

Intro

This is an introductory tutorial to PRAAT, a computer program with which phoneticians can analyse, synthesize, and manipulate speech, and create high-quality pictures for their articles and thesis. You are advised to work through all of this tutorial.

You can read this tutorial sequentially with the help of the ">" and "<" buttons, or go to the desired information by clicking on the blue links.

Intro 1. How to get a sound: record, read, formula.

Intro 2. What to do with a sound: write, view.

Intro 3. Spectral analysis

 spectrograms: view, configure, query, the Spectrogram object.

 spectral slices: view, configure, the Spectrum object.

Intro 4. Pitch analysis

 pitch contours: view, configure, query, the Pitch object.

Intro 5. Formant analysis

 formant contours: view, configure, query, the Formant object.

Intro 6. Intensity analysis

 intensity contours: view, configure, query, the Intensity object.

Intro 7. Annotation

Intro 8. Manipulation: of pitch, duration, intensity, formants.

There are also more specialized tutorials:

Phonetics:

- o Voice analysis (jitter, shimmer, noise): Voice
- o Listening experiments: ExperimentMFC
- o Sound files
- o Filtering
- o Source-filter synthesis
- o Articulatory synthesis

Phonology:

- o OT learning

Statistics:

- o Principal component analysis
- o Multidimensional scaling
- o Discriminant analysis

General:

- o Printing
- o Scripting

The authors

The Praat program is maintained by Paul Boersma and David Weenink of the Institute of Phonetics Sciences of the University of Amsterdam. Its home page is <http://www.praat.org> or <http://www.fon.hum.uva.nl/praat/>.

For questions and suggestions, mail to the Praat discussion list, which is reachable from the Praat home page, or directly to paul.boersma@uva.nl.

Links to this page

- [Formulas 1.8. Formulas for modification](#)
- [Formulas 7. Attributes of objects](#)
- [Sound](#)
- [SoundEditor](#)
- [Statistics](#)
- [time domain](#)
- [Types of objects](#)
- [What's new?](#)

© *ppgb*, February 9, 2004

Intro 1. How to get a sound

Most of the things most people do with Praat start with a sound. There are at least three ways to get a sound:

Intro 1.1. Recording a sound

Intro 1.2. Reading a sound from disk

Intro 1.3. Creating a sound from a formula

Links to this page

- [Intro](#)
-

© *ppgb*, December 12, 2002

Intro 1.1. Recording a sound

To record a speech sound into Praat, you need to have a computer with a microphone. If you do not have a microphone, try to record from an audio CD instead.

To record from the microphone (or CD), perform the following steps:

1. Choose Record mono Sound... from the New menu in the Object window. A SoundRecorder window will appear on your screen (on MacOS X, you may have to choose Record stereo Sound... instead).
2. In the SoundRecorder window, choose the appropriate input device, i.e. choose "Microphone" (or "CD").
3. Use the **Record** and **Stop** buttons to record a few seconds of your speech (or a few seconds of music from your playing CD).
4. Use the **Play** button to hear what you have recorded.
5. Repeat steps 3 and 4 until you are satisfied with your recording.
6. Click the **To list** button. Your recording will now appear as a Sound object called "Sound untitled" in the List of Objects, which is the left part of the Object window.
7. You can now close the SoundRecorder window.
8. The right part of the Object window shows you what you can do with the sound. Try the **Play** and **Edit** buttons.

For more information, see the SoundRecorder manual page.

Links to this page

- [Intro](#)
- [Intro 1. How to get a sound](#)

© ppgb, December 12, 2002

Intro 1.2. Reading a sound from disk

If you do not have a microphone, you could read a sound file from your disk.

With Read from file... from the Read menu, Praat will be able to read most standard types of sound files. If you do not have a sound file on your disk, you can take one from the Internet.

Links to this page

- [Intro](#)
- [Intro 1. How to get a sound](#)

© *ppgb*, April 8, 2001

Intro 1.3. Creating a sound from a formula

If you have no access to the Internet, you can create a sound from a formula with Create Sound... from the New menu.

Links to this page

- [Intro](#)
 - [Intro 1. How to get a sound](#)
-

© *ppgb*, April 8, 2001

Intro 2. What to do with a sound

As soon as you have a Sound object in the List of Objects, the buttons in the Dynamic menu (the right-hand part of the Object window) will show you what you can do with it.

Intro 2.1. Writing a sound to disk

Intro 2.2. Viewing and editing a sound

Links to this page

- [Intro](#)

© *ppgb*, March 16, 2003

Intro 2.1. Writing a sound to disk

There are several ways to write a sound to disk.

First, the File menu of the SoundRecorder window contains commands to save the left channel, the right channel, or both channels of the recorded sound to any of four standard types of sound files (AIFC, WAV, NeXT/Sun, NIST). These four file types are equivalent; Praat will handle them equally well on every computer. The first three of these types will also be recognized by nearly all other sound-playing programs.

Then, once you have a Sound object in the List of Objects, you can save it in several formats with the commands in the Write menu. Again, the AIFF, AIFC, WAV, NeXT/Sun, and NIST formats are equally fine.

For more information, see the Sound files tutorial.

Links to this page

- [Intro](#)
- [Intro 2. What to do with a sound](#)

© *ppgb*, December 12, 2002

Intro 2.2. Viewing and editing a sound

To see the wave form of a Sound that is in the list of objects, select that Sound and click Edit.

If your sound file is longer than about a minute, you may prefer to open it with Open long sound file... This puts a LongSound object into the list. In this way, most of the sound will stay in the file on disk, and at most 60 seconds will be read into memory each time you play or view a part of it.

Links to this page

- [Intro](#)
- [Intro 2. What to do with a sound](#)

© *ppgb*, February 25, 2003

Intro 3. Spectral analysis

This section describes how you can analyse the spectral content of an existing sound. You will learn how to use *spectrograms* and *spectral slices*.

Intro 3.1. Viewing a spectrogram

Intro 3.2. Configuring the spectrogram

Intro 3.3. Querying the spectrogram

Intro 3.4. The Spectrogram object

Intro 3.5. Viewing a spectral slice

Intro 3.6. Configuring the spectral slice

Intro 3.7. The Spectrum object

Links to this page

- [Extract visible spectrogram](#)
- [Intro](#)
- [Show spectrogram](#)
- [Sound: To Spectrogram...](#)
- [Spectrogram](#)
- [Types of objects](#)

© *ppgb*, March 16, 2003

Intro 3.1. Viewing a spectrogram

To see the spectral content of a sound as a function of time, select a Sound or LongSound object and choose Edit. A SoundEditor or LongSoundEditor window will appear on your screen. In the entire bottom half of this window you will see a greyish image, which is called a *spectrogram*. If you do not see it, choose Show spectrogram from the **Spectrogram** menu.

The spectrogram is a spectro-temporal representation of the sound. The horizontal direction of the spectrogram represents time, the vertical direction represents frequency. The time scale of the spectrogram is the same as that of the waveform, so the spectrogram reacts to your zooming and scrolling. To the left of the spectrogram, you see the frequency scale. The frequency at the bottom of the spectrogram is usually 0 Hz (Hertz, cps, cycles per second), and a common value for the frequency at the top is 5000 Hz.

Darker parts of the spectrogram mean higher energy densities, lighter parts mean lower energy densities. If the spectrogram has a dark area around a time of 1.2 seconds and a frequency of 4000 Hz, this means that the sound has lots of energy for those high frequencies at that time. For many examples of spectrograms of speech sounds, see the textbook by Ladefoged (2001) and the reference work by Ladefoged & Maddieson (1996).

To see what time and frequency a certain part of the spectrogram is associated with, just click on the spectrogram and you will see the vertical time cursor showing the time above the waveform and the horizontal frequency cursor showing the frequency to the left of the spectrogram. This is one of the ways to find the *formant* frequencies for vowels, or the main spectral peaks for fricatives.

Hey, there are white vertical stripes at the edges!

This is normal. Spectral analysis requires an *analysis window* of a certain duration. For instance, if PRAAT wants to know the spectrum at 1.342 seconds, it needs to include information about the signal in a 10-milliseconds window around this time point, i.e., PRAAT will use signal information about all times between 1.337 and 1.347 seconds. At the very edges of the sound, this information is not available: if the sound runs from 0 to 1.8 seconds, no spectrum can be computed between 0 and 0.005 seconds or between 1.795 and 1.800 seconds. Hence the white stripes. If you do not see them immediately when you open the sound, zoom in on the beginning or end of the sound.

When you zoom in on the middle of the sound (or anywhere not near the edges), the white stripes vanish. Suddenly you see only the time stretch between 0.45 and 1.35 seconds, for instance. But PRAAT did not forget what the signal looks like just outside the edges of this time window. To display a spectrogram from 0.45 to 1.35 seconds, PRAAT will use information from the wave form between 0.445 and 1.355 seconds, and if this is available, you will see no white stripes at the edges of the window.

Hey, it changes when I scroll!

This is normal as well, especially for long windows. If your visible time window is 20 seconds long, and the window takes up 1000 screen pixels horizontally, then you might think that every one-pixel-wide vertical line should represent the spectrum of 20 milliseconds of sound. But for reasons of computation

speed, PRAAT will only show the spectrum of the part of the sound that lies around the centre of those 20 milliseconds, not the average or sum of all the spectra in those 20 milliseconds. This *undersampling* of the underlying spectrogram is different from what happens in the drawing of the wave form, where a vertical black line connects the minimum and maximum amplitude of all the samples that fall inside a screen pixel. We cannot do something similar for spectrograms. And since scrolling goes by fixed time steps (namely, 5 percent of the duration of the visible window), rather than by a whole number of screen pixels, the centres of the pixels will fall in different parts of the spectrogram with each scroll. Hence the apparent changes. If your visible window is shorter than a couple of seconds, the scrolling spectrogram will appear much smoother.

The darkness of the spectrogram will also change when you scroll, because the visible part with the most energy is defined as black. When a very energetic part of the signal scrolls out of view, the spectrogram will turn darker. The next section will describe a way to switch this off.

Links to this page

- [Intro](#)
- [Intro 3. Spectral analysis](#)

© *ppgb*, May 13, 2003

Intro 3.2. Configuring the spectrogram

With Spectrogram settings... from the **Spectrogram** menu, you can determine how the spectrogram is computed and how it is displayed. These settings will be remembered across Praat sessions. All these settings have standard values ("factory settings"), which appear when you click **Revert to standard**.

View range (Hz)

the range of frequencies to display. The standard is 0 Hz at the bottom and 5000 Hz at the top. If this maximum frequency is higher than the Nyquist frequency of the Sound (which is half its sampling frequency), some values in the spectrogram will be zero, and the higher frequencies will be drawn in white. You can see this if you record a Sound at 44100 Hz and set the view range from 0 Hz to 25000 Hz.

Window length

the duration of the analysis window. If this is 0.005 seconds (the standard), Praat uses for each frame the part of the sound that lies between 0.0025 seconds before and 0.0025 seconds after the centre of that frame (for Gaussian windows, Praat actually uses a bit more than that). The window length determines the *bandwidth* of the spectral analysis, i.e. the width of the horizontal line in the spectrogram of a pure sine wave (see below). For a Gaussian window, the -3 dB bandwidth is $2 \cdot \sqrt{6 \cdot \ln(2)} / (\pi \cdot \text{Window length})$, or $1.2982804 / \text{Window length}$. To get a 'broad-band' spectrogram (bandwidth 260 Hz), keep the standard window length of 5 ms; to get a 'narrow-band' spectrogram (bandwidth 43 Hz), set it to 30 ms (0.03 seconds). The other window shapes give slightly different values.

Dynamic range (dB)

All values that are more than *Dynamic range* dB below the maximum (perhaps after dynamic compression, see Advanced spectrogram settings...) will be drawn in white. Values in-between have appropriate shades of grey. Thus, if the highest peak in the spectrogram has a height of 30 dB/Hz, and the dynamic range is 50 dB (which is the standard value), then values below -20 dB/Hz will be drawn in white, and values between -20 dB/Hz and 30 dB/Hz will be drawn in various shades of grey.

The bandwidth

To see how the window length influences the bandwidth, first create a 1000-Hz sine wave with Create Sound... by typing $1/2 * \sin(2 \cdot \pi \cdot 1000 \cdot x)$ as the formula, then click **Edit**. The spectrogram will show a horizontal black line. You can now vary the window length in the spectrogram settings and see how the thickness of the lines varies. The line gets thinner if you raise the window length. Apparently, if the analysis window comprises more periods of the wave, the spectrogram can tell us the frequency of the wave with greater precision.

To see this more precisely, create a sum of two sine waves, with frequencies of 1000 and 1200 Hz. the formula is $1/4 * \sin(2 \cdot \pi \cdot 1000 \cdot x) + 1/4 * \sin(2 \cdot \pi \cdot 1200 \cdot x)$. In the editor, you will see a single thick band if the analysis window is short (5 ms), and two separate bands if the analysis window is long (30 ms). Apparently, the frequency resolution gets better with longer analysis windows.

So why don't we always use long analysis windows? The answer is that their time resolution is poor. To see this, create a sound that consists of two sine waves and two short clicks. The formula is $0.02 * (\sin(2 * \pi * 1000 * x) + \sin(2 * \pi * 1200 * x)) + (\text{col}=10000) + (\text{col}=10200)$. If you view this sound, you can see that the two clicks will overlap in time if the analysis window is long, and that the sine waves overlap in frequency if the analysis window is short. Apparently, there is a trade-off between time resolution and frequency resolution. One cannot know both the time and the frequency with great precision.

Advanced settings

The Spectrum menu also contains Advanced spectrogram settings....

Links to this page

- [Intro](#)
- [Intro 3. Spectral analysis](#)
- [Intro 3.6. Configuring the spectral slice](#)

© *ppgb*, September 17, 2003

Intro 3.3. Querying the spectrogram

If you click anywhere inside the spectrogram, a cursor cross will appear, and you will see the time and frequency in red at the top and to the left of the window. To see the time in the Info window, choose **Get cursor** from the **Query** menu or press the F6 key. To see the frequency in the Info window, choose **Get frequency** from the **Spec.** menu.

To query the power of the spectrogram at the cursor cross, choose **Get spectral power at cursor cross** from the **Spec.** menu or press the F9 key. The Info window will show you the power density, expressed in Pascal²/Hz.

Links to this page

- [Intro](#)
- [Intro 3. Spectral analysis](#)

© *ppgb*, April 3, 2003

Intro 3.4. The Spectrogram object

To print a spectrogram, or to put it in an EPS file or on the clipboard for inclusion in your word processor, you first have to create a Spectrogram object in the List of Objects. You do this either by choosing **Extract visible spectrogram** from the Spectrum menu in the SoundEditor or TextGridEditor window, or by selecting a Sound object in the list and choosing Sound: To Spectrogram... from the **Spectrum** submenu. In either case, a new Spectrogram object will appear in the list. To draw this Spectrogram object to the Picture window, select it and choose the Spectrogram: Paint... command. From the Picture window, you can print it, save it to an EPS file, or copy it to the clipboard.

Links to this page

- [Intro](#)
- [Intro 3. Spectral analysis](#)

© *ppgb*, March 17, 2003

Intro 3.5. Viewing a spectral slice

With **View spectral slice** from the **Spec.** menu in the SoundEditor and the TextGridEditor, you can see the frequency spectrum at the time cursor or the average frequency spectrum in the time selection.

Spectral slice at the cursor

If you click anywhere in the wave form of the SoundEditor or TextGridEditor windows, a cursor will appear at that time. If you then choose **View spectral slice**, PRAAT will create a Spectrum object named *slice* in the Objects window and show it in a SpectrumEditor window. In this way, you can inspect the frequency contents of the signal around the cursor position.

