Arguments
string variable to be tested

Value
TRUE / FALSE

See Also
isInt, isNum, isLogical

Examples
isString("hello")
isString(2)
isString(c("hello", "world"))
isString(NA_character_)

it.cut

Description
Cut the specified interval from the IntensityTier and preserve time

Usage
it.cut(it, tStart = -Inf, tEnd = Inf)

Arguments
it IntensityTier object
tStart beginning time of interval to be cut (default -Inf = cut from the tmin of the IntensityTier)
tEnd final time of interval to be cut (default Inf = cut to the tmax of the IntensityTier)
it.cut0

Value

IntensityTier object

See Also

it.cut0, it.read, it.plot, it.interpolate, it.legendre, it.legendreSynth, it.legendreDemo

Examples

```r
it <- it.sample()
it2 <- it.cut(it, tStart = 0.3)
it2_0 <- it.cut0(it, tStart = 0.3)
it3 <- it.cut(it, tStart = 0.2, tEnd = 0.3)
it3_0 <- it.cut0(it, tStart = 0.2, tEnd = 0.3)
it4 <- it.cut(it, tEnd = 0.3)
it4_0 <- it.cut0(it, tEnd = 0.3)
it5 <- it.cut(it, tStart = -1, tEnd = 1)
it5_0 <- it.cut0(it, tStart = -1, tEnd = 1)
## Not run:
it.plot(it)
it.plot(it2)
it.plot(it2_0)
it.plot(it3)
it.plot(it3_0)
it.plot(it4)
it.plot(it4_0)
it.plot(it5)
it.plot(it5_0)
## End(Not run)
```

it.cut0

Description

Cut the specified interval from the IntensityTier and shift time so that the new tmin = 0

Usage

`it.cut0(it, tStart = -Inf, tEnd = Inf)`

Arguments

- **it**: IntensityTier object
- **tStart**: beginning time of interval to be cut (default -Inf = cut from the tmin of the IntensityTier)
- **tEnd**: final time of interval to be cut (default Inf = cut to the tmax of the IntensityTier)
**Value**

IntensityTier object

**See Also**

`it.cut, it.read, it.plot, it.interpolate, it.legendre, it.legendreSynth, it.legendreDemo`

**Examples**

```r
it <- it.sample()
it2 <- it.cut(it, tStart = 0.3)
it2_0 <- it.cut0(it, tStart = 0.3)
it3 <- it.cut(it, tStart = 0.2, tEnd = 0.3)
it3_0 <- it.cut0(it, tStart = 0.2, tEnd = 0.3)
it4 <- it.cut(it, tEnd = 0.3)
it4_0 <- it.cut0(it, tEnd = 0.3)
it5 <- it.cut(it, tStart = -1, tEnd = 1)
it5_0 <- it.cut0(it, tStart = -1, tEnd = 1)
## Not run:
it.plot(it)
# it.plot(it2)
# it.plot(it2_0)
# it.plot(it3)
# it.plot(it3_0)
# it.plot(it4)
# it.plot(it4_0)
# it.plot(it5)
# it.plot(it5_0)
## End(Not run)
```

---

**Description**

Interpolates IntensityTier contour in given time instances.

**Usage**

`it.interpolate(it, t)`

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>it</code></td>
<td>IntensityTier object</td>
</tr>
<tr>
<td><code>t</code></td>
<td>vector of time instances of interest</td>
</tr>
</tbody>
</table>
Details

a) If \( t < \min(it$t) \) (or \( t > \max(it$t) \)), returns the first (or the last) value of it$i. b) If \( t \) is existing point in it$t, returns the respective it$f. c) If \( t \) is between two existing points, returns linear interpolation of these two points.

Value

IntensityTier object

See Also

it.read, it.write, it.plot, it.cut, it.cut0, it.legendre

Examples

\[
\begin{align*}
\textit{it} & \leftarrow \textit{it.sample()} \\
\textit{it2} & \leftarrow \textit{it.interpolate(it, seq(it$t[1], it$t[length(it$t)], by = 0.001))}
\end{align*}
\]

\[
\begin{align*}
\# \text{ Not run:} \\
\textit{it.plot(it)} \\
\textit{it.plot(it2)}
\end{align*}
\]

\[
\begin{align*}
\# \text{ End(Not run)}
\end{align*}
\]

Description

Interpolate the IntensityTier in npoints equidistant points and approximate it by Legendre polynomials

Usage

\[
\text{it.legendre(it, npoints = 1000, npolynomials = 4)}
\]

Arguments

\[
\begin{align*}
\text{it} & \quad \text{IntensityTier object} \\
\text{npoints} & \quad \text{Number of points of IntensityTier interpolation} \\
\text{npolynomials} & \quad \text{Number of polynomials to be used for Legendre modelling}
\end{align*}
\]

Value

Vector of Legendre polynomials coefficients

See Also

it.legendreSynth, it.legendreDemo, it.cut, it.cut0, it.read, it.plot, it.interpolate
Examples

```r
it <- it.sample()
it <- it.cut(it, tStart = 0.2, tEnd = 0.4) # cut IntensityTier and preserve time
c <- it.legendre(it)
print(c)
leg <- it.legendreSynth(c)
itLeg <- it
itLeg$t <- seq(itLeg$tmin, itLeg$tmax, length.out = length(leg))
itLeg$i <- leg
## Not run:
plot(it$t, it$i, xlab = "Time (sec)", ylab = "Intensity (dB)"
lines(itLeg$t, itLeg$i, col = "blue")
## End(Not run)
```

Description

Plots first four Legendre polynomials

Usage

```r
it.legendreDemo()
```

See Also

`it.legendre, it.legendreSynth, it.read, it.plot, it.interpolate`

Examples

```r
## Not run:
it.legendreDemo()
## End(Not run)
```

Description

Synthetize the contour from vector of Legendre polynomials `c` in `npoints` equidistant points

Usage

```r
it.legendreSynth(c, npoints = 1000)
```
it.plot

Arguments

c Vector of Legendre polynomials coefficients
npoints Number of points of IntensityTier interpolation

Value
Vector of values of synthetized contour

See Also
it.legendre, it.legendreDemo, it.read, it.plot, it.interpolate

Examples

it <- it.sample()
it <- it.cut(it, tStart = 0.2, tEnd = 0.4) # cut IntensityTier and preserve time
c <- it.legendre(it)
print(c)
leg <- it.legendreSynth(c)
itLeg <- it
itLeg$t <- seq(itLeg$tmin, itLeg$tmax, length.out = length(leg))
itLeg$i <- leg
## Not run:
plot(it$t, it$i, xlab = "Time (sec)", ylab = "Intensity (dB)"
lines(itLeg$t, itLeg$i, col = "blue")
## End(Not run)

it.plot

Description
Plots interactive IntensityTier using dygraphs package.

Usage

it.plot(it, group = "", snd = NULL)

Arguments

it IntensityTier object

See Also
it.read, tg.plot, it.cut, it.cut0, it.interpolate, it.write
## Examples

```r
## Not run:
# it <- it.sample()
# it.plot(it)

## End(Not run)
```

---

### Description

Reads IntensityTier from Praat. Supported formats: text file, short text file.

### Usage

```r
it.read(fileNameIntensityTier, encoding = "UTF-8")
```

### Arguments

- `fileNameIntensityTier`: file name of IntensityTier
- `encoding`: File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

### Value

IntensityTier object

### See Also

- `it.write`, `it.plot`, `it.cut`, `it.cut0`, `it.interpolate`, `tg.read`, `pt.read`, `pitch.read`, `formant.read`, `col.read`

### Examples

```r
## Not run:
# it <- it.read("demo/maminka.IntensityTier")
# it.plot(it)

## End(Not run)
```
it.sample

Description
Returns sample IntensityTier.

Usage
it.sample()

Value
IntensityTier

See Also
it.plot

Examples
it <- it.sample()
it.plot(it)

it.write

Description
Saves IntensityTier to file (in UTF-8 encoding). it is list with at least $t$ and $i$ vectors (of the same length). If there are no $t_{min}$ and $t_{max}$ values, there are set as min and max of $t$ vector.

Usage
it.write(it, fileNameIntensityTier, format = "short")

Arguments

it
IntensityTier object

fileNameIntensityTier
file name to be created

format
Output file format ("short" (short text format - default), "text" (a.k.a. full text format))

See Also
it.read, tg.write, it.interpolate
Examples

```r
## Not run:
it <- it.sample()
it.plot(pt)
it.write(it, "demo/intensity.IntensityTier")

## End(Not run)
```

---

**pitch.cut**

### Description
Cut the specified interval from the Pitch object and preserve time

### Usage

```r
pitch.cut(pitch, tStart = -Inf, tEnd = Inf)
```

### Arguments

- **pitch**: Pitch object (either in Frame or Array format)
- **tStart**: beginning time of interval to be cut (default -Inf = cut from the xmin of the Pitch)
- **tEnd**: final time of interval to be cut (default Inf = cut to the xmax of the Pitch)

### Value
Pitch object

### See Also

`pitch.cut0`, `tg.cut`, `tg.cut0`, `pitch.read`, `pitch.plot`

### Examples

```r
pitch <- pitch.sample()
pitch2 <- pitch.cut(pitch, tStart = 3)
pitch2_0 <- pitch.cut0(pitch, tStart = 3)
pitch3 <- pitch.cut(pitch, tStart = 2, tEnd = 3)
pitch3_0 <- pitch.cut0(pitch, tStart = 2, tEnd = 3)
pitch4 <- pitch.cut(pitch, tEnd = 1)
pitch4_0 <- pitch.cut0(pitch, tEnd = 1)
pitch5 <- pitch.cut(pitch, tStart = -1, tEnd = 1)
pitch5_0 <- pitch.cut0(pitch, tStart = -1, tEnd = 1)
```

```r
## Not run:
pitch.plot(pitch)
pitch.plot(pitch2)
```
Description