Spectral slice from a selection

If you drag the mouse through the wave form of the SoundEditor or TextGridEditor windows, a time selection will appear. If you then choose **View spectral slice**, PRAAT will again create a Spectrum object named *slice* in the Objects window and show it in a SpectrumEditor window. In this way, you can inspect the frequency contents of the signal in the selection.

Links to this page

- [Intro](#)
- [Intro 3. Spectral analysis](#)
- [What's new?](#)

© *ppgb*, March 16, 2003

Intro 3.6. Configuring the spectral slice

Spectral slice at the cursor

What PRAAT does precisely, depends on your Spectrogram settings. Suppose that the *window length* setting is 0.005 seconds (5 milliseconds). If the *window shape* is not Gaussian, PRAAT will extract the part of the sound that runs from 2.5 milliseconds before the cursor to 2.5 ms after the cursor. PRAAT then multiplies this 5 ms long signal by the window shape, then computes a spectrum with the method of Sound: To Spectrum (fft), which is put into the Objects window and opened in an editor window. If the window shape is Gaussian, PRAAT will extract a part of the sound that runs from 5 milliseconds before the cursor to 5 ms after the cursor. The spectrum will then be based on a ‘physical’ window length of 10 ms, although the ‘effective’ window length is still 5 ms (see Intro 3.2. Configuring the spectrogram for details).

Spectral slice from a selection

What PRAAT does precisely, again depends on the *window shape* of your Spectrogram settings. Suppose that your selection is 50 ms long. PRAAT will extract the entire selection, then multiply this 50 ms long signal by the window shape, then compute a spectrum, put it into the Objects window and open it an editor window. This procedure is equivalent to choosing **Extract windowed selection...** with a *relative duration* of 1.0, followed by **To Spectrum (fft)**, followed by **Edit**.

If the window is Gaussian, PRAAT will still only use the selection, without doubling its duration. This means that the spectrum that you see in this case will mainly be based on the centre half of the selection, and the signal near the edges will be largely ignored.

Links to this page

- [Intro](#)
- [Intro 3. Spectral analysis](#)

© ppgb, March 16, 2003

Intro 3.7. The Spectrum object

To compute a Fourier frequency spectrum of an entire sound, select a Sound object and choose To Spectrum (fft) from the **Spectrum** submenu. A new Spectrum object will appear in the List of Objects. To view or modify it (or listen to its parts), click Edit. To print it, choose one of the **Draw** commands to draw the Spectrum object to the Picture window first.

Links to this page

- [Intro](#)
- [Intro 3. Spectral analysis](#)

© *ppgb*, April 3, 2003

Intro 4. Pitch analysis

This section describes how you can analyse the pitch contour of an existing sound.

Intro 4.1. Viewing a pitch contour

Intro 4.2. Configuring the pitch contour

Intro 4.3. Querying the pitch contour

Intro 4.4. The Pitch object

Links to this page

- [Extract visible pitch contour](#)
- [Intro](#)
- [Pitch](#)
- [Show pitch](#)
- [Types of objects](#)

© *ppgb*, March 16, 2003

Intro 4.1. Viewing a pitch contour

To see the pitch contour of an existing sound as a function of time, select a Sound or LongSound object and choose Edit. A SoundEditor window will appear on your screen. The bottom half of this window will contain a pitch contour, drawn as a blue line or as a sequence of blue dots. If you do not see the pitch contour, choose Show pitch from the **Pitch** menu.

To the right of the window, you may see three pitch values, written with blue digits: at the bottom, you see the floor of the viewable pitch range, perhaps 75 Hz; at the top, you see the ceiling of the pitch range, perhaps 600 Hz; and somewhere in between, you see the pitch value at the cursor, or the average pitch in the selection.

Links to this page

- [Intro](#)
- [Intro 4. Pitch analysis](#)

© *ppgb*, May 21, 2003

Intro 4.2. Configuring the pitch contour

With Pitch settings... from the **Pitch** menu, you can determine how the pitch contour is displayed and how it is computed. These settings will be remembered across Praat sessions. All these settings have standard values ("factory settings"), which appear when you click **Revert to standards**.

The *pitch range* setting

This is the most important setting for pitch analysis. The standard range is from 75 to 600 Hertz, which means that the pitch analysis method will only find values between 75 and 600 Hz. The range that you set here will be shown to the right of the analysis window.

For a male voice, you may want to set the floor to 75 Hz, and the ceiling to 300 Hz; for a female voice, set the range to 100-600 Hz instead. For creaky voice you will want to set it much lower than 75 Hz.

Here is why you have to supply these settings. If the pitch floor is 75 Hz, the pitch analysis method requires a 40-millisecond analysis window, i.e., in order to measure the F0 at a time of, say, 0.850 seconds, PRAAT needs to consider a part of the sound that runs from 0.830 to 0.870 seconds. These 40 milliseconds correspond to 3 maximum pitch periods ($3/75 = 0.040$). If you set the pitch floor down to 25 Hz, the analysis window will grow to 120 milliseconds (which is again 3 maximum pitch periods), i.e., all times between 0.790 and 0.910 seconds will be considered. This makes it less easy to see fast F0 changes.

So setting the floor of the pitch range is a technical requirement for the pitch analysis. If you set it too low, you will miss very fast F0 changes, and if you set it too high, you will miss very low F0 values. For children's voices you can often use 200 Hz, although 75 Hz will still give you the same time resolution as you get for the males.

The *units* setting

This setting determines the units of the vertical pitch scale. Most people like to see the pitch range in Hertz, but there are several other possibilities.

Advanced settings

The Pitch menu also contains Advanced pitch settings....

Links to this page

- [Intro](#)
- [Intro 4. Pitch analysis](#)
- [Intro 6.2. Configuring the intensity contour](#)

© *ppgb*, September 16, 2003

Intro 4.3. Querying the pitch contour

With Get pitch from the **Pitch** menu in the SoundEditor or TextGridEditor, you get information about the pitch at the cursor or in the selection. If a cursor is visible in the window, **Get pitch** writes to the Info window the linearly interpolated pitch at that time; if a time selection is visible inside the window, **Get pitch** writes to the Info window the mean (average) pitch in the visible part of that selection; otherwise, **Get pitch** writes the average pitch in the visible part of the sound.

Links to this page

- [Intro](#)
- [Intro 4. Pitch analysis](#)

© *ppgb*, June 14, 2004

Intro 4.4. The Pitch object

The pitch contour that is visible in the SoundEditor or TextGridEditor window, can be copied as a separate Pitch object to the List of Objects. To do this, choose Extract visible pitch contour from the **Pitch** menu.

Another way to get a separate Pitch object is to select a Sound object in the list choose Sound: To Pitch... (preferred) or any of the other methods from the Periodicity submenu.

To view and modify the contents of a Pitch object, select it and choose Edit. This creates a PitchEditor window on your screen.

To save a pitch contour to disk, select the Pitch object in the list and choose one of the commands in the Write menu.

Later on, you can read the saved file again with Read from file... from the Read menu.

To draw a Pitch object to the Picture window, select it and choose any of the commands in the Draw submenu. From the Picture window, you can print it, save it to an EPS file, or copy it to the clipboard for inclusion in your word processor.

Links to this page

- [Intro](#)
- [Intro 4. Pitch analysis](#)

© *ppgb*, March 16, 2003

Intro 5. Formant analysis

This section describes how you can analyse the formant contours of an existing sound.

- Intro 5.1. Viewing formant contours
- Intro 5.2. Configuring the formant contours
- Intro 5.3. Querying the formant contours
- Intro 5.4. The Formant object

Links to this page

- [Extract visible formant contour](#)
- [Intro](#)
- [Show formant](#)
- [Types of objects](#)

© *ppgb*, March 16, 2003

Intro 5.1. Viewing formant contours

To see the formant contours of a sound as functions of time, select a Sound or LongSound object and choose Edit. A SoundEditor window will appear on your screen. The analysis part of this window will contain formant contours, drawn as red speckles. If you do not see the formant contours, choose Show formant from the **Formant** menu.

Links to this page

- [Intro](#)
- [Intro 5. Formant analysis](#)

© *ppgb*, March 16, 2003

Intro 5.2. Configuring the formant contours

The formant analysis parameters, with you can set with the **Formant** menu, are important. For a female voice, you may want to set the maximum frequency to 5500 Hz; for a male voice, set it to 5000 Hz instead. For more information about analysis parameters, see Sound: To Formant (burg)....

Links to this page

- [Intro](#)
- [Intro 5. Formant analysis](#)

© ppgb, March 16, 2003

Intro 5.3. Querying the formant contours

With Get first formant from the Formant menu in the SoundEditor or TextGridEditor, you get information about the first formant at the cursor or in the selection. If there is a cursor, **Get first formant** writes to the Info window the linearly interpolated first formant at that time. If there is a true selection, **Get first formant** writes to the Info window the mean first formant in the visible part of that selection. The same goes for Get second formant and so on.

Links to this page

- [Intro](#)
- [Intro 5. Formant analysis](#)

© *ppgb*, June 16, 2004

Intro 5.4. The Formant object

The formant contours that are visible in the SoundEditor or TextGridEditor window, can be copied as a separate Formant object to the List of Objects. To do this, choose Extract visible formant contour from the Formant menu.

Another way to get a separate Formant object is to select a Sound object in the list choose Sound: To Formant (burg)... (preferred) or any of the other methods from the Formants & LPC submenu.

Saving formant contours to disk

To save formant contours to disk, select the Formant object in the list and choose one of the commands in the Write menu.

Later on, you can read the saved file again with Read from file... from the Read menu.

Drawing formant contours

To draw a Formant object to the Picture window, select it and choose any of the commands in the Draw submenu. From the Picture window, you can print it, save it to an EPS file, or copy it to the clipboard for inclusion in your word processor.

Links to this page

- [Intro](#)
- [Intro 5. Formant analysis](#)

© *ppgb*, March 16, 2003

Intro 6. Intensity analysis

This section describes how you can analyse the intensity contour of an existing sound.

- Intro 6.1. Viewing an intensity contour
- Intro 6.2. Configuring the intensity contour
- Intro 6.3. Querying the intensity contour
- Intro 6.4. The Intensity object

Links to this page

- [Extract visible intensity contour](#)
- [Intro](#)
- [Show intensity](#)
- [Types of objects](#)

© *ppgb*, March 16, 2003

Intro 6.1. Viewing an intensity contour

To see the intensity contour of a sound as a function of time, select a Sound or LongSound object and choose Edit. A SoundEditor window will appear on your screen. The analysis part of this window will contain an intensity contour, drawn as a yellow line (choose **Show intensity** from the **Intensity** menu if it is not visible). This also works in the TextGridEditor.

Links to this page

- [Intro](#)
- [Intro 6. Intensity analysis](#)

© *ppgb*, March 16, 2003

Intro 6.2. Configuring the intensity contour

With **Intensity settings..** from the **Int.** menu, you can set the number of time points for the intensity contour, analogously to the pitch contour. You can also set the vertical scale (e.g. from 50 to 100 dB). The time span over which the intensity is averaged (smoothed) is determined by the **Minimum pitch** setting (see Intro 4.2. Configuring the pitch contour.)

Links to this page

- [Intro](#)
- [Intro 6. Intensity analysis](#)

© *ppgb*, April 3, 2003

Intro 6.3. Querying the intensity contour

To ask for the intensity at the cursor, or the average intensity in the visible part of the selection, choose **Get intensity** from the **Int.** menu or press the F11 key.

Links to this page

- [Intro](#)
- [Intro 6. Intensity analysis](#)

© *ppgb*, March 16, 2003

Intro 6.4. The Intensity object

To print an intensity contour, or to put it in an EPS file or on the clipboard for inclusion in your word processor, you first have to create an Intensity object in the List of Objects. You do this either by choosing **Extract visible intensity contour** from the Intensity menu in the SoundEditor or TextGridEditor window, or by selecting a Sound object in the list and choosing Sound: To Intensity.... In either case, a new Intensity object will appear in the list. To draw the Intensity object to the Picture window, select it and choose **Draw....** From the Picture window, you can print it, save it to an EPS file, or copy it to the clipboard.

Links to this page

- [Intro](#)
- [Intro 6. Intensity analysis](#)

© *ppgb*, March 16, 2003

Intro 7. Annotation

You can annotate existing Sound objects and sound files (LongSound objects).

The labelling data will reside in a TextGrid object. This object is separate from the sound, which means that you will often see two objects in the list: a Sound or LongSound, and a TextGrid.

Creating a TextGrid

You create a new empty TextGrid from the Sound or LongSound with Sound: To TextGrid... or LongSound: To TextGrid... from the **Annotate** submenu. In this way, the time domain of the TextGrid will automatically equal that of the sound (if you choose Create TextGrid... from the New menu instead, you will have to supply the time domain by yourself).

When you create a TextGrid, you specify the names of the *tiers*. For instance, if you want to segment the sound into words and into phonemes, you may want to create two tiers and call them "words" and "phonemes" (you can easily add, remove, and rename tiers later). Since both of these tiers are *interval tiers* (you label the intervals between the word and phoneme boundaries, not the boundaries themselves), you specify "phonemes words" for *Tier names*, and you leave the *Point tiers* empty.

View and edit

You can edit a TextGrid object all by itself, but you will normally want to see the sound in the editor window as well. To achieve this, you select both the Sound (or LongSound) and the TextGrid, and click Edit. A TextGridEditor will appear on your screen. Like the Sound editor, the TextGrid editor will show you a spectrogram, a pitch contour, a formant contour, and an intensity contour. This editor will allow you to add, remove, and edit labels, boundaries, and tiers. Under Help in the TextGridEditor, you will find the TextGridEditor manual page. You are strongly advised to read it, because it will show you how you can quickly zoom (drag the mouse), play (click a rectangle), or edit a label (just type).

Save

You will normally write the TextGrid to disk with Write to text file... or Write to short text file.... It is true that Write to binary file... will also work, but the others give you a file you can read with any text editor.

However you saved it, you can read the TextGrid into Praat later with Read from file....

Links to this page

- [Intro](#)
- [Labelling](#)
- [Segmentation](#)
- [Types of objects](#)

© *ppgb*, January 13, 2003

Intro 8. Manipulation

- Intro 8.1. Manipulation of pitch
- Intro 8.2. Manipulation of duration
- Intro 8.3. Manipulation of intensity

Links to this page

- [Intro](#)
-

© *ppgb*, December 12, 2002

Intro 8.1. Manipulation of pitch

To modify the pitch contour of an existing Sound object, you select this Sound and click **To Manipulation**. A Manipulation object will then appear in the list. You can then click Edit to raise a ManipulationEditor, which will show the pitch contour (PitchTier) as a series of thick dots. To reduce the number of dots, choose **Stylize pitch (2 st)** from the **Pitch** menu; it will then be easy to drag the dots about the time-pitch area.

If you click any of the rectangles (or choose any of the "Play" commands from the "View" menu), you will hear the modified sound. By shift-clicking, you will hear the original sound.

To get the modified sound as a separate object, choose "Publish resynthesis" from the "File" menu.

If you modify the duration curve as well (see Intro 8.2. Manipulation of duration), the modified sound will be based on the modified pitch and duration.

Cloning a pitch contour

To use the pitch contour of one Manipulation object as the pitch contour of another Manipulation object, you first choose **Extract pitch tier** for the first Manipulation object, then select the resulting PitchTier object together with the other Manipulation object (e.g. by a click on the PitchTier and a Command-click on the Manipulation), and choose **Replace pitch tier**.

Precise manipulation of pitch

If you know exactly what pitch contour you want, you can create an empty PitchTier with Create PitchTier... from the New menu, then add points with PitchTier: Add point....

For instance, suppose you want to have a pitch that falls from 350 to 150 Hz in one second. You create the PitchTier, then add a point at 0 seconds and a frequency of 350 Hz, and a point at 1 second with a frequency of 150 Hz. You can put this PitchTier into a Manipulation object in the way described above.

Links to this page

- [Intro](#)
- [Intro 8. Manipulation](#)
- [Intro 8.3. Manipulation of intensity](#)
- [Types of objects](#)

Intro 8.2. Manipulation of duration

You can use PRAAT to modify the relative durations in an existing sound.

First, you select a Sound object and click "To Manipulation". A Manipulation object will then appear in the list. You can then click Edit to raise a ManipulationEditor, which will show an empty DurationTier. You can add targets to this tier by choosing "Add duration point at cursor" from the "Dur" menu. The targets will show up as green dots, which you can easily drag about the duration area.

If you click any of the rectangles (or choose any of the Play commands from the View menu), you will hear the modified sound. By shift-clicking, you will hear the original sound.

To get the modified sound as a separate object, choose "Publish resynthesis" from the File menu.

If you modify the pitch curve as well (see Intro 8.1. Manipulation of pitch), the modified sound will be based on the modified duration and pitch.

Precise manipulation of duration

If you know exactly the times and relative durations, it is advisable to write a script (see Scripting). Suppose, for instance, that you have a 355-ms piece of sound, and you want to shorten the first 85 ms to 70 ms, and the remaining 270 ms to 200 ms.

In your first 85 ms, your relative duration should be $70/85$, and during the last 270 ms, it should be $200/270$. The DurationTier does linear interpolation, so it can only approximate these precise times, but fortunately to any precision you like:

```
Create DurationTier... shorten 0 0.085+0.270
Add point... 0.000 70/85
Add point... 0.084999 70/85
Add point... 0.085001 200/270
Add point... 0.0355 200/270
```

To put this DurationTier back into a Manipulation object, you select the two objects together (e.g. a click on the DurationTier and a Command-click on the Manipulation), and choose **Replace duration tier**.

Links to this page

- Intro
- Intro 8. Manipulation
- Intro 8.3. Manipulation of intensity
- Types of objects

© *ppgb*, March 16, 2003

Intro 8.3. Manipulation of intensity

You can modify the intensity contour of an existing sound.

While the pitch and duration of a sound can be modified with the ManipulationEditor (see Intro 8.1. Manipulation of pitch and Intro 8.2. Manipulation of duration), the modification of the intensity curve is done in a different way.

You can create an IntensityTier with the command Create IntensityTier... from the New menu. With Edit, you can add points to this tier. You can then multiply this tier with a sound, by selecting the Sound and the IntensityTier together and clicking Multiply. The points in the IntensityTier thus represent *relative* intensities in dB.

Links to this page

- [Intro](#)
- [Intro 8. Manipulation](#)
- [Types of objects](#)

© ppgb, March 16, 2003

Intro 8.4. Manipulation of formants

The manipulation of formant contours cannot be as straightforward as the manipulation of pitch, duration, or intensity contours. See the Source-filter synthesis tutorial for an explanation of how formants can be modified in Praat.

Links to this page

- [Intro](#)
-

© *ppgb*, April 8, 2001

Voice

This tutorial describes how you can do voice analysis with PRAAT. To understand this tutorial, you have to be familiar with the Intro, which describes the more general features of the SoundEditor window.

Most of PRAAT's voice analysis methods start from the glottal pulses that are visible in the SoundEditor window as blue vertical lines through the waveform. If you do not see these lines, choose Show pulses from the **Pulses** menu. If your sound is long, you may have to zoom in in order to see the separate pulses. You may notice that for some sounds, the time location of the pulses can vary when you zoom or scroll. This is because only the visible part of the sound is used for the analysis. The measurement results will also vary slightly when you zoom or scroll.

The Pulse menu contains the command Voice report, which will show in the Info window the results of many voice measurements for the visible part of the selection (or for the visible part of the whole sound, if there is a cursor instead of a selection or if the selection is not visible).

In general, you will want to be careful about the pitch range. The standard range is 75-600 Hertz, but take a range of e.g. 50-200 Hertz for pathological male voices if that is the typical range. The 'advanced' pitch settings like *silence threshold* and *octave jump cost* can stay at their standard values.

- Voice 1. Voice breaks
- Voice 2. Jitter
- Voice 3. Shimmer
- Voice 4. Additive noise (HNR, harmonicity)
- Voice 5. Comparison with other programs

Links to this page

- [Types of objects](#)
-

© ppgb, June 16, 2004

ExperimentMFC

One of the types of objects in PRAAT, for running a Multiple Forced Choice listening experiment.

With Praat, you can do simple experiments on identification and discrimination. 'Simple' means that for identification, the subject hears a sound and has to click on one of a set of labelled rectangles (optionally, you can have the subject give a goodness-of-fit judgment). For discrimination, you can have simple same-different choices, or more intricate things like AXB, 4I-oddy, and so on.

The advantage of using Praat for this is that it is free, it works on Windows, Unix, and Macintosh, and the whole experiment (experiment file plus sound files) is portable across computers (you can run it from a CD, for instance). Because of the limited possibilities, it is also quite easy to set up the experiment. Just read the description below.

If you require more from your experiment design, you use a dedicated program like Presentation or E-prime instead of Praat. With these programs, you can measure reaction times as well, or you can let the stimulus depend on the subject's previous responses. If you do not need those extra capabilities, you can use the simpler method of Praat. My estimation is that that is fine for 90 percent of the perception experiments that phoneticians perform.

Example

An experiment is defined in a simple text file, which we call an *experiment file*. The following is an example of such an experiment file. The first two lines have to be typed exactly as in this example, the rest depends on your stimuli, on your response categories, and on the way the experiment is to be presented to the listener. The order of the elements in this file cannot be changed.

```
"ooTextFile"  
"ExperimentMFC 3"  
fileNameHead = "Sounds/"  
fileNameTail = ".wav"  
carrierBefore = "weSayTheWord"  
carrierAfter = "again"  
initialSilenceDuration = 0.5 seconds  
interStimulusInterval = 0  
numberOfDifferentStimuli = 4  
    "heed"  
    "hid"  
    "hood"  
    "hud"  
numberOfReplicationsPerStimulus = 3  
breakAfterEvery = 0  
randomize = <PermuteBalancedNoDoublets>  
startText = "Click to start."  
runText = "Choose the vowel that you heard."
```

```

pauseText = "You can have a short break if you like. Click to proceed."
endText = "The experiment has finished."
numberOfResponseCategories = 5
    0.2 0.3 0.7 0.8 "h I d" "" "i"
    0.3 0.4 0.5 0.6 "h E d" "" "e"
    0.4 0.5 0.3 0.4 "h A d" "" "a"
    0.5 0.6 0.5 0.6 "h O d" "" "o"
    0.6 0.7 0.7 0.8 "h U d" "" "u"
numberOfGoodnessCategories = 5
    0.25 0.35 0.10 0.20 "1 (poor)"
    0.35 0.45 0.10 0.20 "2"
    0.45 0.55 0.10 0.20 "3"
    0.55 0.65 0.10 0.20 "4"
    0.65 0.75 0.10 0.20 "5 (good)"

```

This experiment will play 4 different stimuli to the listener, each 3 times. Thus, the listener is confronted with 12 trials.

1. The stimuli

The names of the sound files containing the stimuli must be identical to the names of the stimuli, bracketed with *fileNameHead* and *fileNameTail*. You can see that the stimulus names are *heed*, *hid*, *hood*, and *hud*. Hence, the 4 stimuli are expected in the following 4 files:

```

Sounds/heed.wav
Sounds/hid.wav
Sounds/hood.wav
Sounds/hud.wav

```

You can also use AIFF files, in which case *fileNameTail* would probably be ".aiff", or any other type of sound file that Praat supports. They all have to have the same sampling frequency.

In this example, the experiment will look for the sound files in the directory **Sounds**, which has to be in the same directory as your experiment file. In other words, "Sounds/heed.wav" is a *relative file path*.

Instead of a relative path, you can also supply a *full file path*. Such a path depends on your computer and on your operating system. For instance, if you have a Windows computer and the stimuli are in the directory **D:\Corpus\Autumn\Speaker23**, you can write

```
fileNameHead = "D:\Corpus\Autumn\Speaker23\"
```

If you have a Macintosh (OS X) or Unix computer and the stimuli are in **/Users/mietta/Sounds/Dutch**, you write

```
fileNameHead = "/Users/mietta/Sounds/Dutch/"
```

If you have an older Macintosh and the stimuli are in **My disk:My experiment:Stimuli**, you write

```
fileNameHead = "My disk:My experiment:Stimuli:"
```

But relative file paths will usually be preferred: they are more *portable*. The advantage of using relative file paths is that you can move your whole experiment (experiment file plus sounds) from one computer to another without changing the experiment file, as long as you put the experiment file in the same directory as where you put the directory **Sounds**. Or you can put the whole experiment on a CD and run the experiment directly from the CD. Since Praat supports the forward slash "/" as a directory separator on all computers, you can run the exact same experiment on Macintosh, Windows and Unix computers, independent of the type of computer where you have created your experiment.

2. The carrier phrase

The stimuli can be embedded in a carrier phrase. In the example, we see that the stimulus will be inserted between the sounds in the files **weSayTheWord.wav** and **again.wav**, both of which are expected to be in the directory **Sounds**. If you do not want a carrier phrase, you do:

```
carrierBefore = ""  
carrierAfter = ""
```

If you want only an introductory phrase before the stimulus, and nothing after the stimulus, you do something like

```
carrierBefore = "listenTo"  
carrierAfter = ""
```

and of course you supply the file **listenTo.wav** in the directory **Sounds**.

If you want to have a short silence before every stimulus (and before the carrier phrase), you supply a non-zero *initialSilenceDuration*, as in the example.

Since the carrier phrase is concatenated with the stimulus before it is played, it should have the same sampling frequency as the stimulus files.

3. Breaks

A new stimulus will arrive as soon as the listener makes her choice. To allow her some breathing time, you can insert a break after every so many trials. In the example, *breakAfterEvery* is 0, because there are only 12 trials. A typical experiment has perhaps 180 trials, and you may want to insert a break after every 40 trials. In that case, set *breakAfterEvery* to 40.

4. Randomization strategies

The 3 times 4 stimuli will have to be played in a certain order. For the least random order, you say

```
randomize = <CyclicNonRandom>
```

In this case, the stimuli will be played in the order in which they were specified in the file, 3 times:

```
heed hid hood hud heed hid hood hud heed hid hood hud
```

The most likely case in which you would want to use this randomization strategy, is if you have, say, 120 different stimuli and you want to play them only once (*numberOfReplicationsPerStimulus* = 1) in a fixed order.

The other extreme, the most random order, is

```
randomize = <WithReplacement>
```

In this case, a stimulus will be chosen at random 12 times without memory, for instance

```
hid hood hood heed hid hood hud hud hid hood heed hid
```

The order will probably differ for each listener. In this example, *hood* and *hid* occur four times each, *heed* and *hud* only twice each. This strategy is too random for most experiments. Usually, you will want to have the same number of replications of each stimulus. The most random way to do this is

```
randomize = <PermuteAll>
```

In this case, all stimuli will be played exactly 3 times, for instance

```
heed hood hud hud hid heed heed hud hood hid hid hood
```

Quite often, you will want a less random order, namely one in which the 12 trials are divided into 3 blocks of 4 stimuli. Within each block, all 4 different stimuli occur in a random order:

```
randomize = <PermuteBalanced>
```

In this case, each stimulus occurs exactly once within each block:

```
heed hood hud hid hood hud hid heed heed hud hood hid
```

This strategy ensures a certain spreading of the stimuli over the sequence of 12 trials. As we see here, it is still possible that the same stimulus (*heed*) occurs twice in a row, namely as the last stimulus of the second block and the first stimulus of the third. If you want to prevent that situation, you use

```
randomize = <PermuteBalancedNoDoublets>
```

This will ensure that the same stimulus is never applied twice in a row:

heed hood hud hid hood hud hid heed hud heed hood hid

This randomization strategy is used in our example, and advised for most listening experiments in which you want to minimize effects of stimulus order.

The randomization procedure does not interfere in any way with the breaks. The order is determined before any breaks are inserted.

5. Instructions

Before the experiment begins, the listener will see the *startText* in the centre of the screen. During each trial, she will see the *runText* at the top of the screen. During breaks, she will see the *pauseText* in the centre of the screen. After all the trials have been performed, she will see the *endText*.

6. Response categories

Every trial comes with the same set of response categories. The example has five of them. For each response category, you supply the area of the screen where a rectangle will be drawn. The whole screen measures from 0.0 (left) to 1.0 (right) and from 0.0 (bottom) to 1.0 (top). Thus, "0.2 0.3 0.7 0.8" means that a rectangle will be drawn somewhere in the top left quadrant of the screen. You also supply the text that will be drawn in this rectangle, for instance the text "h I d".

The second text that you supply for every response is a response key on the keyboard. In the above example this is "", i.e. the subject cannot press a key as a response. If you want the user to be able to press the "i" key instead of clicking in the "h I d" rectangle, the line in the experiment file would be:

```
0.2 0.3 0.7 0.8 "h I d" "i" "i"
```

The third text that you supply is the response category as it will be reported by Praat to you when the user clicks it, e.g. the text "i". If you want Praat to ignore mouse clicks on this rectangle, specify an empty response category, i.e. "".

The border of the rectangles will be yellow, the background of the screen will be light grey. The colour of clickable rectangles will be yellow, that of non-clickable rectangles (those with empty category specifications) light grey.

7. Goodness judgments

If *numberOfGoodnessCategories* is not 0, some more rectangles will be drawn, as in the example. You specify again the locations of these rectangles (in the example, they touch each other), and the texts on them. Praat will record the number of the button when the listener clicks on it. Thus, if she clicks on the button labelled "1 (poor)", Praat will record a goodness judgment of 1, because this is the first button in the list. If she clicks on "5 (good)", Praat will record a goodness judgment of 5.

8. How an experiment proceeds

A text file with an ExperimentMFC object can be read into Praat with Read from file... (it is not a script but a data file, so do not try to read it with **Open Praat script...**). You can then choose **Run**. After the experiment finishes, you can close the experiment window and choose **Extract results**. The resulting ResultsMFC object contains for each trial the stimulus name (e.g. "hood"), the response category (e.g. "u"), and the goodness judgment (e.g. 4). You will want to save this ResultsMFC object to a text file with Write to text file.... You may want to call these text files by the names of the subjects, e.g. **ts.ResultsMFC** and **mj.ResultsMFC**. Once you have collected the results of all your subjects, you can read all the results files into Praat with Read from file..., then select all the resulting ResultsMFC objects (which will have automatically been named **ts**, **mj**, and so on), then choose **Collect to table**. This will result in a table whose first column contains the names of the subjects, while the second column contains the stimulus names and the third column the responses (if there are goodness judgments, these will go into a fourth column). The table can be saved as a table file (with **Write to table file...**), which can be read by programs like Excel and SPSS.

9. A simple discrimination experiment

The example above was an *identification* experiment: the subject had identify a single sound as one element of a set of categories. Phoneticians will often do *discrimination* experiments, which are experiments in which a stimulus consists of multiple sub-stimuli played in sequence, and the subject has to judge the similarity between these sub-stimuli.