Cut the specified interval from the Pitch object and shift time so that the new xmin = 0

Usage

pitch.cut0(pitch, tStart = -Inf, tEnd = Inf)

Arguments

pitch       Pitch object (either in Frame or Array format)
tStart      beginning time of interval to be cut (default -Inf = cut from the xmin of the Pitch)
tEnd        final time of interval to be cut (default Inf = cut to the xmax of the Pitch)

Value

Pitch object

See Also

pitch.cut, tg.cut, tg.cut0, pitch.read, pitch.plot

Examples

pitch <- pitch.sample()
pitch2 <- pitch.cut(pitch, tStart = 3)
pitch2_0 <- pitch.cut0(pitch, tStart = 3)
pitch3 <- pitch.cut(pitch, tStart = 2, tEnd = 3)
pitch3_0 <- pitch.cut0(pitch, tStart = 2, tEnd = 3)
pitch4 <- pitch.cut(pitch, tEnd = 1)
pitch4_0 <- pitch.cut0(pitch, tEnd = 1)
pitch5 <- pitch.cut(pitch, tStart = -1, tEnd = 1)
pitch5_0 <- pitch.cut0(pitch, tStart = -1, tEnd = 1)
## Not run:
pitch.plot(pitch)


## End(Not run)

### Description

Plots interactive Pitch object using dygraphs package.

### Usage

```r
pitch.plot(
  pitch,
  scaleIntensity = TRUE,
  showStrength = FALSE,
  group = "",
  pt = NULL
)
```

### Arguments

- **pitch**: Pitch object
- **scaleIntensity**: Point size scaled according to relative intensity
- **showStrength**: Show strength annotation
- **group** [optional]: character string, name of group for dygraphs synchronization
- **pt** [optional]: PitchTier object

### See Also

- `pitch.read`
- `pitch.sample`
- `pitch.toArray`
- `tg.plot`
- `pt.plot`
- `formant.plot`

### Examples

```r
## Not run:
pitch <- pitch.sample()
pitch.plot(pitch, scaleIntensity = TRUE, showStrength = TRUE)
pitch.plot(pitch, scaleIntensity = TRUE, showStrength = TRUE, pt = pt.sample())
## End(Not run)
```
**Description**


**Usage**

```r
pitch.read(fileNamePitch, encoding = "UTF-8")
```

**Arguments**

- `fileNamePitch`: file name of Pitch object
- `encoding`: File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

**Value**

A Pitch object represents periodicity candidates as a function of time.


- `p$xmin`: start time (seconds)
- `p$xmax`: end time (seconds)
- `p$nx`: number of frames
- `p$dx`: time step = frame duration (seconds)
- `p$x1`: time associated with the first frame (seconds)
- `p$t`: vector of time instances associated with all frames
- `p$ceiling`: a frequency above which a candidate is considered voiceless (Hz)
- `p$maxnCandidates`: maximum number of candidates in frame
- `p$frame[[1]]` to `p$frame[[p$nx]]`: frames
- `p$frame[[1]]$intensity`: intensity of the frame
- `p$frame[[1]]$nCandidates`: actual number of candidates in this frame
- `p$frame[[1]]$frequency`: vector of candidates’ frequency (in Hz)
  (for a voiced candidate), or 0 (for an unvoiced candidate)
- `p$frame[[1]]$strength`: vector of degrees of periodicity of candidates (between 0 and 1)

**See Also**

- `pt.read`, `tg.read`, `it.read`, `col.read`
Examples

## Not run:
```r
p <- pitch.read('demo/sound.Pitch')
names(p)
p$nx
p$t[4]       # time instance of the 4th frame
p$frame[[4]]  # 4th frame: pitch candidates
p$frame[[4]]$frequency[2]
p$frame[[4]]$strength[2]

## End(Not run)
```

Description

Returns sample Pitch object.

Usage

```r
pitch.sample()
```

Value

Pitch

See Also

`tg.sample`, `pt.sample`, `it.sample`, `formant.sample`

Examples

```r
pitch <- pitch.sample()
```

Description

`pitch.toArray`

Usage

```r
pitch.toArray(pitch)
```
pitch.toFrame

**Arguments**

- `pitchArray`  
  Pitch object (array format)

**Value**

Pitch object with frames converted to frequency and strength arrays and intensity vector

**See Also**

`pitch.toFrame`, `pitch.read`, `pitch.plot`

**Examples**

```r
pitchArray <- pitch.toArray(pitch.sample())
pitchArray$t[1:10]
pitchArray$frequencyArray[, 1:10]
pitchArray$bandwidthArray[, 1:10]
pitchArray$intensityVector[1:10]
```

---

**Description**

`pitch.toFrame`

**Usage**

`pitch.toFrame(pitchArray)`

**Arguments**

- `pitchArray`  
  Pitch object (array format)

**Value**

Pitch object with frames

**See Also**

`pitch.toArray`, `pitch.read`, `pitch.plot`

**Examples**

```r
pitchArray <- pitch.toArray(pitch.sample())
pitch <- pitch.toFrame(pitchArray)
```
pitch.write

Description

Saves Pitch to the file.

Usage

pitch.write(pitch, fileNamePitch, format = "short")

Arguments

- pitch: Pitch object
- fileNamePitch: Output file name
- format: Output file format ("short" (default, short text format) or "text" (a.k.a. full text format))

See Also

pitch.read, pt.read

Examples

```r
## Not run:
pitch <- pitch.sample()
pitch.write(pitch, "demo_output.Pitch")
## End(Not run)
```

pt.cut

Description

Cut the specified interval from the PitchTier and preserve time

Usage

pt.cut(pt, tStart = -Inf, tEnd = Inf)

Arguments

- pt: PitchTier object
- tStart: beginning time of interval to be cut (default -Inf = cut from the tmin of the PitchTier)
- tEnd: final time of interval to be cut (default Inf = cut to the tmax of the PitchTier)
pt.cut0

Value
PitchTier object

See Also

Examples

```r
pt <- pt.sample()
pt2 <- pt.cut(pt, tStart = 3)
pt2_0 <- pt.cut0(pt, tStart = 3)
pt3 <- pt.cut(pt, tStart = 2, tEnd = 3)
pt3_0 <- pt.cut0(pt, tStart = 2, tEnd = 3)
pt4 <- pt.cut(pt, tEnd = 1)
pt4_0 <- pt.cut0(pt, tEnd = 1)
pt5 <- pt.cut(pt, tStart = -1, tEnd = 1)
pt5_0 <- pt.cut0(pt, tStart = -1, tEnd = 1)

## Not run:
pt.plot(pt)
pt.plot(pt2)
pt.plot(pt2_0)
pt.plot(pt3)
pt.plot(pt3_0)
pt.plot(pt4)
pt.plot(pt4_0)
pt.plot(pt5)
pt.plot(pt5_0)

## End(Not run)
```

Description
Cut the specified interval from the PitchTier and shift time so that the new tmin = 0

Usage

```r
pt.cut0(pt, tStart = -Inf, tEnd = Inf)
```

Arguments

- **pt**: PitchTier object
- **tStart**: beginning time of interval to be cut (default -Inf = cut from the tmin of the PitchTier)
- **tEnd**: final time of interval to be cut (default Inf = cut to the tmax of the PitchTier)
**Value**

PitchTier object

**See Also**


**Examples**

```r
pt <- pt.sample()
pt2 <- pt.cut(pt, tStart = 3)
pt2_0 <- pt.cut0(pt, tStart = 3)
pt3 <- pt.cut(pt, tStart = 2, tEnd = 3)
pt3_0 <- pt.cut0(pt, tStart = 2, tEnd = 3)
pt4 <- pt.cut(pt, tEnd = 1)
pt4_0 <- pt.cut0(pt, tEnd = 1)
pt5 <- pt.cut(pt, tStart = -1, tEnd = 1)
pt5_0 <- pt.cut0(pt, tStart = -1, tEnd = 1)
```

Not run:

```r
pt.plot(pt)
pt.plot(pt2)
pt.plot(pt2_0)
pt.plot(pt3)
pt.plot(pt3_0)
pt.plot(pt4)
pt.plot(pt4_0)
pt.plot(pt5)
pt.plot(pt5_0)
```

End(Not run)

---

**pt.Hz2ST**

**pt.Hz2ST**

**Description**

Converts Hz to Semitones with given reference (default 0 ST = 100 Hz).

**Usage**

```r
pt.Hz2ST(pt, ref = 100)
```

**Arguments**

- **pt**  
  PitchTier object

- **ref**  
  reference value (in Hz) for 0 ST. Default: 100 Hz.
pt.interpolate

Value

PitchTier object

See Also


Examples

pt <- pt.sample()
pt2 <- pt.Hz2ST(pt, ref = 200)
## Not run:
pt.plot(pt) %>% dygraphs::dyAxis(“y”, label = “Frequency (Hz)”)  
pt.plot(pt2) %>% dygraphs::dyAxis(“y”, label = “Frequency (ST re 200 Hz)”)  
## End(Not run)

Description

Interpolates PitchTier contour in given time instances.

Usage

pt.interpolate(pt, t)

Arguments

pt  PitchTier object

 t  vector of time instances of interest

Details

a) If t < min(pt$t) (or t > max(pt$t)), returns the first (or the last) value of pt$f.  
b) If t is an existing point in pt$t, returns the respective pt$f.  
c) If t is between two existing points, returns linear interpolation of these two points.