The simplest discrimination task has only two sub-stimuli, and the subject has to say whether these are the *same* or *different*. Suppose you have vowel-like sounds along an F1 continuum with seven steps, say 300, 320, 340, 360, 380, 400, and 420 Hertz, and you are interested in knowing how well the listeners can distinguish these. As your stimuli, you create pairs of these sounds, separated by 0.8 seconds of silence. It is important to include stimuli in which the sounds are identical, e.g. stimuli in which both sounds have an F1 of 340 Hz (see the literature on signal detection theory). Since sounds that are very different acoustically will always be heard as different, you do not include pairs in which the distance is larger than 60 Hz. The experiment file will look like this:

```
"ooTextFile"  
"ExperimentMFC 3"  
"stimuli/" ".wav"  
carrier phrase "" ""  
initial silence 0.5 seconds  
inter-stimulus interval 0.8 seconds  
37 different stimuli  
  "300,300" "300,320" "300,340" "300,360"  
  "320,300" "320,320" "320,340" "320,360" "320,380"  
  "340,300" "340,320" "340,340" "340,360" "340,380" "340,400"  
  "360,300" "360,320" "360,340" "360,360" "360,380" "360,400"  
"360,420"  
  "380,320" "380,340" "380,360" "380,380" "380,400" "380,420"  
  "400,340" "400,360" "400,380" "400,400" "400,420"  
  "420,360" "420,380" "420,400" "420,420"
```

```

10 replications per stimulus
break after every 50 stimuli
<PermuteBalancedNoDoublets>
"Click to start."
"Say whether these sounds were the same or different."
"You can have a short break if you like. Click to proceed."
"The experiment has finished. Call the experimenter."
2 response categories
    0.1 0.4 0.35 0.65 "same" "" "same"
    0.6 0.9 0.35 0.65 "different" "" "different"
0 goodness categories

```

In this example, the subject will have to click 370 times. After every 50 times, she will have the opportunity to sip her tea. A 0.5-second silence is played before every stimulus, so that the listener will not hear the stimulus immediately after her mouse click.

The experimenter does not have to create the stimulus pairs as sound files. You can specify multiple sound files by separating them with commas. Thus, "320,300" means that Praat will play the files **320.wav** and **300.wav**. These two substimuli will be separated here by a silence of 0.8 seconds, called the *inter-stimulus interval*.

Note that the text in this file is rather different from the previous example. It does not matter whether you write "numberOfDifferentStimuli", or "different stimuli", or anything else; Praat ignores these texts as long as they do not contain numbers, quoted strings, or things between <>.

10. An AXB discrimination experiment

In the AXB task, the subject will hear three stimuli in sequence, and has to say whether the second (X) is more similar to the first (A) or to the second (B). An experiment file could look like follows:

```

"ooTextFile"
"ExperimentMFC 3"
"stimuli/" ".wav"
carrier "" ""
initial silence 0.5
inter-stimulus interval 0.3
100 stimuli
    "300,300,320" "300,320,340" "300,340,340" "300,340,360"
    ...
    (and 96 more triplets of substimuli)
    ...
4 replications
break every 50
<PermuteBalancedNoDoublets>
"Click to start."
"Say whether the second sound is more similar to the first or to the
third."

```

```

"You can have a short break if you like. Click to proceed."
"The experiment has finished."
3 response categories
  0.1 0.3 0.4 0.6 "first" "" "A"
  0.4 0.6 0.4 0.6 "second" "" ""
  0.7 0.9 0.4 0.6 "third" "" "B"
0 goodness categories

```

In this example, the subject has to click 400 times. She sees three buttons, labelled *first*, *second*, and *third*, but the second button (the one with the empty response category) is not clickable: it has a light grey rather than a yellow border and cannot be chosen by the subject. In your ResultsMFC object, you will only see *A* and *B* responses.

11. A 4I-odddity experiment

In the four-items-odddity task, the subject will hear four stimuli in sequence, and has to say whether the second or the third is the odd one out. The other three substimuli are identical. An experiment file could look as follows:

```

"ooTextFile"
"ExperimentMFC 3"
"stimuli/" ".wav"
carrierBefore = ""
carrierAfter = ""
initialSilenceDuration = 0.5 seconds
interStimulusInterval = 0.3 seconds
numberOfDifferentStimuli = 60
  "300,300,320,300" "300,320,300,300"
  "300,300,340,300" "300,340,300,300"
  "300,300,360,300" "300,360,300,300"
  "320,320,300,320" "320,300,320,320"
  "320,320,340,320" "320,340,320,320"
  "320,320,360,320" "320,360,320,320"
  "320,320,380,320" "320,380,320,320"
  "340,340,300,340" "340,300,340,340"
  "340,340,320,340" "340,320,340,340"
  "340,340,360,340" "340,360,340,340"
  "340,340,380,340" "340,380,340,340"
  "340,340,400,340" "340,400,340,340"
  "360,360,300,360" "360,300,360,360"
  "360,360,320,360" "360,320,360,360"
  "360,360,340,360" "360,340,360,360"
  "360,360,380,360" "360,380,360,360"
  "360,360,400,360" "360,400,360,360"
  "360,360,420,360" "360,420,360,360"
  "380,380,320,380" "380,320,380,380"
  "380,380,340,380" "380,340,380,380"

```

```

"380,380,360,380" "380,360,380,380"
"380,380,400,380" "380,400,380,380"
"380,380,420,380" "380,420,380,380"
"400,400,340,400" "400,340,400,400"
"400,400,360,400" "400,360,400,400"
"400,400,380,400" "400,380,400,400"
"400,400,420,400" "400,420,400,400"
"420,420,360,420" "420,360,420,420"
"420,420,380,420" "420,380,420,420"
"420,420,400,420" "420,400,420,420"
numberOfReplicationsPerStimulus = 5
breakAfterEvery = 40
randomize = <PermuteBalancedNoDoublets>
startText = "Click to start."
runText = "Say whether the second or the third sound is different from
the rest."
pauseText = "You can have a short break if you like. Click to proceed."
endText = "The experiment has finished."
numberOfResponseCategories = 4
    0.04 0.24 0.4 0.6 "first" "" ""
    0.28 0.48 0.4 0.6 "second" "" "2"
    0.52 0.72 0.4 0.6 "third" "" "3"
    0.76 0.96 0.4 0.6 "fourth" "" ""
numberOfGoodnessCategories = 0

```

In this example, the subject has to click 300 times. She sees four buttons, but the first and fourth buttons cannot be chosen. In your ResultsMFC object, you will only see the responses 2 and 3.

12. Further tricks

Praat only supports a fixed inter-stimulus interval, but sometimes you may want to test discrimination as a function of the inter-stimulus interval itself. You can achieve this by supplying an *interStimulusInterval* of 0 and using sound files with various silences:

```
"300,silence0.5,320" "300,silence1.5,320" "300,silence4.5,320"
```

In this example, you have to supply the sound files **silence0.5.wav** and so on. You can create them with the help of Create Sound... (supply a *formula* of 0).

Links to this page

- [Intro](#)
- [What's new?](#)

© *ppgb*, July 9, 2004

Sound files

This tutorial describes the sound files that you can read and write with PRAAT. It assumes you are familiar with the Intro.

You can read this tutorial sequentially with the help of the "< 1" and "1 >" buttons.

1. General structure
 - 1.1. Sampling (sampling frequency)
 - 1.2. Quantization (linear, endian, μ -law, A-law)
 - 1.3. Channels (mono, stereo)
 - 1.4. The header
 - 1.5. Size
 - 1.6. Compression
2. File types
 - 2.1. WAV files
 - 2.2. AIFF files
 - 2.3. AIFC files
 - 2.4. NeXT/Sun (.au) files
 - 2.5. NIST files
3. Files that Praat can read
4. Files that Praat can write

Links to this page

- [Intro 2.1. Writing a sound to disk](#)
- [LongSound](#)
- [Sound](#)
- [Types of objects](#)
- [Write to AIFC file...](#)
- [Write to AIFF file...](#)
- [Write to NeXT/Sun file...](#)
- [Write to NIST file...](#)
- [Write to WAV file...](#)

© ppgb, February 23, 2004

Filtering

This tutorial describes the use of filtering techniques in PRAAT. It assumes you are familiar with the Intro.

Frequency-domain filtering

Modern computer techniques make possible an especially simple batch filtering method: multiplying the complex spectrum in the frequency domain by any real-valued filter function. This leads to a zero phase shift for each frequency component. The impulse response is symmetric in the time domain, which also means that the filter is *acausal*: the filtered signal will show components before they start in the original.

- Spectrum: Filter (pass Hann band)...
- Spectrum: Filter (stop Hann band)...
- Sound: Filter (pass Hann band)...
- Sound: Filter (stop Hann band)...
- Sound: Filter (formula)...

Spectro-temporal:

- band filtering in the frequency domain

Fast time-domain filtering

Some very fast Infinite Impulse Response (IIR) filters can be defined in the time domain. These include recursive all-pole filters and pre-emphasis. These filters are causal but have non-zero phase shifts. There are versions that create new Sound objects:

- Sound: Filter (one formant)...
- Sound: Filter (pre-emphasis)...
- Sound: Filter (de-emphasis)...

And there are in-line versions, which modify the existing Sound objects:

- Sound: Filter with one formant (in-line)...
- Sound: Pre-emphasize (in-line)...
- Sound: De-emphasize (in-line)...

Convolution

A Finite Impulse Response (FIR) filter can be described as a sampled sound. Filtering with such a filter amounts to a *convolution* of the original sound and the filter:

- Sounds: Convolve

Described elsewhere

Described in the Source-filter synthesis tutorial:

- [Sound & Formant: Filter](#)
- [Sound & FormantTier: Filter](#)
- [LPC & Sound: Filter...](#)
- [LPC & Sound: Filter \(inverse\)](#)

Links to this page

- [Sound](#)
 - [Types of objects](#)
-

© *ppgb*, March 16, 2003

Source-filter synthesis

This tutorial describes how you can do acoustic synthesis with PRAAT. It assumes that you are familiar with the Intro.

1. The source-filter theory of speech production

The source-filter theory hypothesizes that an acoustic speech signal can be seen as a *source* signal (the glottal source, or noise generated at a constriction in the vocal tract), *filtered* with the resonances in the cavities of the vocal tract downstream from the glottis or the constriction.

In the Praat program, you can create a *source* signal from an existing speech signal or from scratch, and you can extract a *filter* from an existing speech signal or from scratch. You can manipulate (change, adapt) both the source and the filter before doing the actual synthesis, which combines the two.

2. How to extract the *filter* from an existing speech sound

You can separate source and filter with the help of the technique of *linear prediction* (see Sound: LPC analysis). This technique tries to approximate a given frequency spectrum with a small number of peaks, for which it finds the mid frequencies and the bandwidths. If we do this for an overlapping sequence of windowed parts of a sound signal (i.e. a *short-term analysis*), we get a quasi-stationary approximation of the signal's spectral characteristics as a function of time, i.e. a smoothed version of the Spectrogram.

For a speech signal, the peaks are identified with the resonances (*formants*) of the vocal tract. Since the spectrum of a vowel spoken by an average human being falls off with approximately 6 dB per octave, *pre-emphasis* is applied to the signal before the linear-prediction analysis, so that the algorithm will not try to match only the lower parts of the spectrum.

For an average (i.e. adult female) human voice, tradition assumes five formants in the range between 0 and 5500 Hertz. This number comes from a computation of the formants of a straight tube, which has resonances at wavelengths of four tube lengths, four thirds of a tube length, four fifths, and so on. For a straight tube 16 centimetres long, the shortest wavelength is 64 cm, which, with a sound velocity of 352 m/s, means a resonance frequency of $352/0.64 = 550$ Hertz. The other resonances will be at 1650, 2750, 3850, and 4950 Hertz. For the linear prediction in Praat, you will have to implement this 5500-Hz band-limiting by resampling the original speech signal to 11 kHz. For a male voice, you would use 10 kHz; for a young child, 20 kHz.

To perform the resampling, you use Sound: Resample...: you select a Sound object, and click **Resample...** In the rest of this tutorial, I will use the syntax that you would use in a script, though you will usually do these things by clicking on objects and buttons. Thus:

```
select Sound hallo
Resample... 11000 50
```

You can then perform a linear-prediction analysis on the resampled sound with Sound: To LPC (burg)...:

```
select Sound hallo_11000
To LPC (burg)... 10 0.025 0.005 50
```

This says that your analysis is done with 10 linear-prediction parameters (which will yield at most five formant-bandwidth pairs), with an analysis window effectively 25 milliseconds long, with time steps of 5 milliseconds (so that the windows will appreciably overlap), and with a pre-emphasis frequency of 50 Hz (which is the point above which the sound will be amplified by 6 dB/octave prior to the analysis proper).

As a result, an object called "LPC hallo" will appear in the list of objects. This LPC object is a time function with 10 *linear-prediction coefficients* in each *time frame*. These coefficients are rather opaque even to the expert (try to view them with Inspect), but they are the raw material from which formant and bandwidth values can be computed. To see the smoothed Spectrogram associated with the LPC object, choose LPC: To Spectrogram...:

```
select LPC hallo_11000
To Spectrogram... 20 0 50
Paint... 0 0 0 0 50 0 0 yes
```

Note that when drawing this Spectrogram, you will want to set the pre-emphasis to zero (the fifth 0 in the last line), because pre-emphasis has already been applied in the analysis.

You can get and draw the formant-bandwidth pairs from the LPC object, with LPC: To Formant and Formant: Speckle...:

```
select LPC hallo_11000
To Formant
Speckle... 0 0 5500 30 yes
```

Note that in converting the LPC into a Formant object, you may have lost some information about spectral peaks at very low frequencies (below 50 Hz) or at very high frequencies (near the Nyquist frequency of 5500 Hz. Such peaks usually try to fit an overall spectral slope (if the 6 dB/octave model is inappropriate), and are not seen as related with resonances in the vocal tract, so they are ignored in a formant analysis. For resynthesis purposes, they might still be important.

Instead of using the intermediate LPC object, you could have done a formant analysis directly on the original Sound, with Sound: To Formant (burg)...:

```
select Sound hallo
To Formant (burg)... 0.005 5 5500 0.025 50
```

A Formant object has a fixed sampling (time step, frame length), and for every *formant frame*, it contains a number of formant-bandwidth pairs.

From a Formant object, you can create a FormantTier with Formant: Down to FormantTier. A FormantTier object contains a number of time-stamped *formant points*, each with a number of formant-bandwidth pairs.

Any of these three classes (LPC, Formant, and FormantTier) can represent the *filter* in source-filter synthesis.

3. How to extract the *source* from an existing speech sound

If you are only interested in the *filter* characteristics, you can get by with Formant objects. To get at the *source* signal, however, you need the raw LPC object: you select it together with the resampled Sound, and apply *inverse filtering*:

```
select Sound hallo_11000
plus LPC hallo_11000
Filter (inverse)
```

A new Sound named "hallo_11000" will appear in the list of objects (you could rename it to "source"). This is the estimated source signal. Since the LPC analysis was designed to yield a spectrally flat filter (through the use of pre-emphasis), this source signal represents everything in the speech signal that cannot be attributed to the resonating cavities. Thus, the "source signal" will consist of the glottal volume-velocity source (with an expected spectral slope of -12 dB/octave for vowels) and the radiation characteristics at the lips, which cause a 6 dB/octave spectral rise, so that the resulting spectrum of the "source signal" is actually the *derivative* of the glottal flow, with an expected spectral slope of -6 dB/octave.