Value

PitchTier object

See Also

Examples

```r
pt <- pt.sample()
pt <- pt.Hz2ST(pt, ref = 100)  # conversion of Hz to Semitones, reference 0 ST = 100 Hz.
pt2 <- pt.interpolate(pt, seq(pt$t[1], pt$t[length(pt$t)], by = 0.001))
## Not run:
pt.plot(pt)
pt.plot(pt2)
## End(Not run)
```

Description

Interpolate the PitchTier in `npoints` equidistant points and approximate it by Legendre polynomials

Usage

```r
pt.legendre(pt, npoints = 1000, npolynomials = 4)
```

Arguments

- **pt**: PitchTier object
- **npoints**: Number of points of PitchTier interpolation
- **npolynomials**: Number of polynomials to be used for Legendre modelling

Value

Vector of Legendre polynomials coefficients

See Also


Examples

```r
pt <- pt.sample()
pt <- pt.Hz2ST(pt)
pt <- pt.cut(pt, tStart = 3)  # cut PitchTier from t = 3 sec and preserve time
c <- pt.legendre(pt)
print(c)
leg <- pt.legendreSynth(c)
ptLeg <- pt
ptLeg$t <- seq(ptLeg$tmin, ptLeg$tmax, length.out = length(leg))
ptLeg$f <- leg
## Not run:
plot(pt$t, pt$f, xlab = "Time (sec)", ylab = "F0 (ST re 100 Hz)"
```
lines(ptLeg$t, ptLeg$f, col = "blue")

## End(Not run)

```r
pt.legendreDemo()
```

## Not run:

```r
pt.legendreDemo()
```

## End(Not run)

---

### Description

Plots first four Legendre polynomials

### Usage

```r
pt.legendreDemo()
```

### See Also


### Examples

```r
## Not run:
pt.legendreDemo()
## End(Not run)
```

---

### Description

Synthetize the contour from vector of Legendre polynomials \( c \) in \( npoints \) equidistant points

### Usage

```r
pt.legendreSynth(c, npoints = 1000)
```

### Arguments

- \( c \) Vector of Legendre polynomials coefficients
- \( npoints \) Number of points of PitchTier interpolation

### Value

Vector of values of synthetized contour
See Also


Examples

```r
pt <- pt.sample()
p <- pt.Hz2ST(pt)
pt <- pt.cut(pt, tStart = 3)  # cut PitchTier from t = 3 sec and preserve time
c <- pt.legendre(pt)
print(c)
leg <- pt.legendreSynth(c)
ptLeg <- pt
ptLeg$t <- seq(ptLeg$tmin, ptLeg$tmax, length.out = length(leg))
ptLeg$f <- leg
## Not run:
plot(pt$t, pt$f, xlab = "Time (sec)", ylab = "F0 (ST re 100 Hz)"
lines(ptLeg$t, ptLeg$f, col = "blue")
## End(Not run)
```

Description

Plots interactive PitchTier using dygraphs package.

Usage

```r
pt.plot(pt, group = "")
```

Arguments

- `pt` PitchTier object
- `group` [optional] character string, name of group for dygraphs synchronization

See Also


Examples

```r
## Not run:
pt <- pt.sample()
pt.plot()
## End(Not run)
```
Description

Reads PitchTier from Praat. Supported formats: text file, short text file, spreadsheet, headerless spreadsheet (headerless not recommended, it does not contain tmin and tmax info).

Usage

pt.read(fileNamePitchTier, encoding = "UTF-8")

Arguments

fileNamePitchTier
  file name of PitchTier

encoding
  File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

PitchTier object

See Also


Examples

## Not run:
pt <- pt.read("demo/H.PitchTier")
pt.plot(pt)
## End(Not run)

Description

Returns sample PitchTier.

Usage

pt.sample()
pt.write

Value
PitchTier

See Also
pt.plot

Examples
pt <- pt.sample()
pt.plot(pt)

pt.write(fileNamePitchTier, format = "spreadsheet")

Arguments

pt PitchTier object
fileNamePitchTier file name to be created
format Output file format ("short" (short text format), "text" (a.k.a. full text format), "spreadsheet" (default), "headerless" (not recommended, it does not contain tmin and tmax info))

See Also
pt.read, tg.write, pt.Hz2ST, pt.interpolate

Examples

## Not run:
pt <- pt.sample()  
pt <- pt.Hz2ST(pt)  # conversion of Hz to Semitones, reference 0 ST = 100 Hz.
pt.plot(pt)
pt.write(pt, "demo/H_st.PitchTier")

## End(Not run)
round2

Description

Rounds a number to the specified order. Round half away from zero (this is the difference from built-in round function.)

Usage

round2(x, order = 0)

Arguments

x  number to be rounded
order  0 (default) = units, -1 = 0.1, +1 = 10

Value

rounded number to the specified order

See Also

round, trunc, ceiling, floor

Examples

round2(23.5)  # = 24, compare: round(23.5) = 24
round2(23.4)  # = 23
round2(24.5)  # = 25, compare: round(24.5) = 24
round2(-23.5) # = -24, compare: round(-23.5) = -24
round2(-23.4) # = -23
round2(-24.5) # = -25, compare: round(-24.5) = -24
round2(123.456, -1)  # 123.5
round2(123.456, -2)  # 123.46
round2(123.456, 1)   # 120
round2(123.456, 2)   # 100
round2(123.456, 3)   # 0
round2(-123.456, -1) # -123.5
round2(-123.456, -2) # -123.46
round2(-123.456, 1)  # -120
round2(-123.456, 2)  # -100
round2(-123.456, 3)  # 0
Description

Matlab-like behaviour of colon operator or linspace for creating sequences, for-loop friendly.

Usage

seqM(from = NA, to = NA, by = NA, length.out = NA)

Arguments

from starting value of the sequence (the first number)
to end value of the sequence (the last number or the boundary number)
by increment of the sequence (if specified, do not use the length.out parameter). If both by and length.out are not specified, then by = +1.
length.out desired length of the sequence (if specified, do not use the by parameter)

Details

Like seq() but with Matlab-like behavior ([: operator] with by or [linspace] with length.out).

If I create a for-loop, I would like to get an empty vector for 3:1 (I want a default step +1) and also an empty vector for seq(3, 1, by = 1) (not an error). This is solved by this seqM function.

Value

returns a vector of type "integer" or "double"

Comparison

R: seqM Matlab R: seq

| seqM(1, 3) | 1 2 3 | 1:3 | the same | the same |
| seqM(1, 3, by=.8) | [1] 1.0 1.8 2.6 | 1:8:3 | the same | the same |
| seqM(1, 3, by=5) | [1] 1 | 1:5:3 | the same | the same |
| seqM(3, 1) | integer(0) | 3:1 | the same | [1] 3 2 1 |
| seqM(3, 1, by=+1) | integer(0) | 3:1:1 | the same | Error: wrong 'by' |
| seqM(3, 1, by=-1) | [1] 3 2 1 | 3:-1:1 | the same | the same |
| seqM(3, 1, by=-3) | [1] 3 | 3:-3:1 | the same | the same |
| seqM(1, 3, len=5) | [1] 1.0 1.5 2.0 2.5 3.0 | linspace(1,3,5) | the same | the same |
| seqM(1, 3, len=3) | [1] 1 2 3 | linspace(1,3,3) | the same | the same |
| seqM(1, 3, len=2) | [1] 1 3 | linspace(1,3,2) | the same | the same |
| seqM(1, 3, len=1) | [1] 1 | linspace(1,3,1) | the same | [1] 1 |
| seqM(1, 3, len=0) | integer(0) + warning | linspace(1,3,0) | the same without warning | the same without warning |
| seqM(3, 1, len=3) | [1] 3 2 1 | linspace(3,1,3) | the same | the same |
See Also

round2, isNum, isInt, ifft.

Examples

seqM(1, 3)
seqM(1, 3, by=.8)
seqM(1, 3, by=5)
seqM(3, 1)
seqM(3, 1, by=+1)
seqM(3, 1, by=-1)
seqM(3, 1, by=-3)
seqM(1, 3, len=5)
seqM(1, 3, len=3)
seqM(1, 3, len=2)
seqM(1, 3, len=1)
seqM(1, 3, len=0)
seqM(3, 1, len=3)

Description

Cut the specified interval from the Sound object and preserve time

Usage

snd.cut(snd, Start = -Inf, End = Inf, units = "seconds")

Arguments

snd  Sound object (list with $sig and $fs members at least)
Start beginning sample/time of interval to be cut (default -Inf = cut from the beginning of the Sound)
End  final sample/time of interval to be cut (default Inf = cut to the end of the Sound)
units Units of Start and End arguments: "samples" (starting from 1, i.e., 1 == index of the 1st sample) or "seconds" (starting from 0)

Value

Sound object
See Also

snd.cut0, tg.cut, tg.cut0, snd.read, snd.plot

Examples

```r
snd <- snd.sample()
snd2 <- snd.cut(snd, Start = 0.3)
snd2_0 <- snd.cut0(snd, Start = 0.3)
snd3 <- snd.cut(snd, Start = 0.2, End = 0.3)
snd3_0 <- snd.cut0(snd, Start = 0.2, End = 0.3)
snd4 <- snd.cut(snd, End = 0.1)
snd4_0 <- snd.cut0(snd, End = 0.1)
snd5 <- snd.cut(snd, Start = -0.1, End = 0.1)
snd5_0 <- snd.cut0(snd, Start = -0.1, End = 0.1)
snd6 <- snd.cut(snd, End = 1000, units = "samples")
snd6_0 <- snd.cut0(snd, End = 1000, units = "samples")
```