Note that with inverse filtering you cannot measure the actual spectral slope of the source signal. Even if the actual slope is very different from -6 dB/octave, formant extraction will try to match the pre-emphasized spectrum. Thus, by choosing a pre-emphasis of -6 dB/octave, you *impose* a slope of -6 dB/octave on the source signal.

4. How to do the synthesis

You can create a new Sound from a source Sound and a filter, in at least four ways.

If your filter is an LPC object, you select it and the source, and choose LPC & Sound: Filter...:

```
select Sound source
plus LPC filter
Filter... no
```

If you had computed the source and filter from an LPC analysis, this procedure should give you back the original Sound, except that windowing has caused 25 milliseconds at the beginning and end of the signal to be set to zero.

If your filter is a Formant object, you select it and the source, and choose Sound & Formant: Filter:

```
select Sound source
plus Formant filter
Filter
```

If you had computed the source and filter from an LPC analysis, this procedure will not generally give you back the original Sound, because some linear-prediction coefficients will have been ignored in the conversion to formant-bandwidth pairs.

If your filter is a FormantTier object, you select it and the source, and choose Sound & FormantTier:
Filter:

```
select Sound source
plus FormantTier filter
Filter
```

Finally, you could just know the *impulse response* of your filter (in a Sound object). You then select both Sound objects, and choose Sounds: Convolve:

```
select Sound source
plus Sound filter
Convolve
```

5. How to manipulate the filter

You can hardly change the values in an LPC object in a meaningful way: you would have to manually change its rather opaque data with the help of Inspect.

A Formant object can be changed in a friendlier way, with Formant: Formula (frequencies)... and Formant: Formula (bandwidths).... For instance, to multiply all formant frequencies by 0.9, you do

```
select Formant filter
Formula (frequencies)... self * 0.9
```

To add 200 Hertz to all values of F_2 , you do

```
Formula (frequencies)... if row = 2 then self + 200 else self fi
```

A FormantTier object can be changed by adding or removing points:

- FormantTier: Add point...
- Remove point...
- Remove point near...
- Remove points between...

6. How to manipulate the source signal

You can manipulate the source signal in the same way you that would manipulate any sound, for instance with the ManipulationEditor.

7. How to create a filter from scratch

You can create a FormantTier object with Create FormantTier..., and add some points to it with FormantTier: Add point...:

```
Create FormantTier... filter 0 0.5
Add point... 0.00 100 50 500 100 2500 200 3600 300 4700 400
Add point... 0.05 700 50 1100 100 2500 200 3600 300 4700 400
```

This creates a spectral specification whose F_1 rises from 100 to 700 Hertz during the first 50 milliseconds (as for any obstruent), and whose F_2 rises from 500 to 1100 Hertz. This may be a [ba]-like formant transition.

8. How to create a source signal from scratch

It is easy to create a pulse train: use Create PitchTier... and PitchTier: Add point..., for instance:

```
Create PitchTier... filter 0 0.5
Add point... 0 300
Add point... 0.5 200
```

The resulting PitchTier falls linearly from 300 to 200 Hz during its time domain.

Form this PitchTier, you can create a PointProcess with PitchTier: To PointProcess. The resulting PointProcess can represent a series of glottal pulses. To make some parts of this point process voiceless, you can use PointProcess: Remove points between....

To create the pulse-train source signal, you use PointProcess: To Sound (pulse train)....

You are then ready to create the acoustic result with Sound & FormantTier: Filter.

The resulting sound will have fairly straight intensity contour. You can change it with the **Formula** command (Sound: Formula...), or by multiplying the source signal or the acoustic result with an Intensity or IntensityTier object. To get the spectral slope at -6 dB/octave, you may need to use Sound: De-emphasize (in-line)....

9. Example: a ba-da continuum

We are going to create a [ba]-[da] continuum in ten steps. The acoustic difference between [ba] and [da] is the initial F_2 , which is 500 Hz for [ba], and 2500 Hz for [da].

We use the same PitchTier throughout, to model a falling intonation contour:

```
Create PitchTier... f0 0.00 0.50
Add point... 0.00 300
Add point... 0.50 200
```

The first and last 50 milliseconds are voiceless:

```
To PointProcess
Remove points between... 0.00 0.05
Remove points between... 0.45 0.50
```

Generate the pulse train:

```
To Sound (pulse train)... 22050 1 0.05 300
```

During the labial or coronal closure, the sound is almost silent, so we use an IntensityTier that models this:

```
Create IntensityTier... intens 0.00 0.50
Add point... 0.05 60
Add point... 0.10 80
```

Generate the source signal:

```
plus Sound f0
Multiply
Rename... source
```

The filters will be spectrally flat, and the source is also spectrally flat, so in order to end up with a natural spectral slope of -6 dB/octave, we de-emphasize the source signal:

```
De-emphasize (in-line)... 50
```

The ten sounds are generated in a loop:

```
for i from 1 to 10
  f2_locus = 500 + (2500/9) * (i - 1) ; variable names start with
  lower case!
  Create FormantTier... filter 0.00 0.50
  Add point... 0.05 100 50 'f2_locus' 100
  ... 3000 300 4000 400 5000 500
  Add point... 0.10 700 50 1100 100
  ... 3000 300 4000 400 5000 500
  plus Sound source
  Filter (no scale)
  Rename... bada'i'
  select FormantTier filter
  Remove
endfor
```

Clean up:

```
select Sound source
plus Sound f0
plus IntensityTier intens
```

```
plus PointProcess f0
plus PitchTier f0
Remove
```

In this example, filtering was done without automatic scaling, so that the resulting signals have equal intensities in the areas where they have equal formants. You will probably want to multiply all these signals with the same value in order to bring their amplitudes in a suitable range between -1 and +1 Pascal.

Links to this page

- [Create IntensityTier...](#)
- [Filtering](#)
- [Intro 8.4. Manipulation of formants](#)
- [Sound & Formant: Filter \(no scale\)](#)
- [Sound & FormantTier: Filter \(no scale\)](#)
- [Sound: Filter \(de-emphasis\)...](#)
- [Types of objects](#)
- [What's new?](#)

© *ppgb*, March 28, 2004

Articulatory synthesis

This is a description of the articulatory synthesis package in PRAAT. For a detailed description of the physics and mathematics behind the model, see Boersma (1998), chapters 2 and 3. For examples of how to synthesize utterances, consult <http://www.fon.hum.uva.nl/paul/diss/ch5/>.

How to start (after reading the Intro)

We are going to have the synthesizer say [apa]. We need a Speaker and an Artword object.

1. Create a speaker with Create Speaker... from the New menu.
2. Create an articulation word of 0.5 seconds with Create Artword...
3. Edit the Artword by selecting it and clicking **Edit**.
4. To set the glottis to a position suitable for phonation, use the ArtwordEditor to set the *Interarytenoid* activity to 0.5 throughout the utterance. You set two targets: 0.5 at a time of 0 seconds, and 0.5 at a time of 0.5 seconds.
5. To prevent air escaping from the nose, close the nasopharyngeal port by setting the *LevatorPalatini* activity to 1.0 throughout the utterance.
6. To generate the lung pressure needed for phonation, you set the *Lungs* activity at 0 seconds to 0.2, and at 0.1 seconds to 0.
7. To force a jaw movement that closes the lips, set the *Masseter* activity at 0.25 seconds to 0.7, and the *OrbicularisOris* activity at 0.25 seconds to 0.2.
8. Select the Speaker and the Artword and click **Movie**; you will see a closing-and-opening gesture of the mouth.
9. Select the Speaker and the Artword and click **To Sound...** (see Artword & Speaker: To Sound...).
10. Just click *OK*; the synthesis starts.
11. If you are sitting at a 1997 computer, this will last for 5 minutes or so. If this is too slow for you, click *Interrupt*. Otherwise, you can watch the vibrating vocal cords and the changing vocal-tract shape.
12. You can play, view, and analyse the resulting Sound as you would any other. You can see and hear a sound movie if you select the Speaker, the Artword, and the Sound, and click **Play**.

Links to this page

- [Types of objects](#)
-

© *ppgb*, March 16, 2003

OT learning

This tutorial describes how you can draw Optimality-Theoretic tableaux and simulate Optimality-Theoretic learning with PRAAT.

You can read this tutorial sequentially with the help of the "< 1" and "1 >" buttons.

1. Kinds of OT grammars (ordinal and stochastic, OTGrammar)
2. The grammar
 - 2.1. Viewing a grammar (NOCODA example, OTGrammarEditor)
 - 2.2. Inside the grammar (saving, inspecting)
 - 2.3. Defining your own grammar
 - 2.4. Evaluation (noise)
 - 2.5. Editing a grammar
 - 2.6. Variable output (place assimilation example)
 - 2.7. Tableau pictures (printing, EPS)
 - 2.8. Asking for one output
 - 2.9. Output distributions
3. Generating language data
 - 3.1. Data from a pair distribution
 - 3.2. Data from another grammar (tongue-root-harmony example)
4. Learning an ordinal grammar
5. Learning a stochastic grammar
6. Shortcut to OT learning
7. Learning from overt forms

Links to this page

- [Constraints](#)
- [Create tongue-root grammar...](#)
- [Intro](#)
- [New menu](#)
- [Optimality Theory](#)
- [OTAnyGrammar](#)
- [OTAnyGrammar & 2 Strings: Learn \(GLA\)...](#)
- [OTAnyGrammar & 2 Strings: Learn \(T&S\)](#)
- [OTAnyGrammar examples](#)
- [OTAnyGrammar: Generate inputs...](#)
- [OTAnyGrammar: Learn one \(GLA\)...](#)
- [OTAnyGrammar: Learn one \(T&S\)...](#)
- [OTAnyGrammarEditor](#)
- [OTGrammar_tongueRoot](#)
- [Types of objects](#)
- [What's new?](#)

© *ppgb*, March 16, 2003

Principal component analysis

This tutorial describes how you can perform principal component analysis with the PRAAT.

Principal component analysis (PCA) involves a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called *principal components*. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

Traditionally, principal component analysis is performed on a square symmetric matrix of type SSCP (pure sums of squares and cross products), Covariance (scaled sums of squares and cross products), or Correlation (sums of squares and cross products from standardized data). The analysis results for objects of type SSCP and Covariance do not differ, since these objects only differ in a global scaling factor. A Correlation object has to be used if the variances of individual variates differ much, or if the units of measurement of the individual variates differ.

The result of a principal component analysis on such objects will be a new object of type PCA.

1. Objectives of principal component analysis

- To discover or to reduce the dimensionality of the data set.
- To identify new meaningful underlying variables.

2. How to start

We assume that the multi-dimensional data have been collected in a TableOfReal object which is a rectangular matrix with (optional) row and column labels.

If the variances of the individual columns in the TableOfReal differ much or the measurement units of the columns differ then you should first standardize the data. You can do this by choosing Standardize columns. Performing a principal component analysis on a standardized data matrix has the same effect as performing the analysis on the correlation matrix (the covariance matrix from standardized data is equal to the correlation matrix of these data).

Select the TableOfReal object from the list of objects and choose To PCA. This results in a new PCA object in the list of objects.

We can now make a scree plot of the eigenvalues, Draw eigenvalues... to get an indication of the importance of each eigenvalue. The exact contribution of each eigenvalue (or a range of eigenvalues) to the "explained variance" can also be queried: Get fraction variance accounted for.... You might also check for the equality of a number of eigenvalues: Get equality of eigenvalues....

3. Determining the number of components

There are two methods to help you to choose the number of components. Both methods are based on relations between the eigenvalues.

- Plot the eigenvalues, Draw eigenvalues...
- If the points on the graph tend to level out (show an "elbow"), these eigenvalues are usually close enough to zero that they can be ignored.
- Limit variance accounted for, Get number of components (VAF)....

4. Getting the principal components

Principal components are obtained by projecting the multivariate datavectors on the space spanned by the eigenvectors. This can be done in two ways:

1. Directly from the TableOfReal without first forming a PCA object: To Configuration (pca).... You can then draw the Configuration or display its numbers.
2. Select a PCA and a TableOfReal object together and choose To Configuration.... In this way you project the TableOfReal onto the PCA's eigenspace.

5. Mathematical background on principal component analysis

The mathematical technique used in PCA is called eigen analysis: we solve for the eigenvalues and eigenvectors of a square symmetric matrix with sums of squares and cross products. The eigenvector associated with the largest eigenvalue has the same direction as the first principal component. The eigenvector associated with the second largest eigenvalue determines the direction of the second principal component. The sum of the eigenvalues equals the trace of the square matrix and the maximum number of eigenvectors equals the number of rows (or columns) of this matrix.

6. Algorithms

If our matrix happens to be symmetric, with some sort of sums of squares and cross products, we solve for the eigenvalue and eigenvectors by first performing a Householder reduction to tridiagonal form, followed by the QL algorithm with implicit shifts.

If, conversely, our starting point is a non-square or non-symmetric data matrix A (that could originate from a TableOfReal), we do not have to form explicitly the matrix with sums of squares and cross products, $A'A$. Instead, we proceed by a numerically more stable method, and form the singular value decomposition $U\Sigma V'$ of A . The matrix V then contains the eigenvectors, and the squared diagonal elements of Σ contain the eigenvalues.

Links to this page

- Configuration
- Intro
- Statistics

- Types of objects
 - What's new?
-

© *djmw*, March 23, 1999

Multidimensional scaling

This tutorial describes how you can use PRAAT to perform **MultiDimensional Scaling (MDS)** analysis.

MDS helps us to represent *dissimilarities* between objects as *distances* in a *Euclidean space*. In effect, the more dissimilar two objects are, the larger the distance between the objects in the Euclidean space should be. The data types in PRAAT that incorporate these notions are Dissimilarity, Distance and Configuration.

In essence, an MDS-analysis is performed when you select a Dissimilarity object and choose one of the **To Configuration (xxx)...** commands to obtain a Configuration object. In the above, method (xxx) represents one of the possible multidimensional scaling models.

MDS-analysis

Let us first create a Dissimilarity object. You can for example create a Dissimilarity object from a file. Here we will use the Dissimilarity object from the letter R example. We have chosen the default value (32.5) for the (uniform) noise range. Note that this may result in substantial distortions between the dissimilarities and the distances.

Now you can do the following, for example:

Select the Dissimilarity and choose To Configuration (monotone mds)..., and you perform a kruskal-like multidimensional scaling which results in a new Configuration object. (This Configuration could subsequently be used as the starting Configuration for a new MDS-analysis!).

Select the Configuration and choose Draw... and the following picture will result

[sorry, no pictures yet in the web version of this manual]

The following script summarizes:

```
Create letter R example... 32.5
select Dissimilarity R
To Configuration (monotone mds)... 2 "Primary approach" 0.00001 50 1
Draw... 1 2 -0.8 1.2 -0.8 0.7 yes
```

Obtaining the stress value

Select the Dissimilarity and the Configuration together and query for the stress value with: Get stress (monotone mds)....

The following script summarizes:

```
select Dissimilarity R
plus Configuration R_monotone
Get stress (monotone mds)... "Primary approach" "Kruskals's stress-1")
```

The Shepard diagram

Select the Dissimilarity and the Configuration together to draw the Shepard diagram.

[sorry, no pictures yet in the web version of this manual]

The following script summarizes:

```
select Dissimilarity R
plus Configuration R_monotone
Draw Shepard diagram...0 200 0 2.2 1 + yes
```

The (monotone) regression

Select the Dissimilarity and the Configuration together to draw the monotone regression of distances on dissimilarities.

[sorry, no pictures yet in the web version of this manual]

The following script summarizes:

```
select Dissimilarity R
plus Configuration R_monotone
Draw monotone regresion... "Primary approach" 0 200 0 2.2 1 + yes
```

When you enter $noiseRange = 0$ in the form for the letter **R**, perfect reconstruction is possible. The Shepard diagram then will show a perfectly smooth monotonically increasing function.