## Not run:

```r
snd.plot(snd)
snd.plot(snd2)
snd.plot(snd3)
snd.plot(snd4)
snd.plot(snd6)
```

## End(Not run)

### Description

Cut the specified interval from the Sound object and and shift time so that the new `snd$t[1] = 0`

### Usage

```r
snd.cut0(snd, Start = -Inf, End = Inf, units = "seconds")
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snd</td>
<td>Sound object (list with $sig and $fs members at least)</td>
</tr>
<tr>
<td>Start</td>
<td>beginning sample/time of interval to be cut (default -Inf = cut from the beginning of the Sound)</td>
</tr>
<tr>
<td>End</td>
<td>final sample/time of interval to be cut (default Inf = cut to the end of the Sound)</td>
</tr>
<tr>
<td>units</td>
<td>Units of Start and End arguments: &quot;samples&quot; (starting from 1, i.e., 1 == index of the 1st sample) or &quot;seconds&quot; (starting from 0)</td>
</tr>
</tbody>
</table>
Value

Sound object

See Also

snd.cut, tg.cut, tg.cut0, snd.read, snd.plot

Examples

```r
snd <- snd.sample()
snd2 <- snd.cut(snd, Start = 0.3)
snd2_0 <- snd.cut0(snd, Start = 0.3)
snd3 <- snd.cut(snd, Start = 0.2, End = 0.3)
snd3_0 <- snd.cut0(snd, Start = 0.2, End = 0.3)
snd4 <- snd.cut(snd, End = 0.1)
snd4_0 <- snd.cut0(snd, End = 0.1)
snd5 <- snd.cut(snd, Start = -0.1, End = 0.1)
snd5_0 <- snd.cut0(snd, Start = -0.1, End = 0.1)
snd6 <- snd.cut(snd, End = 1000, units = "samples")
snd6_0 <- snd.cut0(snd, End = 1000, units = "samples")

## Not run:
snd.plot(snd)
snd.plot(snd2)
snd.plot(snd2_0)
snd.plot(snd3)
snd.plot(snd3_0)
snd.plot(snd4)
snd.plot(snd4_0)
snd.plot(snd5)
snd.plot(snd5_0)
snd.plot(snd6)
snd.plot(snd6_0)

## End(Not run)
```

Description

Plots interactive Sound object using dygraphs package. If the sound is 2-channel (stereo), the 1st channel is plotted around mean value +1, the 2nd around mean value -1.

Usage

```r
snd.plot(snd, group = "", stemPlot = FALSE)
```
Arguments

- **snd**: Sound object (with $\text{sig}$ and $\text{fs}$ members at least)
- **group**: [optional] character string, name of group for dygraphs synchronization
- **stemPlot**: [optional] discrete style of plot using

See Also

- `snd.read`

Examples

```r
## Not run:
snd <- snd.sample()
snd.plot(snd)

snd.plot(list(sig = sin(seq(0, 2*pi, length.out = 4000)), fs = 8000))
## End(Not run)
```

Description

Loads sound file (.wav or .mp3) using tuneR package.

Usage

```r
snd.read(
  fileNameSound,
  fileType = "auto",
  from = 1,
  to = Inf,
  units = "samples"
)
```

Arguments

- **fileNameSound**: Sound file name (.wav or .mp3)
- **fileType**: "wav", "mp3" or "auto"
- **from**: Where to start reading in units (beginning "samples": 1, "seconds": 0)
- **to**: Where to stop reading in units (Inf = end of the file)
- **units**: Units of from and to arguments: "samples" (starting from 1) or "seconds" (starting from 0)
Value

Sound object with normalized amplitude (PCM / 2^(nbits-1) - 1) resulting to the range of [-1; +1]. In fact, the minimum value can be one quantization step lower (e.g. PCM 16bit: -32768).

_vector of discrete time instances (seconds) sig ... signal matrix (nrow(snd$sig) = number of samples, ncol(snd$sig) = number of channels, i.e., sig[,1] ... 1st channel) fs ... sample rate (Hz) nChannels ... number of signal channels (ncol(snd$sig)), 1 == mono, 2 == stereo nBits ... number of bits per one sample nSamples ... number of samples (nrow(snd$sig)) duration ... duration of signal (seconds), snd$duration == snd$nSamples/snd$fs

Examples

## Not run:
snd <- snd.read("demo/H.wav")
snd.plot(snd)

## End(Not run)

---

Description

Returns sample Sound object.

Usage

snd.sample()

Value

snd

See Also

snd.plot

Examples

snd <- snd.sample()
snd.plot(snd)
Description

Saves Sound object to a file. snd is a list with $sig and $fs members at least. If $nBits is not present, default value of 16 bits is used. Vector $t is ignored. If the sound signal is 2-channel (stereo), $sig must be a two-column matrix (1st column corresponds to the left channel, 2nd column to the right channel). If the sound is 1-channel (mono), $sig can be either a numeric vector or a one-column matrix. optional $t, $nChannels, $nSamples, $duration vectors are ignored.

Usage

snd.write(snd, fileNameSound)

Arguments

snd Sound object (with $sig, $nBits and $fs members)
fileNameSound file name to be created

See Also

snd.read

Examples

## Not run:
snd <- snd.sample()
snd.plot(snd)
snd.write(snd, "temp1.wav")
signal <- 0.8*sin(seq(0, 2*pi*440, length.out = 8000))
snd.write(list(sig = signal, fs = 8000, nBits = 16), "temp2.wav")
left <- 0.3*sin(seq(0, 2*pi*440, length.out = 4000))
right <- 0.5*sin(seq(0, 2*pi*220, length.out = 4000))
snd.write(list(sig = matrix(c(left, right), ncol = 2), fs = 8000, nBits = 16), "temp3.wav")

## End(Not run)
**strTrim**

Description

Trim leading and trailing whitespace in character string.

Usage

strTrim(string)

Arguments

string character string

Details

Like str_trim() in stringr package or trimws() in R 3.2.0 but way faster.


Value

returns a character string with removed leading and trailing whitespace characters.

See Also

isString for testing whether it is 1 character vector, str_contains for finding string in string without regexp, str_find for all indices without regexp, str_find1 for the first index without regexp.

Examples

strTrim("   Hello World!   ")

**str_contains**

Description

Find string in another string (without regular expressions), returns TRUE / FALSE.

Usage

str_contains(string, patternNoRegex)
Arguments

string string in which we try to find something
patternNoRegex string we want to find, "as it is" - no regular expressions

Value

TRUE / FALSE

See Also

str_find, str_find1, isString

Examples

str_contains("Hello world", "wor") # TRUE
str_contains("Hello world", "WOR") # FALSE
str_contains(tolower("Hello world"), tolower("wor")) # TRUE
str_contains("Hello world", "") # TRUE

str_find

Description

Find string in another string (without regular expressions), returns indices of all occurrences.

Usage

str_find(string, patternNoRegex)

Arguments

string string in which we try to find something
patternNoRegex string we want to find, "as it is" - no regular expressions

Value

indices of all occurrences (1 = 1st character)

See Also

str_find1, str_contains, isString

Examples

str_find("Hello, hello, hello world", "ell") # 2 9 16
str_find("Hello, hello, hello world", "q") # integer(0)
**str_find1**

**Description**

Find string in another string (without regular expressions), returns indices of the first occurrence only.

**Usage**

\begin{verbatim}
str_find1(string, patternNoRegex)
\end{verbatim}

**Arguments**

- **string** string in which we try to find something
- **patternNoRegex** string we want to find, "as it is" - no regular expressions

**Value**

index of the first occurrence only (1 = 1st character)

**See Also**

- `str_find`, `str_contains`, `isString`

**Examples**

\begin{verbatim}
str_find1("Hello, hello, hello world", "ell")  # 2
str_find1("Hello, hello, hello world", "q")   # integer(0)
\end{verbatim}

---

**tg.boundaryMagnet**

**Description**

Aligns boundaries of intervals in the target tier (typically: "word") to the closest boundaries in the pattern tier (typically: "phone"). If there is no boundary within the tolerance limit in the pattern tier, the boundary position in the target tier is kept at its original position.

**Usage**

\begin{verbatim}
tg.boundaryMagnet(
    tg,
    targetTier,
    patternTier,
    boundaryTolerance = Inf,
    verbose = TRUE
)
\end{verbatim}
tg.checkTierInd

Arguments

- **tg**: TextGrid object
- **targetTier**: index or "name" of the tier to be aligned
- **patternTier**: index or "name" of the pattern tier
- **boundaryTolerance**: if there is not any boundary in the pattern tier within this tolerance, the target boundary is kept at its position [default: Inf]
- **verbose**: if TRUE, every boundary shift is printed [default: TRUE]

Value

TextGrid object

See Also

- `tg.insertBoundary`
- `tg.insertInterval`
- `tg.duplicateTier`

Examples

```r
## Not run:
tg <- tg.sample()
tg <- tg.removeTier(tg, "phoneme")
tg <- tg.removeTier(tg, "syllable")
tg <- tg.removeTier(tg, "phrase")

# garble times in "word" tier a little
n <- length(tg$word$label)
deltaT <- runif(n - 1, min = -0.01, max = 0.015)
tg$word$t2[1: (n-1)] <- tg$word$t2[1: (n-1)] + deltaT
 tg$word$t1[2: n] <- tg$word$t2[1: (n-1)]
tg.plot(tg)

# align "word" tier according to "phone tier"
tg2 <- tg.boundaryMagnet(tg, targetTier = "word", patternTier = "phone")
tg.plot(tg2)