Weighing the dissimilarities

When you can't have equal confidence in all the number in the Dissimilarity object, you can give different weights to these numbers by associating a Weight object with the Dissimilarity object. An easy way to do this is to select the Dissimilarity object and first choose To Weight. Then you might change the individual weights in the Weight object with the Set value... command (remember: make $w_{ij} = w_{ji}$).

The following script summarizes:

```
select Dissimilarity R
To Weight
! Change [i][j] and [j][i] cells in the Weight object
Set value... i j val
Set value... j i val
...
! now we can do a weighed analysis.
select Dissimilarity R
plus Weight R
To Configuration (monotone mds)... 2 "Primary approach" 0.00001 50 1
```

You can also query the stress values with three objects selected. The following script summarizes:

```
select Dissimilarity R
plus Weight R
plus Configuration R_s_monotone
Get stress (monotone mds)... "Primary approach" "Kruskals's stress-1")
```

Using a start Configuration

You could also use a Configuration object as a starting configuration in the minimization process. Lets assume that you are not satisfied with the stress value from the Configuration object that you obtained in the previous analysis. You can than use this Configuration object as a starting point for further analysis:

The following script summarizes:

```
select Dissimilarity R
plus Configuration R_monotone
plus Weight R
To Configuration (monotone mds)... 2 "Primary approach" 0.00001 50 1
```

Multiple Dissimilarity's (INDSCAL)

When you have multiple Dissimilarity objects you can also perform individual difference scaling (often called INDSCAL analysis).

As an example we can use an example taken from Carrol & Wish. Because INDSCAL only works on metrical data, we can not use Dissimilarity objects directly. We have to transform them first to Distance objects.

This type of analysis on multiple objects results in two new objects: a Configuration and a Saliense.

Links to this page

- [Intro](#)
- [Statistics](#)
- [Types of objects](#)
- [What's new?](#)

© *djmw*, April 9, 2001

Discriminant analysis

This tutorial will show you how to perform discriminant analysis with PRAAT

As an example, we will use the dataset from Pols et al. (1973) with the frequencies and levels of the first three formants from the 12 Dutch monophthongal vowels as spoken in /h_t/ context by 50 male speakers. This data set has been incorporated into PRAAT and can be called into play with the Create TableOfReal (Pols 1973)... command that can be found in the "New / TableOfReal" menu.

In the list of objects a new TableOfReal object will appear with 6 columns and 600 rows (50 speakers \times 12 vowels). The first three columns contain the formant frequencies in Hz, the last three columns contain the levels of the first three formants given in decibels below the overall sound pressure level of the measured vowel segment. Each row is labelled with a vowel label.

Pols et al. use logarithms of frequency values, we will too. Because the measurement units in the first three columns are in Hz and in the last three columns in dB, it is probably better to standardize the columns. The following script summarizes our achievements up till now:

```
Create TableOfReal (Pols 1973)... yes
Formula... if col < 4 then log10 (self) else self fi
Standardize columns
# change the column labels too, for nice plot labels.
Set column label (index)... 1 standardized log (%F__1_)
Set column label (index)... 2 standardized log (%F__2_)
Set column label (index)... 3 standardized log (%F__3_)
Set column label (index)... 4 standardized %L__1_
Set column label (index)... 5 standardized %L__2_
Set column label (index)... 6 standardized %L__3_
```

To get an indication of what these data look like, we make a scatter plot of the first standardized log-formant-frequency against the second standardized log-formant-frequency. With the next script fragment you can reproduce the following picture.

```
Viewport... 0 5 0 5
select TableOfReal pols_50males
Draw scatter plot... 1 2 0 0 -2.9 2.9 -2.9 2.9 no yes
```

[sorry, no pictures yet in the web version of this manual]

Apart from a difference in scale this plot is the same as fig. 3 in the Pols et al. article.

1. How to perform a discriminant analysis

Select the TableOfReal and choose from the dynamic menu the option To Discriminant. This command is available in the "Multivariate statistics" action button. The resulting Discriminant object will bear the same name as the TableOfReal object. The following script summarizes:

```
select TableOfReal pols_50males
To Discriminant
```

2. How to project data on the discriminant space

You select a TableOfReal and a Discriminant object together and choose: To Configuration.... One of the options of the newly created Configuration object is to draw it. The following picture shows how the data look in the plane spanned by the first two dimensions of this Configuration. The directions in this configuration are the eigenvectors from the Discriminant.

[sorry, no pictures yet in the web version of this manual]

The following script summarizes:

```
select TableOfReal pols_50males
plus Discriminant pols_50males
To Configuration... 0
Viewport... 0 5 0 5
Draw... 1 2 -2.9 2.9 -2.9 2.9 yes
```

If you are only interested in this projection, there also is a short cut without an intermediate Discriminant object: select the TableOfReal object and choose To Configuration (1da)....

3. How to draw concentration ellipses

Select the Discriminant object and choose Draw sigma ellipses.... In the form you can fill out the coverage of the ellipse by way of the *Number of sigmas* parameter. You can also select the projection plane. The next figure shows the 1- σ concentration ellipses in the standardized $\log F_1$ vs $\log F_2$ plane. When the data are multnormally distributed, a 1- σ ellipse will cover approximately 39.3% of the data. The following code summarizes:

```
select Discriminant pols_50males
Draw sigma ellipses... 1.0 no 1 2 -2.9 2.9 -2.9 2.9 yes
```

[sorry, no pictures yet in the web version of this manual]

4. How to classify

Select together the Discriminant object (the classifier), and a TableOfReal object (the data to be classified). Next you choose To ClassificationTable. Normally you will enable the option *Pool covariance matrices* and the pooled covariance matrix will be used for classification.

The ClassificationTable can be converted to a Confusion object and its fraction correct can be queried with: Confusion: Get fraction correct.

In general you would separate your data into two independent sets, TRAIN and TEST. You would use TRAIN to train the discriminant classifier and TEST to test how well it classifies. Several possibilities for splitting a dataset into two sets exist. We mention the jackknife ("leave-one-out") and the bootstrap

methods ("resampling").

Links to this page

- [Canonical correlation analysis](#)
- [Configuration](#)
- [Discriminant](#)
- [Feedforward neural networks 3. FFNet versus discriminant classifier](#)
- [Intro](#)
- [Statistics](#)
- [Types of objects](#)
- [What's new?](#)

© *djmw*, February 27, 1999

Printing

The best results will be obtained on PostScript printers, since these have built-in facilities for images (e.g. spectrograms) and rotated text. However, the printed page will look reasonable on colour inkjet printers as well.

1. Printing on Unix

Most Unix networks (i.e. SGI, Solaris, HP-UX) are traditionally connected to a PostScript printer. When you tell Praat to print a picture or manual page, Praat will write the picture to a temporary PostScript file and send this file to a printer with the *print command* (typically `lpr`), which you can change with PostScript settings....

On Linux, you do not need a PostScript printer to print PostScript directly, because the `lpr` program sends PostScript files through the GhostScript program, which is a part of all modern Linux distributions. The print command is typically `lpr %s`. By changing the print command (with PostScript settings...), you can change it to something fancier. For instance, if you want to save the woods and print two pages on one sheet of paper, you change it to `cat %s | mpage -2 -o -f -m0 | lpr`.

2. Printing on Macintosh

Many Macintoshes are in a network that includes a PostScript printer, typically a LaserWriter®. If a PostScript printer is available, Praat will usually write direct PostScript commands to that printer (see PostScript settings... if you want to switch this off). Non-PostScript printers, such as most colour inkjet printers, are also supported.

3. Printing on Windows

Some Windows computers are in a network that includes a PostScript printer. If a PostScript printer is available, Praat will usually write direct PostScript commands to that printer (see PostScript settings... if you want to switch this off). Non-PostScript printers, such as most colour inkjet printers, are also supported.

4. Indirect printing with GhostView

If you don't have a PostScript printer, and you still want PostScript quality, you can save the picture to an EPS file (Write to EPS file...). You can then view this file with the freely available GhostView™ program, which you can download from <http://www.cs.wisc.edu/~ghost>.

5. Indirect printing with your word processor

If you save your picture to an EPS file, you will be able to include it as a picture in your favourite word processor (Microsoft® Word™, LaTeX...). On Macintosh computers, you will even have a screen preview. The picture will print correctly on a PostScript printer.

If you don't have a PostScript printer, you could again use GhostView™ to print your document to any printer, after you printed your document to a PostScript file. You can do the same if you are the lucky owner of Adobe® Acrobat™ Distiller™, which is more reliable than GhostView but is also expensive.

6. Creating a PDF file

If you have Distiller or GhostView, you can print the entire Word™ or LaTeX document to a PostScript file, and convert this to a PDF file, which anyone in the world can view and print with the free Adobe® Acrobat™ Reader program. Note: when creating a PDF file on Windows if you have Acrobat, **do not use PDFWriter**, but choose Distiller as your printer, and retrieve the PDF file from **C:\Program Files\Adobe\Acrobat 4.0\PDF Outputs** or so; also, **do not use "Print to PDF"** from your Microsoft Word File menu; otherwise, your EPS files will not show in the PDF file. Similarly, when creating a PDF file on MacOS X, **do not use "Save as PDF..."** in your printing dialog, but choose Adobe PDF as your printer or save the document to a PostScript file and convert it with Distiller or GhostView; otherwise, your EPS files will not show in the PDF file.

Pictures included in your word processor on the Macintosh via Copy to clipboard or Write to Mac PICT file..., or on Windows via Copy to clipboard or Write to Windows metafile... will also print fine, though not as nicely as EPS files.

Links to this page

- [Intro](#)
- [Print...](#)

© ppgb, May 29, 2003

Scripting

This is one of the tutorials of the PRAAT program. It assumes you are familiar with the Intro.

A *script* is a text that consists of menu commands and action commands. If you *run* the script (perhaps from a ScriptEditor), the commands are executed as if you clicked on them.

You can read this tutorial sequentially with the help of the "< 1" and "1 >" buttons.

- Scripting 1. My first script (how to create, how to run)
- Scripting 2. Arguments to commands (numeric, check, radio, text, file)
- Scripting 3. Layout (white space, comments, continuation lines)
- Scripting 4. Object selection (selecting and querying)
- Scripting 5. Language elements
 - Scripting 5.1. Variables (numeric, string, copy, substitution)
 - Scripting 5.2. Formulas (numeric, string)
 - Scripting 5.3. Jumps (if, then, elsif, else, endif)
 - Scripting 5.4. Loops (for/endfor, while/endwhile, repeat/until)
 - Scripting 5.5. Procedures (call, procedure)
 - Scripting 5.6. Arrays
 - Scripting 5.7. Including other scripts
 - Scripting 5.8. Quitting (exit)
- Scripting 6. Communication outside the script
 - Scripting 6.1. Arguments to the script (form/endform, execute)
 - Scripting 6.2. Calling system commands (system, system_nocheck)
 - Scripting 6.3. Writing to the Info window (echo, print, printtab, printline)
 - Scripting 6.4. Query commands (Get, Count)
 - Scripting 6.5. Files (fileReadable, <, >, >>, filedelete, fileappend)
 - Scripting 6.6. Controlling the user (pause)
 - Scripting 6.7. Sending a message to another program (sendsocket)
 - Scripting 6.8. Error message to the user (exit)
 - Scripting 6.9. Calling from the command line
- Scripting 7. Scripting the editors
 - Scripting 7.1. Scripting an editor from a shell script (editor/endeditor)
 - Scripting 7.2. Scripting an editor from within
- Scripting 8. Controlling Praat from another program
 - Scripting 8.1. The sendpraat subroutine
 - Scripting 8.2. The sendpraat program
 - Scripting 8.3. The sendpraat directive

Also see the scripting examples.

Links to this page

- [Feedforward neural networks 1.1. The learning phase](#)
- [Formulas 1.9. Formulas in scripts](#)
- [History mechanism](#)
- [Intro 8.2. Manipulation of duration](#)
- [Praat script](#)
- [Programming with Praat](#)
- [What's new?](#)

© *ppgb*, May 28, 2003

Formulas 1.8. Formulas for modification

Analogously to the formulas that you can use for creating new objects (see the previous page), you can use formulas for modifying existing objects. You do this with the command *Formula...* that you can find in the **Modify** submenu when you select an object.

Modifying a Sound with a formula

Record a sound with your microphone and talk very lowly. If you don't know how to record a sound in Praat, consult the Intro. Once the Sound object is in the list, click **Play**. The result will sound very soft. Then choose *Formula...* from the **Modify** submenu and type

```
self * 3
```

Click OK, then click **Play** again. The sound is much louder now. You have multiplied the amplitude of every sample in the sound with a factor of 3.

Replacing the contents of a Sound with a formula

If you don't use `self` in your formula, the formula does not refer to the existing contents of the Sound. Hence, the formula

```
1/2 * sin (2 * pi * 377 * x)
```

will simply replace your recorded speech with a 377-Hz sine wave.

Modifying a Matrix with a formula

Many objects can be thought of as matrices: they consist of a number of rows and columns with data in every cell:

Sound: one row; columns represent samples.

Spectrum: two rows (first row is real part, second row is imaginary part); columns represent frequencies.

Spectrogram, Cochleagram: rows represent frequencies; columns represent times.

Excitation: one row; columns represent frequency bands.

Harmonicity: one row; columns represent time frames.

The formula is performed on every column of every row. The formula

```
self^2
```

will square all matrix elements.

The formula first works on the first row, and in that row from the first column on; this can work recursively. The formula

```
self + self [row, col - 1]
```

integrates each row.

Referring to the current position in the object

You can refer to the current position in a Matrix (or Sound, etc.) by index or by x and y values:

row

the current row

col

the current column

x

the x value associated with the current column:

for a Sound, Spectrogram, Cochleagram, or Harmonicity: time

for a Spectrum: frequency (Hertz)

for an Excitation: frequency (Bark)

y

the y value associated with the current row:

for a Spectrogram: frequency (Hertz)

for a Cochleagram: frequency (Bark)

Referring to the contents of the object itself

You can refer to values in the current Matrix (or Sound, etc.) by index.

self

refers to the value in the current Matrix at the current row and column, or to the value in the current Sound at the current sample.

self [column-expression]

refers to the value in the current Sound (or Intensity etc.) at the current sample (or frame). The *column-expression* is rounded to the nearest integer. If the index is out of range (less than 1 or greater than n_x), the expression evaluates as 0.

Example. An integrator is

```
self [col - 1] + self * dx
```

self [row-expression, column-expression]

refers to the value in the current Matrix (or Spectrogram etc.) at the specified row and column. The expressions are rounded to the nearest integers.

You can refer to values in the current Matrix (or Spectrogram, etc.) by x and y position:

o self (x -expression, y -expression)

the expressions are linearly interpolated between the four nearest matrix points.

You can refer to values in the current Sound (or Intensity etc.) by x position:

o self (x -expression)

the expression is linearly interpolated between the two nearest samples (or frames).

Links to this page

- [Formulas](#)
 - [Formulas 1. My first formulas](#)
 - [Formulas 1.9. Formulas in scripts](#)
-

© *ppgb*, December 4, 2002

Formulas 7. Attributes of objects

You can refer to several attributes of objects that are visible in the List of Objects. To do so, use the type and the name of the object, connected with an underscore. Thus, `Sound_hallo` refers to an existing Sound object whose name is "hallo" (if there is more than one such object, it refers to the one that was created last).