## End(Not run)
```

Description

Returns tier index. Input can be either index (number) or tier name (character string). It performs checks whether the tier exists.
Usage

tg.checkTierInd(tg, tierInd)

Arguments

tg
TextGrid object
tierInd
Tier index or "name"

Value

Tier index

See Also

tg.getTierName, tg.isIntervalTier, tg.isPointTier, tg.plot, tg.getNumberOfTiers

Examples

tg <- tg.sample()
tg.checkTierInd(tg, 4)
tg.checkTierInd(tg, "word")

tg.countLabels(tg, tierInd, label)

Arguments

tg
TextGrid object
tierInd
tier index or "name"
label
character string: label to be counted

Value

integer number

See Also

tg.findLabels, tg.getLabel
tg.createNewTextGrid

tg.createNewTextGrid
tg.createNewTextGrid

description

Creates new and empty TextGrid. tMin and tMax specify the total start and end time for the TextGrid. If a new interval tier is added later without specified start and end, they are set to TextGrid start and end.

Usage

tg.createNewTextGrid(tMin, tMax)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tMin</td>
<td>Start time of TextGrid</td>
</tr>
<tr>
<td>tMax</td>
<td>End time of TextGrid</td>
</tr>
</tbody>
</table>

Details

This empty TextGrid cannot be used for almost anything. At least one tier should be inserted using `tg.insertNewIntervalTier()` or `tg.insertNewPointTier()`.

Value

TextGrid object

See Also

tg.insertNewIntervalTier, tg.insertNewPointTier

Examples

tg <- tg.createNewTextGrid(0, 5)
tg <- tg.insertNewIntervalTier(tg, 1, "word")
tg <- tg.insertInterval(tg, "word", 1, 2, "hello")
tg.plot(tg)
Description

Cut the specified time frame from the TextGrid and preserve time.

Usage

tg.cut(tg, tStart = -Inf, tEnd = Inf)

Arguments

  * `tg`: TextGrid object
  * `tStart`: beginning time of time frame to be cut (default -Inf = cut from the tmin of the TextGrid)
  * `tEnd`: final time of time frame to be cut (default Inf = cut to the tmax of the TextGrid)

Value

TextGrid object

See Also

tg.cut0, pt.cut, pt.cut0, tg.read, tg.plot, tg.write, tg.insertInterval

Examples

tg <- tg.sample()
tg2 <- tg.cut(tg, tStart = 3)
tg2_0 <- tg.cut0(tg, tStart = 3)
tg3 <- tg.cut(tg, tStart = 2, tEnd = 3)
tg3_0 <- tg.cut0(tg, tStart = 2, tEnd = 3)
tg4 <- tg.cut(tg, tEnd = 1)
tg4_0 <- tg.cut0(tg, tEnd = 1)
tg5 <- tg.cut(tg, tStart = -1, tEnd = 5)
tg5_0 <- tg.cut0(tg, tStart = -1, tEnd = 5)
## Not run:
tg.plot(tg)
tg.plot(tg2)
tg.plot(tg2_0)
tg.plot(tg3)
tg.plot(tg3_0)
tg.plot(tg4)
tg.plot(tg4_0)
tg.plot(tg5)
tg.plot(tg5_0)
## End(Not run)
Description

Cut the specified time frame from the TextGrid and shift time so that the new tmin = 0

Usage

tg.cut0(tg, tStart = -Inf, tEnd = Inf)

Arguments

tg TextGrid object

tStart beginning time of time frame to be cut (default -Inf = cut from the tmin of the TextGrid)

tEnd final time of time frame to be cut (default Inf = cut to the tmax of the TextGrid)

Value

TextGrid object

See Also

tg.cut, pt.cut, pt.cut0, tg.read, tg.plot, tg.write, tg.insertInterval

Examples

tg <- tg.sample()
tg2 <- tg.cut(tg, tStart = 3)
tg2_0 <- tg.cut0(tg, tStart = 3)
tg3 <- tg.cut(tg, tStart = 2, tEnd = 3)
tg3_0 <- tg.cut0(tg, tStart = 2, tEnd = 3)
tg4 <- tg.cut(tg, tEnd = 1)
tg4_0 <- tg.cut0(tg, tEnd = 1)
tg5 <- tg.cut(tg, tStart = -1, tEnd = 5)
tg5_0 <- tg.cut0(tg, tStart = -1, tEnd = 5)
## Not run:
tg.plot(tg)
tg.plot(tg2)
tg.plot(tg2_0)
tg.plot(tg3)
tg.plot(tg3_0)
tg.plot(tg4)
tg.plot(tg4_0)
tg.plot(tg5)
tg.plot(tg5_0)
## End(Not run)
Description

Duplicates tier `originalInd` to new tier with specified index `newInd` (existing tiers are shifted). It is highly recommended to set a name to the new tier (this can also be done later by `tg.setTierName()`). Otherwise, both original and new tiers have the same name which is permitted but not recommended. In such a case, we cannot use the comfort of using tier name instead of its index in other functions.

Usage

```
tg.duplicateTier(tg, originalInd, newInd = Inf, newTierName = "")
```

Arguments

- `tg` TextGrid object
- `originalInd` tier index or "name"
- `newInd` new tier index (1 = the first, Inf = the last [default])
- `newTierName` [optional but recommended] name of the new tier

Value

TextGrid object

See Also

`tg.duplicateTierMergeSegments, tg.setTierName, tg.removeTier, tg.boundaryMagnet`

Examples

```
tg <- tg.sample()
tg2 <- tg.duplicateTier(tg, "word", 1, "NEW")
tg.plot(tg2)
```
Description

Duplicate tier originalInd and merge segments (according to the pattern) to the new tier with specified index newInd (existing tiers are shifted). Typical use: create new syllable tier from phone tier. It merges phones into syllables according to separators in pattern.

Usage

tg.duplicateTierMergeSegments(
    tg,
    originalInd,
    newInd = Inf,
    newTierName,
    pattern,
    sep = "-
"
)

Arguments

tg 
    TextGrid object
originalInd 
    tier index or "name"
newInd 
    new tier index (1 = the first, Inf = the last [default])
newTierName 
    name of the new tier
pattern 
    merge segments pattern for the new tier (e.g., "he-llo-world")
sep 
    separator in pattern (default: ":-
")

Details

Note 1: there can be segments with empty labels in the original tier (pause), do not specify them in the pattern

Note 2: if there is an segment with empty label in the original tier in the place of separator in the pattern, the empty segment is duplicated into the new tier, i.e. at the position of the separator, there may or may not be an empty segment, if there is, it is duplicated. And they are not specified in the pattern.

Note 3: if the segment with empty label is not at the position corresponding to separator, it leads to error - the part specified in the pattern between separators cannot be split by empty segments

Note 4: beware of labels that appear empty but they are not (space, new line character etc.) - these segments are handled as classical non-empty labels. See example - one label is " ", therefore it must be specified in the pattern.
tg.findLabels

Value
TextGrid object

See Also
tg.duplicateTier, tg.setTierName, tg.removeTier

Examples

tg <- tg.sample()
tg <- tg.removeTier(tg, "syllable")
collapsed <- paste0(tg$phone$label, collapse = "") # get actual labels
print(collapsed) # all labels in collapsed form - copy the string, include separators -> pattern
pattern <- "ja:-ci-P\ek-nu-t_so-?u-J\e-1a:S- -nej-dP\i:f-naj-deZ-h\ut_S-ku-?a-?a-ta-ma-na:"
tg2 <- tg.duplicateTierMergeSegments(tg, "phone", 1, "syll", pattern, sep = "-")
## Not run:
tg.plot(tg)
tg.plot(tg2)
## End(Not run)

Description
Find label or consecutive sequence of labels and returns their indices.

Usage
tg.findLabels(tg, tierInd, labelVector, returnTime = FALSE)

Arguments
tg TextGrid object
tierInd tier index or "name"
labelVector character string (one label) or vector of character strings (consecutive sequence of labels) to be found
returnTime If TRUE, return vectors of begin (t1) and end time (t2) for each found group of sequence of labels instead of indices (when FALSE = default).

Value
If returnTime == FALSE, returns list of all occurrences, each member of the list is one occurrence and contains vector of label indices, if returnTime == TRUE, returns list with vectors t1 (begin) and t2 (end) for each found group of sequence of labels.
tg.getEndTime

See Also
tg.countLabels, tg.getLabel, tg.duplicateTierMergeSegments

Examples

tg <- tg.sample()
i <- tg.findLabels(tg, "phoneme", "n")
i
length(i)
i[[1]]
i[[2]]
tg$phoneme$label[unlist(i)]
i <- tg.findLabels(tg, "phone", c("?", "a"))
i
length(i)
tg$phone$label[i[[1]]]
tg$phone$label[i[[2]]]
tg$phone$label[unlist(i)]

t <- tg.findLabels(tg, "phone", c("?", "a"), returnTime = TRUE)
t
$t2[1] - t$t1[1] # duration of the first result
$t2[2] - t$t1[2] # duration of the second result

i <- tg.findLabels(tg.sample(), "word", c("ti", "reknu", "co"))
i
length(i)
length(i[[1]])
i[[1]]
i[[1]][3]
tg$word$label[i[[1]]]

i <- tg.findLabels(tg.sample(), "word", c("ti", "reknu", "co"), returnTime = TRUE)
pt <- pt.sample()
tStart <- t$t1[1]
tEnd <- t$t2[1]
## Not run:
pt.plot(pt.cut(pt, tStart, tEnd))

## End(Not run)

tg.getEndTime
tg.getEndTime

Description

Returns end time. If tier index is specified, it returns end time of the tier, if it is not specified, it returns end time of the whole TextGrid.
Usage

tg.getEndTime(tg, tierInd = 0)

Arguments

tg TextGrid object
tierInd [optional] tier index or "name"

Value

numeric

See Also

tg.getStartTime, tg.getTotalDuration

Examples

tg <- tg.sample()
tg.getEndTime(tg)
tg.getEndTime(tg, "phone")

---

tg.getIntervalDuration

tg.getIntervalDuration

Description

Return duration (i.e., end - start time) of interval in interval tier.

Usage

tg.getIntervalDuration(tg, tierInd, index)

Arguments

tg TextGrid object
tierInd tier index or "name"
index index of interval

Value

numeric

See Also

tg.getIntervalStartTime, tg.getIntervalEndTime, tg.getIntervalIndexAtTime, tg.findLabels
Examples

tg <- tg.sample()
tg.getIntervalDuration(tg, "phone", 5)

tg.getIntervalEndTime(tg, tierInd, index)

Arguments

tg        TextGrid object
tierInd   tier index or "name"
index     index of interval

Value

numeric

See Also

tg.getIntervalStartTime, tg.getIntervalDuration, tg.getIntervalIndexAtTime, tg.findLabels

Examples

tg <- tg.sample()
tg.getIntervalEndTime(tg, "phone", 5)
Description

Returns index of interval which includes the given time, i.e. \( t_{\text{Start}} \leq \text{time} < t_{\text{End}} \). Tier index must belong to interval tier.

Usage

\[
tg.\text{getIntervalIndexAtTime}(tg, \text{tierInd}, \text{time})
\]

Arguments

- **tg**: TextGrid object
- **tierInd**: tier index or "name"
- **time**: time which is going to be found in intervals

Value

integer

See Also

- `tg.\text{getIntervalStartTime}`, `tg.\text{getIntervalEndTime}`, `tg.\text{getLabel}`, `tg.\text{findLabels}`

Examples

\[
tg \leftarrow tg.\text{sample}()
tg.\text{getIntervalIndexAtTime}(tg, \text{"word"}, 0.5)
\]

---

Description

Returns start time of interval in interval tier.

Usage

\[
tg.\text{getIntervalStartTime}(tg, \text{tierInd}, \text{index})
\]
Arguments

- `tg` (TextGrid object)
- `tierInd` (tier index or "name")
- `index` (index of interval)

Value

numeric

See Also

tg.getIntervalEndTime, tg.getIntervalDuration, tg.getIntervalIndexAtTime, tg.findLabels

Examples

tg <- tg.sample()
tg.getIntervalStartTime(tg, "phone", 5)

tg.getLabel(tg, "phoneme", 4)
tg.getLabel(tg, "phone", 4)

description

Return label of point or interval at the specified index.

Usage

tg.getLabel(tg, tierInd, index)

Arguments

- `tg` (TextGrid object)
- `tierInd` (tier index or "name")
- `index` (index of point or interval)

Value

character string

See Also

tg.setLabel, tg.countLabels, tg.findLabels

Examples

tg <- tg.sample()
tg.getLabel(tg, "phoneme", 4)
tg.getLabel(tg, "phone", 4)
tg.getNumberOfIntervals

**Description**

Returns number of intervals in the given interval tier.

**Usage**

```
tg.getNumberOfIntervals(tg, tierInd)
```

**Arguments**

- `tg` : TextGrid object
- `tierInd` : tier index or "name"

**Value**

integer

**See Also**

`tg.getNumberOfPoints`

**Examples**

```
tg <- tg.sample()
tg.getNumberOfIntervals(tg, "phone")
```

tg.getNumberOfPoints

**Description**

Returns number of points in the given point tier.

**Usage**

```
tg.getNumberOfPoints(tg, tierInd)
```

**Arguments**

- `tg` : TextGrid object
- `tierInd` : tier index or "name"
Value

integer

See Also

tg.getNumberofIntervals

Examples

tg <- tg.sample()
tg.getNumberofPoints(tg, "phoneme")

tg.getNumberofTiers
tg.getNumberofTiers

description

Returns number of tiers.

Usage

tg.getNumberofTiers(tg)

Arguments

tg TextGrid object

Value

integer

See Also

tg.getTierName, tg.isIntervalTier, tg.isPointTier

Examples

tg <- tg.sample()
tg.getNumberofTiers(tg)
Description

Returns index of point which is nearest the given time from right, i.e. \( \text{time} \leq \text{pointTime} \). Tier index must belong to point tier.

Usage

tg.getPointIndexHigherThanTime(tg, tierInd, time)

Arguments

- \( \text{tg} \): TextGrid object
- \( \text{tierInd} \): tier index or "name"
- \( \text{time} \): time which is going to be found in points

Value

integer

See Also

tg.getPointIndexNearestTime, tg.getPointIndexLowerThanTime, tg.getLabel, tg.findLabels

Examples

\[
\begin{align*}
\text{tg} & \leftarrow \text{tg.sample()} \\
\text{tg.getPointIndexHigherThanTime}(\text{tg}, \text{"phoneme"}, 0.5)
\end{align*}
\]

Description

Returns index of point which is nearest the given time from left, i.e. \( \text{pointTime} \leq \text{time} \). Tier index must belong to point tier.

Usage

tg.getPointIndexLowerThanTime(tg, tierInd, time)
Arguments

tg
TextGrid object
tierInd
tier index or "name"
time
time which is going to be found in points

Value

integer

See Also

tg.getPointIndexNearestTime, tg.getPointIndexHigherThanTime, tg.getLabel, tg.findLabels

Examples

tg <- tg.sample()
tg.getPointIndexLowerThanTime(tg, "phoneme", 0.5)

tg.getPointIndexNearestTime(tg, "phoneme", 0.5)

describe

Returns index of point which is nearest the given time (from both sides). Tier index must belong to point tier.

Usage

tg.getPointIndexNearestTime(tg, tierInd, time)

Arguments

tg
TextGrid object
tierInd
tier index or "name"
time
time which is going to be found in points

Value

integer

See Also

tg.getPointIndexLowerThanTime, tg.getPointIndexHigherThanTime, tg.getLabel, tg.findLabels

Examples

tg <- tg.sample()
tg.getPointIndexNearestTime(tg, "phoneme", 0.5)
tg.getPointTime

Description
Return time of point at the specified index in point tier.

Usage
tg.getPointTime(tg, tierInd, index)

Arguments
- **tg**: TextGrid object
- **tierInd**: tier index or "name"
- **index**: index of point

Value
numeric

See Also
tg.getLabel, tg.getPointIndexNearestTime, tg.getPointIndexLowerThanTime, tg.getPointIndexHigherThanTime, tg.findLabels

Examples
tg <- tg.sample()
tg.getPointTime(tg, "phoneme", 4)

tg.getStartTime

Description
Returns start time. If tier index is specified, it returns start time of the tier, if it is not specified, it returns start time of the whole TextGrid.

Usage
tg.getStartTime(tg, tierInd = 0)
tg.getTierName

Arguments

tg
TextGrid object
tierInd
[optional] tier index or "name"

Value

numeric

See Also

tg.getEndTime, tg.getTotalDuration

Examples

tg <- tg.sample()
tg.getStartTime(tg)
tg.getStartTime(tg, "phone")

tg.getTierName(tg, 2)

tg.getTierName(tg.getTierName)
Description

Returns total duration. If tier index is specified, it returns duration of the tier, if it is not specified, it returns total duration of the TextGrid.

Usage

tg.getTotalDuration(tg, tierInd = 0)

Arguments

tg TextGrid object
tierInd [optional] tier index or "name"

Value

numeric

See Also

tg.getStartTime, tg.getEndTime

Examples

tg <- tg.sample()
tg.getTotalDuration(tg)
tg.getTotalDuration(tg, "phone")

Description

Inserts new boundary into interval tier. This creates a new interval, to which we can set the label (optional argument).

Usage

tg.insertBoundary(tg, tierInd, time, label = "")
Arguments

- **tg**: TextGrid object
- **tierInd**: tier index or "name"
- **time**: time of the new boundary
- **label**: [optional] label of the new interval

Details

There are more possible situations which influence where the new label will be set.

a) New boundary into the existing interval (the most common situation): The interval is split into two parts. The left preserves the label of the original interval, the right is set to the new (optional) label.

b) On the left of existing interval (i.e., enlarging the tier size): The new interval starts with the new boundary and ends at the start of originally first existing interval. The label is set to the new interval.

c) On the right of existing interval (i.e., enlarging the tier size): The new interval starts at the end of originally last existing interval and ends with the new boundary. The label is set to the new interval. This is somewhat different behavior than in a) and b) where the new label is set to the interval which is on the right of the new boundary. In c), the new label is set on the left of the new boundary. But this is the only logical possibility.

It is a nonsense to insert a boundary between existing intervals to a position where there is no interval. This is against the basic logic of Praat interval tiers where, at the beginning, there is one large empty interval from beginning to the end. And then, it is divided to smaller intervals by adding new boundaries. Nevertheless, if the TextGrid is created by external programmes, you may rarely find such discontinuities. In such a case, at first, use the `tgRepairContinuity()` function.

Value

TextGrid object

See Also

- `tg.insertInterval`, `tg.removeIntervalLeftBoundary`, `tg.removeIntervalRightBoundary`, `tg.removeIntervalBothBoundaries`, `tg.boundaryMagnet`, `tg.duplicateTierMergeSegments`

Examples