To refer to an attribute, you use the period ('.'). Thus, `Sound_hallo.nx` is the number of samples of `Sound_hallo`, and `1 / Sound_hallo.dx` is the sampling frequency of `Sound_hallo`.

Attributes in the calculator

Record a Sound (read the Intro if you do not know how to do that), name it "mysound" (or anything else), and type the following formula into the calculator:

```
Sound_mysound.nx
```

After you click OK, the Info window will show the number of samples. Since you could have got this result by simply choosing *Get number of samples* from the Query menu, these attribute tricks are not very useful in the calculator. We will see that they are much more useful in creation and modification formulas and in scripts.

List of possible attributes

The following attributes are available:

xmin

the start of the time domain (usually 0) for a Sound, Pitch, Formant, Spectrogram, Intensity, Cochleagram, PointProcess, or Harmonicity object, in seconds; the lowest frequency (always 0) for a Spectrum object, in Hertz; the lowest frequency (usually 0) for an Excitation object, in Bark; the left edge of the x domain for a Matrix object.

xmax

the end of the time domain (usually the duration, if $xmin$ is zero) for a Sound, Pitch, Formant, Spectrogram, Intensity, Cochleagram, PointProcess, or Harmonicity object, in seconds; the highest frequency (Nyquist frequency) for a Spectrum object, e.g. 11025 Hertz; the highest frequency for an Excitation object, often 25.6 Bark; the right edge of the x domain for a Matrix object.

ncol

the number of columns in a Matrix, TableOfReal, or Table object.

nrow

the number of rows in a Matrix, TableOfReal, or Table object.

nx

the number of samples in a Sound object; the number of analysis frames in a Pitch, Formant, Spectrogram, Intensity, Cochleagram, or Harmonicity object; the number of frequency bins in a Spectrum or Excitation object; the number of divisions of the x domain for a Matrix object (= $ncol$).

dx

the sample period (time distance between consecutive samples) in a Sound object (the inverse of the sampling frequency), in seconds; the time step between consecutive frames in a Pitch, Formant, Spectrogram, Intensity, Cochleagram, or Harmonicity object, in seconds; the width of a frequency bin in a Spectrum object, in Hertz; the width of a frequency bin in an Excitation object, in Bark; the horizontal distance between cells in a Matrix object.

ymin

the lowest frequency (usually 0) for a Spectrogram object, in Hertz; the lowest frequency (usually 0) for a Cochleagram object, in Bark; the bottom of the y domain for a Matrix object.

ymax

the highest frequency for a Spectrogram object, e.g. 5000 Hertz; the highest frequency for a Cochleagram object, often 25.6 Bark; the top of the y domain for a Matrix object.

ny

the number of frequency bands in a Spectrogram or Cochleagram object; for a Spectrum object: always 2 (first row is real part, second row is imaginary part) the number of divisions of the y domain for a Matrix object (= *nrow*).

dy

the distance between adjacent frequency bands in a Spectrogram object, in Hertz; the distance between adjacent frequency bands in a Cochleagram object, in Hertz; the vertical distance between cells in a Matrix object.

Attributes in a creation formula

In formulas for creating a new object, you can refer to the attributes of any object, but you will often want to refer to the attributes of the object that is just being created. You can do that in two ways.

The first way is to use the name of the object, as above. Choose Create Sound..., supply *hello* for its name, supply arbitrary values for the starting and finishing time, and type the following formula:

```
(x - Sound_hello.xmin) / (Sound_hello.xmax - Sound_hello.xmin)
```

When you edit this sound, you can see that it creates a straight line that rises from 0 to 1 within the time domain.

The formula above will also work if the Sound under creation is called *goodbye*, and a Sound called *hello* already exists; of course, in such a case `Sound_hello.xmax` refers to a property of the already existing sound.

If a formula refers to an object under creation, there is a shorter way: you do not have to supply the name of the object at all, so you can simply write

```
(x - xmin) / (xmax - xmin)
```

The attributes that you can use in this implicit way are *xmin*, *xmax*, *ncol*, *nrow*, *nx*, *dx*, *ny*, and *dy*.

Attributes in a modification formula

In formulas for modifying an existing object, you refer to attributes in the same way as in creation formulas, i.e., you do not have to specify the name of the object that is being modified. The formula

```
self * 20 ^ (- (x - xmin) / (xmax - xmin))
```

causes the sound to decay exponentially in such a way that it has only 5 percent of its initial amplitude at the end. If you apply this formula to multiple Sound objects at the same time, *xmax* will refer to the finishing time of each Sound separately as it is modified.

More examples of the use of attributes are on the next page.

Links to this page

- [Formulas](#)
-

© *ppgb*, December 4, 2002

Sound

One of the types of objects in PRAAT. For tutorial information, see all of the Intro.

Commands

Creation:

- Record mono Sound... (from microphone or line input, with the SoundRecorder)
- Record stereo Sound...
- Create Sound... (from a formula)
- Create Sound from tone complex...
- Create Sound from gamma-tone...
- Create Sound from Shepard tone...

Reading and writing:

- Sound files

You can also use the text and binary (real-valued) formats for Sounds, like for any other class:

- Write to text file...
- Write to binary file...

Playing:

- Sound: Play
- PointProcess: Hum
- PointProcess: Play

Viewing and editing:

- SoundEditor, ManipulationEditor, TextGridEditor, PointEditor, PitchTierEditor, SpectrumEditor

Queries:

structure:

- time domain
- Get number of samples
- Get sampling period
- Get sampling frequency
- Get time from sample number...
- Get sample number from time...

content:

- Sound: Get value at time...
- Sound: Get value at sample number...

shape:

- o Sound: Get minimum...
- o Sound: Get time of minimum...
- o Sound: Get maximum...
- o Sound: Get time of maximum...
- o Sound: Get absolute extremum...
- o Sound: Get nearest zero crossing...

statistics:

- o Sound: Get mean...
- o Sound: Get root-mean-square...
- o Sound: Get standard deviation...

energy:

- o Sound: Get energy...
- o Sound: Get power...

in air:

- o Sound: Get energy in air
- o Sound: Get power in air
- o Sound: Get intensity (dB)

Modification:

- Matrix: Formula...
- Sound: Set value at sample number...
- Sound: Filter with one formant (in-line)...
- Sound: Pre-emphasize (in-line)...
- Sound: De-emphasize (in-line)...

Annotation (see Intro 7. Annotation):

- Sound: To TextGrid...

Periodicity analysis:

- Sound: To Pitch...
- Sound: To Pitch (ac)...
- Sound: To Pitch (cc)...
- Sound: To Pitch (shs)...
- Sound: To Harmonicity (ac)...
- Sound: To Harmonicity (cc)...
- Sound: To PointProcess (periodic, cc)...
- Sound: To PointProcess (periodic, peaks)...
- Sound & Pitch: To PointProcess (cc)
- Sound & Pitch: To PointProcess (peaks)...
- Sound: To Intensity...

Spectral analysis:

- Sound: To Spectrum (fft)
- Sound: To Spectrogram...
- Sound: To Formant (burg)...
- Sound: To Formant (sl)...
- Sound: LPC analysis
- Sound: To LPC (autocorrelation)...
- Sound: To LPC (covariance)...
- Sound: To LPC (burg)...
- Sound: To LPC (marple)...

Filtering (see Filtering tutorial):

- Sound: Filter (pass Hann band)...
- Sound: Filter (stop Hann band)...
- Sound: Filter (formula)...
- Sound: Filter (one formant)...
- Sound: Filter (pre-emphasis)...
- Sound: Filter (de-emphasis)...
- LPC & Sound: Filter...
- LPC & Sound: Filter (inverse)
- Sound & Formant: Filter
- Sound & FormantTier: Filter

Conversion:

- Sound: Resample...

Enhancement:

- Sound: Lengthen (PSOLA)...: lengthen by a constant factor
- Sound: Deepen band modulation...: strengthen intensity modulations in each critical band

Combination:

- Sounds: Convolve
- Sounds: Concatenate

Synthesis

- Source-filter synthesis tutorial
- Manipulation (PSOLA etc.)
- Spectrum: To Sound (fft)
- **Pitch: To Sound...**
- PointProcess: To Sound (pulse train)...
- PointProcess: To Sound (hum)...

- **Pitch & PointProcess: To Sound...**
- Articulatory synthesis tutorial
- Artword & Speaker: To Sound...

Inside a Sound

With Inspect, you will see that a Sound contains the following data:

x_{min}

starting time, in seconds.

$x_{max} > x_{min}$

end time, in seconds.

n_x

the number of samples (≥ 1).

dx

sample period, in seconds. The inverse of the sampling frequency (in Hz).

x_1

the time associated with the first sample (in seconds). This will normally be in the range $[x_{min}, x_{max}]$. The time associated with the last sample (i.e., $x_1 + (n_x - 1) dx$) will also normally be in that range. Mostly, the sound starts at $t = 0$ seconds and $x_1 = dx / 2$. Also, usually, $x_{max} = n_x dx$.

$z [1] [1..n_x]$

the amplitude of the sound (stored as single-precision floating-point numbers). For the most common applications (playing and file I-O), Praat assumes that the amplitude is greater than -1 and less than +1. For some applications (modelling of the inner ear; articulatory synthesis), Praat assumes that the amplitude is expressed in Pascal units. If these interpretations are combined, we see that the maximum peak amplitude of a calibrated sound is 1 Pascal; for a sine wave, this means 91 dB SPL.

Limitations

Since the Sound object completely resides in memory, its size is limited to the amount of RAM in your computer. For sounds longer than a few minutes, you could use the LongSound object instead, which you can view in the LongSoundEditor.

Links to this page

- AIFF and AIFC files
- Band filtering in the frequency domain
- Formants & LPC submenu
- Formulas 1.7. Formulas for creation
- Formulas 1.8. Formulas for modification
- Formulas 7. Attributes of objects
- How to concatenate sound files
- Intro 1.1. Recording a sound
- Intro 2. What to do with a sound

- Intro 2.1. Writing a sound to disk
- Intro 2.2. Viewing and editing a sound
- Intro 3.1. Viewing a spectrogram
- Intro 3.7. The Spectrum object
- Intro 4.1. Viewing a pitch contour
- Intro 4.4. The Pitch object
- Intro 5.1. Viewing formant contours
- Intro 5.4. The Formant object
- Intro 6.1. Viewing an intensity contour
- Intro 8.1. Manipulation of pitch
- Intro 8.2. Manipulation of duration
- Intro 8.3. Manipulation of intensity
- Macintosh sound files
- Manipulation: Extract original sound
- Manipulation: Replace original sound
- NIST files
- Periodicity submenu
- Play
- sampling frequency
- sampling period
- Sesam/LVS files
- Sound & Formant: Filter (no scale)
- Sound & FormantTier: Filter (no scale)
- Sound & IntensityTier: Multiply
- Sound & Pitch: Change gender...
- Sound & Pitch: To FormantFilter...
- Sound files 3. Files that Praat can read
- Sound: Change gender...
- Sound: Formula...
- Sound: To BarkFilter...
- Sound: To Formant (keep all)...
- Sound: To FormantFilter...
- Sound: To MelFilter...
- Sound: To MFCC...
- Sound: To Spectrum (dft)
- SpellingChecker
- TextGrid
- TextTier
- What's new?
- Write to AIFC file...
- Write to AIFF file...
- Write to NeXT/Sun file...
- Write to NIST file...

- Write to WAV file...
-

© *ppgb*, April 20, 2004

SoundEditor

An Editor for viewing and editing a Sound object. Most of the functions of this editor are described in the Intro.

The markers

To set the cursor, use the left mouse button. A horizontal line will also be shown; this line crosses the cursor line at the sound's function value. This function value is the sinc-interpolated value, and is generally different from the value that you would expect when looking at the linearly interpolated version of the sampled sound.

To select a part of the time domain, use the time selection mechanism.

Playing

To play any part of the sound, click on one of the rectangles below or above the sound window (there can be 1 to 8 of these rectangles), or choose a Play command from the View menu.

Publishing

To perform analyses on the selection, or save it to a file, create an independent Sound as a copy of the selection, by clicking on the button that will copy the selection to the List of Objects; the resulting Sound will be called "Sound untitled".

Editing

- Cut: cut the selection to the clipboard.
- Copy: copy the selection to the clipboard.
- Paste: paste the clipboard to the cursor.
- Zero: set the selected samples to zero.
- Reverse: reverse the selected part of the sound.

You can undo these commands with Undo (Command-Z).

The Group button

To synchronize a SoundEditor window with other windows that show a time signal, push the Group button in all the windows that you want to synchronize. You cannot Cut from or Paste into a synchronized SoundEditor window.

Links to this page

- [Advanced pitch settings...](#)
- [Extract visible formant contour](#)
- [Extract visible intensity contour](#)
- [Extract visible pitch contour](#)
- [Extract visible spectrogram](#)
- [Formant analysis...](#)
- [Get first formant](#)
- [Get pitch](#)
- [Get second formant](#)
- [Intro 3.1. Viewing a spectrogram](#)
- [Intro 3.5. Viewing a spectral slice](#)
- [Intro 4.1. Viewing a pitch contour](#)
- [Intro 4.3. Querying the pitch contour](#)
- [Intro 4.4. The Pitch object](#)
- [Intro 5.1. Viewing formant contours](#)
- [Intro 5.3. Querying the formant contours](#)
- [Intro 5.4. The Formant object](#)
- [Intro 6.1. Viewing an intensity contour](#)
- [Log files](#)
- [LongSoundEditor](#)
- [Pitch settings...](#)
- [Play](#)
- [Show formant](#)
- [Show intensity](#)
- [Show pitch](#)
- [Show pulses](#)
- [Show spectrogram](#)
- [Spectrogram settings...](#)
- [TextGridEditor](#)
- [Time step settings...](#)
- [Types of objects](#)
- [Voice](#)
- [What's new?](#)

© ppgb, March 12, 2003

Statistics

This is the tutorial about basic statistical techniques in Praat, which work with the Table object. It assumes that you are familiar with the Intro.

(Under construction.....)

For more sophisticated techniques, see:

- Principal component analysis
- Multidimensional scaling
- Discriminant analysis

Links to this page

- [What's new?](#)
-

© *ppgb*, June 24, 2002

time domain

This manual page assumes that you have read the Intro.

Many objects in Praat are *functions of time*. Examples are: Sound, Pitch, Spectrogram, Formant, Intensity, TextGrid, PitchTier, DurationTier, Harmonicity, PointProcess.

In Praat, these functions have a contiguous **time domain**, i.e. a single time stretch with a starting time and a finishing time. The total duration of such a function is the difference between the starting time and the finishing time. There are up to five ways to see the time domain of an object.

The time domain in editor windows

If you select an object that is a function of time and click **Edit**, an editor window will appear on the screen. The rectangle at the bottom will show the starting time, the finishing time, and the total duration.

The time domain in the picture window

If you select an object that is a function of time and choose one of the **Draw** commands, the settings window invites you to supply a time range. If you keep this time range at its standard setting (from 0.0 to 0.0 seconds), Praat will draw the object for the whole time domain and print the starting time and the finishing time below the horizontal axis (if **Garnish** is on).

The time domain in the Info window

If you select an object that is a function of time and click **Info**, the Info window will tell you the starting time, the finishing time, and the total duration (among other properties of the object).

Time domain query commands

If you select an object that is a function of time, the following three commands will become available in the **Query** menu:

- Get starting time**
- Get finishing time**
- Get total duration**

If you choose one of these commands, the Info window will tell you the result, expressed in seconds. These commands are most useful in a Praat script. Example:

```
select Pitch hello
startingTime = Get starting time
finishingTime = Get finishing time
centreTime = (startingTime + finishingTime) / 2
echo This Pitch runs from 'startingTime' to 'finishingTime' seconds,
```

`println` and the centre of its time domain is at `'centreTime'` seconds.