```r
tg <- tg.sample()
tg2 <- tg.insertNewIntervalTier(tg, 1, "INTERVALS")
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.8)
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.1, "Interval A")
tg2 <- tg.insertInterval(tg2, "INTERVALS", 1.2, 2.5, "Interval B")
## Not run:
tg.plot(tg2)
## End(Not run)
```
tg.insertInterval

Description

Inserts new interval into an empty space in interval tier: a) Into an already existing interval with empty label (most common situation because, e.g., a new interval tier has one empty interval from beginning to the end. b) Outside of existing intervals (left or right), this may create another empty interval between.

Usage

tg.insertInterval(tg, tierInd, tStart, tEnd, label = "")

Arguments

tg TextGrid object
tierInd tier index or "name"
tStart start time of the new interval
tEnd end time of the new interval
label [optional] label of the new interval

Details

In most cases, this function is the same as 1.) tgInsertBoundary(tEnd) and 2.) tgInsertBoundary(tStart, "new label"). But, additional checks are performed: a) tStart and tEnd belongs to the same empty interval, or b) both times are outside of existing intervals (both left or both right).

Intersection of the new interval with more already existing (even empty) does not make a sense and is forbidden.

In many situations, in fact, this function creates more than one interval. E.g., let’s assume an empty interval tier with one empty interval from 0 to 5 sec. 1.) We insert a new interval from 1 to 2 with label "he". Result: three intervals, 0-1 "", 1-2 "he", 2-5 "". 2.) Then, we insert an interval from 7 to 8 with label "lot". Result: five intervals, 0-1 "", 1-2 "he", 2-5 "", 5-7 "", 7-8 "lot" Note: the empty 5-7 "" interval is inserted because we are going outside of the existing tier. 3.) Now, we insert a new interval exactly between 2 and 3 with label "said". Result: really only one interval is created (and only the right boundary is added because the left one already exists): 0-1 "", 1-2 "he", 2-3 "said", 3-5 "", 5-7 "", 7-8 "lot". 4.) After this, we want to insert another interval, 3 to 5: label "a". In fact, this does not create any new interval at all. Instead of that, it only sets the label to the already existing interval 3-5. Result: 0-1 "", 1-2 "he", 2-3 "said", 3-5 "a", 5-7 "", 7-8 "lot". This function is not implemented in Praat (6.0.14). And it is very useful for adding separate intervals to an empty area in interval tier, e.g., result of voice activity detection algorithm. On the other hand, if we want continuously add new consequential intervals, tgInsertBoundary() may be more useful. Because, in the tgInsertInterval() function, if we calculate both boundaries separately for each interval, strange situations may happen due to numeric round-up errors, like 3.14*5 != 15.7. In such cases, it may be hard to obtain precisely consequential time instances. As 3.14*5 is
slightly larger than 15.7 (let’s try to calculate 15.7 - 3.14*5), if you calculate tEnd of the first interval as 3.14*5 and tStart of the second interval as 15.7, this function refuse to create the second interval because it would be an intersection. In the opposite case (tEnd of the 1st: 15.7, tStart of the 2nd: 3.14*5), it would create another "micro" interval between these two slightly different time instances. Instead of that, if you insert only one boundary using the tgInsertBoundary() function, you are safe that only one new interval is created. But, if you calculate the "15.7" (no matter how) and store in the variable and then, use this variable in the tgInsertInterval() function both for the tEnd of the 1st interval and tStart of the 2nd interval, you are safe, it works fine.

Value

TextGrid object

See Also

tg.insertBoundary, tg.removeIntervalLeftBoundary, tg.removeIntervalRightBoundary, tg.removeIntervalBothBoundaries, tg.boundaryMagnet, tg.duplicateTierMergeSegments

Examples

tg <- tg.sample()
tg2 <- tg.insertNewIntervalTier(tg, 1, "INTERVALS")
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.8)
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.1, "Interval A")
tg2 <- tg.insertInterval(tg2, "INTERVALS", 1.2, 2.5, "Interval B")
## Not run:
tg.plot(tg2)
## End(Not run)

tg.insertNewIntervalTier

tg.insertNewIntervalTier

description

Inserts new interval tier to the specified index (existing tiers are shifted). The new tier contains one empty interval from beginning to end. Then, if we add new boundaries, this interval is divided to smaller pieces.

Usage

tg.insertNewIntervalTier(tg, newInd = Inf, newTierName, tMin = NA, tMax = NA)
tg.insertNewPointTier

Arguments

tg        TextGrid object
newInd    new tier index (1 = the first, Inf = the last [default])
newTierName    new tier name

Value

TextGrid object

See Also

tg.insertInterval, tg.insertNewPointTier, tg.duplicateTier, tg.duplicateTierMergeSegments, tg.removeTier

Examples

## Not run:
tag <- tg.sample()
tag2 <- tg.insertNewIntervalTier(tag, 1, "INTERVALS")
tag2 <- tg.insertBoundary(tag2, "INTERVALS", 0.8)
tag2 <- tg.insertBoundary(tag2, "INTERVALS", 0.1, "Interval A")
tag2 <- tg.insertInterval(tag2, "INTERVALS", 1.2, 2.5, "Interval B")
tag2 <- tg.insertNewIntervalTier(tag2, Inf, "LastTier")
tag2 <- tg.insertInterval(tag2, "LastTier", 1, 3, "This is the last tier")
tag.plot(tag2)

## End(Not run)

---

tg.insertNewPointTier  tg.insertNewPointTier

description

Inserts new point tier to the specified index (existing tiers are shifted).

Usage

tg.insertNewPointTier(tg, newInd = Inf, newTierName)

Arguments

tg        TextGrid object
newInd    new tier index (1 = the first, Inf = the last [default])
newTierName    new tier name
Value

TextGrid object

See Also

tg.insertPoint, tg.insertNewIntervalTier, tg.duplicateTier, tg.removeTier

Examples

```r
## Not run:
tg <- tg.sample()
tg2 <- tg.insertNewPointTier(tg, 1, "POINTS")
tg2 <- tg.insertPoint(tg2, "POINTS", 3, "MY POINT")
tg2 <- tg.insertNewPointTier(tg2, Inf, "POINTS2")  # the last tier
tg2 <- tg.insertPoint(tg2, "POINTS2", 2, "point in the last tier")
tg.plot(tg2)
```

## End(Not run)

---

tg.insertPoint  
tg.insertPoint

Description

Inserts new point to point tier of the given index.

Usage

tg.insertPoint(tg, tierInd, time, label)

Arguments

- **tg**: TextGrid object
- **tierInd**: tier index or "name"
- **time**: time of the new point
- **label**: time of the new point

Value

TextGrid object

See Also

tg.removePoint, tg.insertInterval, tg.insertBoundary
## Not run:
tg <- tg.sample()
tg2 <- tg.insertPoint(tg, "phoneme", 1.4, "NEW POINT")
tg.plot(tg2)

## End(Not run)

tg.isIntervalTier

tg.isIntervalTier

---

### Description

Returns TRUE if the tier is IntervalTier, FALSE otherwise.

### Usage

tg.isIntervalTier(tg, tierInd)

### Arguments

- **tg**: TextGrid object
- **tierInd**: tier index or "name"

### Value

TRUE / FALSE

### See Also

tg.isPointTier, tg.getTierName, tg.findLabels

### Examples

tg <- tg.sample()
tg.isIntervalTier(tg, 1)
tg.isIntervalTier(tg, "word")
tg.isPointTier

Description

Returns TRUE if the tier is PointTier, FALSE otherwise.

Usage

tg.isPointTier(tg, tierInd)

Arguments

- **tg**: TextGrid object
- **tierInd**: tier index or "name"

Value

TRUE / FALSE

See Also

tg.isIntervalTier, tg.getTierName, tg.findLabels

Examples

tg <- tg.sample()
tg.isPointTier(tg, 1)
tg.isPointTier(tg, "word")

tg.plot

Description

Plots interactive TextGrid using dygraphs package.

Usage

tg.plot(
    tg,
    group = "",
    pt = NULL,
    it = NULL,
    formant = NULL,
    formantScaleIntensity = TRUE,
tg.plot

formantDrawBandwidth = TRUE,
pitch = NULL,
pitchScaleIntensity = TRUE,
pitchShowStrength = FALSE,
snd = NULL
)

Arguments

tg TextGrid object

group [optional] character string, name of group for dygraphs synchronization

pt [optional] PitchTier object

it [optional] IntensityTier object

formant [optional] Formant object

formantScaleIntensity
[optional] Point size scaled according to relative intensity

formantDrawBandwidth
[optional] Draw formant bandwidth

pitch [optional] Pitch object

pitchScaleIntensity
[optional] Point size scaled according to relative intensity

pitchShowStrength
[optional] Show strength annotation

snd [optional] Sound object

See Also
tg.read, pt.plot, it.plot, pitch.plot

Examples

## Not run:
tg <- tg.sample()
tg.plot(tg)
tg.plot(tg.sample(), pt = pt.sample())

## End(Not run)
tg.removeIntervalBothBoundaries

Description

Loads TextGrid from Praat in Text or Short text format (UTF-8), it handles both Interval and Point tiers. Labels can may contain quotation marks and new lines.

Usage

tg.read(fileNameTextGrid, encoding = "UTF-8")

Arguments

fileNameTextGrid    Input file name
encoding             File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

TextGrid object

See Also

tg.write, tg.plot, tg.repairContinuity, tg.createNewTextGrid, tg.findLabels, tg.duplicateTierMergeSegments, pt.read, pitch.read, formant.read, it.read, col.read

Examples

## Not run:
tg <- tg.read("demo/H.TextGrid")
tg.plot(tg)

## End(Not run)
**Description**

Remove both left and right boundary of interval of the given index in Interval tier. In fact, this operation concatenate three intervals into one (and their labels). It cannot be applied to the first and the last interval because they contain beginning or end boundary of the tier. E.g., let’s assume interval 1-2-3. We remove both boundaries of the 2nd interval. The result is one interval 123. If we do not want to concatenate labels (we wanted to remove the label including its interval), we can set the label of the second interval to the empty string "" before this operation. If we only want to remove the label of interval "without concatenation", i.e., the desired result is 1-empty-3, it is not this operation of removing boundaries. Just set the label of the second interval to the empty string "".

**Usage**

tg.removeIntervalBothBoundaries(tg, tierInd, index)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tg</td>
<td>TextGrid object</td>
</tr>
<tr>
<td>tierInd</td>
<td>tier index or &quot;name&quot;</td>
</tr>
<tr>
<td>index</td>
<td>index of the interval</td>
</tr>
</tbody>
</table>

**Value**

TextGrid object

**See Also**

tg.removeIntervalLeftBoundary, tg.removeIntervalRightBoundary, tg.insertBoundary, tg.insertInterval

**Examples**

```r
## Not run:
tg <- tg.sample()
tg.plot(tg)
tg2 <- tg.removeIntervalBothBoundaries(tg, "word", 3)
tg.plot(tg2)

## End(Not run)
```
Description

Remove left boundary of the interval of the given index in Interval tier. In fact, it concatenates two intervals into one (and their labels). It cannot be applied to the first interval because it is the start boundary of the tier. E.g., we have interval 1-2-3, we remove the left boundary of the 2nd interval, the result is two intervals 12-3. If we do not want to concatenate labels, we have to set the label to the empty string "" before this operation.

Usage

tg.removeIntervalLeftBoundary(tg, tierInd, index)

Arguments

tg TextGrid object
tierInd tier index or "name"
index index of the interval

Value

TextGrid object

See Also

tg.removeIntervalRightBoundary, tg.removeIntervalBothBoundaries, tg.insertBoundary, tg.insertInterval

Examples

## Not run:
tg <- tg.sample()
tg.plot(tg)
tg2 <- tg.removeIntervalLeftBoundary(tg, "word", 3)
tg.plot(tg2)

## End(Not run)
tg.removeIntervalRightBoundary

Description

Remove right boundary of the interval of the given index in Interval tier. In fact, it concatenates two intervals into one (and their labels). It cannot be applied to the last interval because it is the end boundary of the tier. E.g., we have interval 1-2-3, we remove the right boundary of the 2nd interval, the result is two intervals 1-23. If we do not want to concatenate labels, we have to set the label to the empty string "" before this operation.

Usage

tg.removeIntervalRightBoundary(tg, tierInd, index)

Arguments

tg TextGrid object
tierInd tier index or "name"
index index of the interval

Value

TextGrid object

See Also

tg.removeIntervalLeftBoundary, tg.removeIntervalBothBoundaries, tg.insertBoundary, tg.insertInterval

Examples

## Not run:
tg <- tg.sample()
tg.plot(tg)
tg2 <- tg.removeIntervalRightBoundary(tg, "word", 3)
tg.plot(tg2)

## End(Not run)
tg.removePoint

Description
Remove point of the given index from the point tier.

Usage
tg.removePoint(tg, tierInd, index)

Arguments
- tg: TextGrid object
- tierInd: tier index or "name"
- index: index of point to be removed

Value
TextGrid object

See Also
tg.insertPoint, tg.getNumberPoints, tg.removeIntervalBothBoundaries

Examples
tg <- tg.sample()
tg$phoneme$label
tg2 <- tg.removePoint(tg, "phoneme", 1)
tg2$phoneme$label

tg.removeTier

Description
Removes tier of the given index.

Usage
tg.removeTier(tg, tierInd)

Arguments
- tg: TextGrid object
- tierInd: tier index or "name"
tg.repairContinuity

Value

TextGrid object

See Also

tg.insertNewIntervalTier, tg.insertNewPointTier, tg.duplicateTier

Examples

## Not run:
tg <- tg.sample()
tg.plot(tg)
tg2 <- tg.removeTier(tg, "word")
tg.plot(tg2)
## End(Not run)

tg.repairContinuity  tg.repairContinuity

description

Repairs problem of continuity of T2 and T1 in interval tiers. This problem is very rare and it should not appear. However, e.g., automatic segmentation tool Prague Labeller produces random numeric round-up errors featuring, e.g., T2 of preceding interval is slightly higher than the T1 of the current interval. Because of that, the boundary cannot be manually moved in Praat edit window.

Usage

tg.repairContinuity(tg, verbose = TRUE)

Arguments

tg  
TextGrid object

verbose  
[optional, default=TRUE] If FALSE, the function performs everything quietly.

Value

TextGrid object

See Also

tg.sampleProblem
Examples

## Not run:
```r
tgProblem <- tg.sampleProblem()
tgNew <- tg.repairContinuity(tgProblem)
tg.write(tgNew, "demo_problem_OK.TextGrid")
## End(Not run)
```

tg.sample

Description

Returns sample TextGrid.

Usage

tg.sample()

Value

TextGrid

See Also

tg.plot

Examples

tg <- tg.sample()
tg.plot(tg)

tg.sampleProblem

Description

Returns sample TextGrid with continuity problem.

Usage

tg.sampleProblem()

Value

TextGrid
See Also

tg.repairContinuity

Examples

tg <- tg.sampleProblem()
tg2 <- tg.repairContinuity(tg)
tg2 <- tg.repairContinuity(tg2)
tg.plot(tg2)

tg.setLabel  tg.setLabel

Description

Sets (changes) label of interval or point of the given index in the interval or point tier.

Usage

tg.setLabel(tg, tierInd, index, newLabel)

Arguments

tg  TextGrid object
tierInd  tier index or "name"
index  index of interval or point
newLabel  new "label"

See Also

tg.getLabel

Examples

tg <- tg.sample()
tg2 <- tg.setLabel(tg, "word", 3, "New Label")
tg.getLabel(tg2, "word", 3)
Description

Sets (changes) name of tier of the given index.

Usage

tg.setTierName(tg, tierInd, name)

Arguments

tg
TextGrid object
tierInd
tier index or "name"
name
new "name" of the tier

See Also

tg.getTierName

Examples

tg <- tg.sample()
tg2 <- tg.setTierName(tg, "word", "WORDTIER")
tg.getTierName(tg2, 4)

Description

Saves TextGrid to the file. TextGrid may contain both interval and point tiers (tg[[1]], tg[[2]],
tg[[3]], etc.). If tier type is not specified in $type, is is assumed to be "interval". If specified,
$type have to be "interval" or "point". If there is no class(tg)["tmin"] and class(tg)["tmax"],
they are calculated as min and max of all tiers. The file is saved in UTF-8 encoding.

Usage

tg.write(tg, fileNameTextGrid, format = "short")
**tg.write**

**Arguments**

- `tg`: TextGrid object
- `fileNameTextGrid`: Output file name
- `format`: Output file format ("short" (default, short text format) or "text" (a.k.a. full text format))

**See Also**

`tg.read, pt.write`

**Examples**

```r
## Not run:
tg <- tg.sample()
tg.write(tg, "demo_output.TextGrid")

## End(Not run)
```
Index

as.formant, 4
as.it, 4
as.pitch, 5
as.pt, 6
as.snd, 6
as.tg, 7
ceiling, 43
col.read, 8, 9, 14, 26, 31, 41, 82
col.write, 9
Conj, 17
detectEncoding, 10
fft, 17
floor, 43
formant.cut, 10, 12
formant.cut0, 11, 11
formant.plot, 11, 12, 12, 15, 16, 30, 40
formant.read, 8, 11–13, 13, 15, 16, 26, 41, 82
formant.sample, 13, 14, 32
formant.toArray, 13, 15, 16
formant.toFrame, 15
formant.write, 16
ifft, 17, 45
Im, 17
isInt, 17, 18–20, 45
isLogical, 18, 18, 19, 20
isNum, 18, 19, 20, 45
isString, 18, 19, 20, 51–53
it.cut, 20, 22, 23, 25, 26
it.cut0, 21, 21, 23, 25, 26
it.interpolate, 21, 22, 22, 23–27
it.legendre, 21–23, 23, 24, 25
it.legendreDemo, 21–23, 24, 25
it.legendreSynth, 21–24, 24
it.plot, 21–25, 25, 26, 27, 81
it.read, 8, 14, 21–25, 26, 27, 31, 41, 82
it.sample, 14, 27, 32
it.write, 23, 25, 26, 27
Mod, 17
pitch.cut, 28, 29
pitch.cut0, 28, 29
pitch.plot, 28, 29, 30, 33, 40, 81
pitch.read, 8, 14, 26, 28–30, 31, 33, 34, 41, 82
pitch.sample, 14, 30, 32
pitch.toArray, 30, 32, 33
pitch.toFrame, 33, 33
pitch.write, 34
pt.cut, 34, 36–38, 40, 41, 57, 58
pt.cut0, 35, 35, 37, 38, 40, 41, 57, 58
pt.Hz2ST, 35, 36, 36, 37–42
pt.interpolate, 35–37, 37, 38–42
pt.legendre, 35–37, 37, 39–41
pt.legendreDemo, 35, 36, 38, 39, 40
pt.legendreSynth, 35, 36, 38, 39, 39
pt.plot, 30, 35–40, 40, 41, 42, 81
pt.read, 8, 14, 26, 31, 34–40, 41, 42, 82
pt.sample, 14, 32, 41
pt.write, 37, 40, 41, 42, 91
Re, 17
round, 43
round2, 43, 45
seqM, 44
snd.cut, 45, 47
snd.cut0, 46, 46
snd.plot, 46, 47, 47, 49
snd.read, 46–48, 48, 50
snd.sample, 49
snd.write, 50
str_contains, 51, 51, 52, 53
str_find, 51, 52, 53
str_find1, 51, 52, 53
strTrim, 51