Details for hackers

If you select an object that is a function of time and you click Inspect, you can see how the time domain information is stored in the object: the starting time is the object's **xmin** attribute and the finishing time is its **xmax** attribute. The total duration is not stored in the object, since it can easily be computed as **xmax** minus **xmin**.

© *ppgb*, May 6, 2004

Types of objects

PRAAT contains the following types of objects and Editors. For an introduction and tutorials, see Intro.

General purpose:

- Matrix: a sampled real-valued function of two variables
- Polygon
- PointProcess: a point process (PointEditor)
- Sound: a sampled continuous process (SoundEditor, SoundRecorder, Sound files)
- LongSound: a file-based version of a sound (LongSoundEditor)
- Strings
- Distributions, PairDistribution
- Table, TableOfReal
- Sequence
- ParamCurve

Periodicity analysis:

- Tutorials:
- Intro 4. Pitch analysis
- Intro 6. Intensity analysis
- Voice (jitter, shimmer, noise)
- Pitch: articulatory fundamental frequency, acoustic periodicity, or perceptual pitch (PitchEditor)
- Harmonicity: degree of periodicity
- Intensity, IntensityTier: intensity contour

Spectral analysis:

- Tutorials:
- Intro 3. Spectral analysis
- Intro 5. Formant analysis
- Spectrum: complex-valued equally spaced frequency spectrum (SpectrumEditor)
- Ltas: long-term average spectrum
- Spectro-temporal: Spectrogram, BarkFilter, MelFilter, FormantFilter
- Formant: acoustic formant contours
- LPC: coefficients of Linear Predictive Coding, as a function of time
- Cepstrum, CC, LFCC, MFCC (cepstral coefficients)
- Excitation: excitation pattern of basilar membrane
- Excitations: an ensemble of **Excitation** objects
- Cochleagram: excitation pattern as a function of time

Labelling and segmentation (see Intro 7. Annotation):

- TextGrid (TextGridEditor), IntervalTier, TextTier

Listening experiments:

- ExperimentMFC

Manipulation of sound:

- Tutorials:
- Intro 8.1. Manipulation of pitch
- Intro 8.2. Manipulation of duration
- Intro 8.3. Manipulation of intensity
- Filtering
- Source-filter synthesis
- PitchTier (PitchTierEditor)
- Manipulation (ManipulationEditor): PSOLA
- DurationTier
- FormantTier

Articulatory synthesis (see the Articulatory synthesis tutorial):

- Speaker: speaker characteristics of a woman, a man, or a child
- **Articulation**: snapshot of articulatory specifications (muscle activities)
- Artword: articulatory target specifications as functions of time
- (VocalTract: area function)

Neural net package:

- FFNet: feed-forward neural net
- Pattern
- Categories: for classification (**CategoriesEditor**)

Numerical and statistical analysis:

- Eigen: eigenvectors and eigenvalues
- Polynomial, Roots, ChebyshevSeries, LegendreSeries, ISpline, MSpline
- Covariance: covariance matrix
- Confusion: confusion matrix
- Discriminant analysis: Discriminant
- Principal component analysis: PCA
- Correlation, ClassificationTable, SSCP
- DTW: dynamic time warping

Multidimensional scaling:

- Configuration (Saliency)
- Kruskal analysis: Dissimilarity (Weight), Similarity
- INDSCAL analysis: Distance, ScalarProduct
- Correspondence analysis: ContingencyTable

Optimality-theoretic learning (see the OT learning tutorial)

- OTGrammar (OTGrammarEditor)
- OTAnyGrammar (OTAnyGrammarEditor): OTGrammar_tongueRoot

Bureaucracy

- WordList, SpellingChecker

Links to this page

- AffineTransform
- CCA
- Procrustus
- Proximity

© *ppgb*, March 16, 2003

What's new?

Latest, coming, and requested changes in PRAAT.

4.2.17 (September 14, 2004)

- Info window: can save.
- Script editor: line numbers.
- Unix: picture highlighting as on Mac and Windows.

4.2.16 (September 5, 2004)

- One can now drag the inner viewport in the Picture window, excluding the margins. This is nice e.g. for creating square viewports or for drawing a waveform and a spectrogram in close contact.

4.2.14 (August 20, 2004)

- Windows: "Write to fontless EPS file..." now correctly references SILDoulosIPA rather than SILDoulosIPA-Regular.

4.2.13 (August 12, 2004)

- ExperimentMFC: removed a bug that caused Praat to crash if the carrier-before phrase was longer than the carrier-after phrase.

4.2.11 (August 11, 2004)

- Optimality Theory: added WeightByPosition and *MoraicConsonant, to learn coda weights.

4.2.10 (August 10, 2004)

- Optimality Theory: Remove harmonically bounded candidates.
- Linux: removed the same bug as in 4.1.13, but now for scripts.

4.2.09 (July 28, 2004)

- Command line scripting: better handling of spaces.
- Optimality Theory: added *Clash and *Lapse.

4.2.08 (July 9, 2004)

- ExperimentMFC: subjects can now respond with keyboard presses.
- Shimmer measurements: more accurate and less sensitive to additive noise.

4.2.07 (June 16, 2004)

- Sound editor: queries/reports/listings sometimes report on the whole visible part (and then say so).
- SSCP: Draw sigma ellipse: allow reversed axes.
- Scripting: corrected readability of voice report.

4.2.06 (May 14, 2004)

- Reading and opening 24-bit and 32-bit sound files (saving still goes in 16 bits).
- Sound: Scale intensity.

4.2.05 (May 13, 2004)

- More extensive voice report (pitch statistics; harmonicity).
- Pitch-corrected LTAS analysis.
- Much faster Fourier transform (non-FFT version).
- More drawing methods for Sound and Ltas (curve, bars, poles, speckles).
- More TableOfReal extraction commands.
- Removed bugs from the orthographic representation of vowels in the Peterson & Barney table.
- Removed a bug introduced in 4.2.01 that caused a lack of redrawing in the button editor window.

4.2.04 (April 21, 2004)

- Removed a bug introduced in 4.2.01 that caused a lack of redrawing in the PointProcess and LongSound editor windows.
- Sound: To Spectrum (dft)

4.2.03 (April 20, 2004)

- MacOS X and Linux: removed a bug introduced in 4.2.01 by which **Create Strings from file list...** did not work with a wildcard ("*").

4.2.01 (April 7, 2004)

- Improved frequency precision in spectrogram drawing.
- Less flashing in sound window.
- Better warnings against use of the LPC object.

4.2 (March 4, 2004)

4.1.28 (February 17, 2004)

- Better selections in Picture window and editor windows.

4.1.27 (February 11, 2004)

- ManPages: script links can receive arguments.
- Picture window: better handling of rectangles and ellipses for reversed axes.

4.1.26 (February 4, 2004)

- Windows: corrected positioning of pictures on clipboard and in metafiles.
- Windows: EPS files check availability of Times and TimesNewRomanPSMT.

4.1.25 (January 29, 2004)

- Optimality Theory: metrics grammar supports 'impoverished overt forms', i.e. without secondary stress even if surface structures do have secondary stress.
- Polygon: can now also paint in colour instead of only in grey values.
- Unlimited number of points for polygons in PostScript (may not work on very old printers).
- MacOS X: removed a bug by which the script editor did not always know that the text had changed.
- Windows: in EPS files use Times New Roman instead of Times.

4.1.24 (January 13, 2004)

- MacOS X: removed a major bug introduced in 4.1.22 that caused text fields to be unclickable.

4.1.23 (January 11, 2004)

- MacOS X: removed a serious bug introduced in 4.1.22 that caused loss of text when saving and running scripts.
- Macintosh: removed a bug that could cause wrong line widths in copied pictures.
- Picture window: line widths on all printers and clipboards are now equal to line widths used on PostScript printers: a line with a line width of "1" will be drawn with a width 3/8 points. This improves the looks of pictures printed on non-PostScript printers, improves the looks of pictures copied to your wordprocessor when printed, and changes the looks of pictures copied to your presentation program.

4.1.22 (January 7, 2004)

- Script editor: support for unlimited size of script files in editor window on Windows XP and MacOS X (the Unix editions already had this).
- Script editor: better scrolling of text on Windows XP and MacOS X (the Unix editions already had this).
- Inspect: removed several bugs that caused ugly scrolling of text fields.

4.1.21 (December 15, 2003)

- Removed a major bug introduced in 4.1.16 that caused a Sound to change when computing an Intensity from it.

4.1.20 (December 9, 2003)

- Removed a major bug introduced in 4.1.19 that could give weird analysis settings in the sound editor window, or "Wrong field in dialog; notify author." crashes.

4.1.19 (December 3, 2003)

- Editor windows give better feedback of undersampling.
- Windows: repaired an age-old bug by which the Picture window could not be closed twice.
- SoundEditor gives more complete settings report.

4.1.18 (November 26, 2003)

- Repaired a small memory leak in Sound: To Manipulation.
- Manual pages: variable duration of recording.

4.1.17 (November 20, 2003)

- PitchTier: improved Interpolate quadratically.

4.1.16 (November 19, 2003)

- PitchTier: Interpolate quadratically.

4.1.15 (November 7, 2003)

- More phonetic diacritics.

4.1.14 (October 29, 2003)

- TextGrids can be saved chronologically (and Praat can read that file again).

4.1.13 (October 22, 2003)

- Removed a bug that caused Matrix: Solve equation... to generate an incorrect result and sometimes crash PRAAT whenever the number of rows was larger than the number of columns.
- Linux: removed a bug that caused PRAAT to play sounds at double speed for some sound cards.
- OT learning: full support for crucially tied constraints and tied candidates; queries for testing grammaticality.

4.1.12 (October 17, 2003)

- OT learning: some support for crucial ties and backtracking in EDCCD.

4.1.11 (October 9, 2003)

- Removed a bug that caused Praat to crash whenever "Extract intervals" found an interval with a label longer than 200 characters.

4.1.10 (October 3, 2003)

- Sound editor window Time step settings...: "Automatic", "Fixed", and "View-dependent".
- Improved the reception of *sendpraat* commands on Windows XP.

4.1.9 (September 27, 2003)

- Removed a major bug introduced by the new FFT algorithms in version 4.1.3, which caused a doubling of the amplitude when resampling or filtering a sound, and an octave jump when smoothing a pitch curve.

4.1.8 (September 17, 2003)

- Sound window: distinguish basic from advanced spectrogram and pitch settings.
- Read TableOfReal from headerless spreadsheet file...: cells with strings are considered zero.
- Sound window: introduced time step as advanced setting.
- Sound window: reintroduced view range as advanced setting.

4.1.7 (September 12, 2003)

- Sound Designer II support (reading and opening).
- Sound window scripting: corrected extraction of spectrogram/pitch/formant/pulses.

4.1.6 (August 22, 2003)

- MacOS X: prevented SoundRecorder from crashing Praat in case of loud signals on some computers.
- Sun workstations: support audio servers.

4.1.4 (July 10, 2003)

- Open source code (General Public Licence).
- More precision in numeric libraries.

4.1.3 (July 2, 2003)

- Faster computation of spectrum, spectrogram, and pitch.

4.1.2 (June 19, 2003)

- Ltas: Compute trend line, Subtract trend line.

4.1.1 (June 5, 2003)

- Formulas for PitchTier, IntensityTier, AmplitudeTier, DurationTier.
- Jitter: corrected the computation of PPQ5 (divide by $N-4$ instead of $N-2$).

4.1 (May 28, 2003)

4.0.54 (May 27, 2003)

- Spectrogram painting: allow fixed maximum or autoscaling.
- OT: learning in editor.

4.0.53 (May 21, 2003)

- Documented jitter and shimmer measurements.
- Sound editor: constant time step for pitch analysis.

4.0.52 (May 9, 2003)

- TextGrid & Sound: Extract non-empty intervals.
- Optimality-Theoretic learning: more constraints in metrics grammar.

4.0.51 (April 16, 2003)

- Much more accurate shimmer measurements.
- Ltas: merge.

4.0.50 (April 3, 2003)

- More Intro.
- Editors: Get spectral power at cursor cross.

4.0.49 (March 12, 2003)

- Windows 2000 and XP: put preferences files in home directory.
- Removed crashing bug from TextGrid: Extract part.

4.0.48 (March 9, 2003)

- ExperimentMFC: multiple substimuli for discrimination tests.
- Formulas: can use variables without quotes.
- Printing: hard-coded image interpolation for EPS files and PostScript printers.
- Read more NIST files.
- Sound: To PointProcess (periodic, peaks)...

4.0.47 (February 26, 2003)

- Editors: Intro 3.5. Viewing a spectral slice.
- Windows: corrected multiple selection in lists.
- Sound editor: clearer settings dialogs.

4.0.46 (February 13, 2003)

- Scripting: disallowed ambiguous expressions like -3^2 .

4.0.45 (February 7, 2003)

- PSOLA synthesis: reduced buzz in voiceless parts.
- ExperimentMFC: stimulus file path can be relative to directory of experiment file.

4.0.44 (February 6, 2003)

- Direct PostScript printing now possible again on Windows 95 and 98.

4.0.43 (February 5, 2003)

- Removed a bug by which WAV files could not be read.

4.0.42 (February 5, 2003)

- Sound: Change gender...

4.0.41 (January 15, 2003)

- Sound editor window: added queries for formants higher than the fifth.

4.0.40 (January 3, 2003)

- Scripting: removed bug with multiple includes.

4.0.38 (December 18, 2002)

- Improved manuals.

4.0.37 (December 15, 2002)

- Scripting: stopped support of things that had been undocumented for the last four years: **let**, **getnumber**, **getstring**, **ARGS**, **copy**, **proc**, variables with capitals, and strings in numeric variables; there are messages about how to modify your old scripts.
- Improved manual and tutorials.

4.0.36 (December 11, 2002)

- Removed a serious scripting bug introduced in 4.0.35.
- Scripting 5.7. Including other scripts.

4.0.35 (December 4, 2002)

- Rewritten Intro.
- Rewritten Scripting tutorial.
- New Formulas tutorial.
- String formulas in the calculator.
- Scripting: `extractNumber`, `extractWord$`, `extractLine$`. See Formulas 5. String functions.

4.0.34 (November 19, 2002)

- Many jitter and shimmer measures in the Sound editor window.

4.0.33 (November 18, 2002)

- Direct PostScript printing now possible also on Windows 2000 and XP.
- Removed bug that incorrectly genericized some non-ASCII characters.

4.0.31 (November 12, 2002)

- OT learning: tutorial for bidirectional learning.
- OT learning: random choice between equally violating candidates.

4.0.30 (October 16, 2002)

- Made pitch analysis in Sound editor insensitive to infinite window lengths.
- PCA bug removed.

4.0.29 (October 1, 2002)

- Searches in the manual are case-insensitive.
- Scripting on Windows: removed bugs from the **system** command (caused by the CodeWarrior C library).

4.0.28 (September 6, 2002)

- More than 99 percent of the source code distributed under the General Public Licence.

4.0.27 (August 30, 2002)

- Multiple ResultsMFC: **To Table**, so that the whole experiment can go into a single statistics file.

4.0.24 (July 8, 2002)

- ExperimentMFC: goodness judgments.

4.0.23 (June 24, 2002)

- Scripting and formulas: refer to cells by row or column name, e.g. Table_tokens [i, "F1"].
- Scripting: assignment by modification, as with += -= *= /=.
- Scripting: date\$().

4.0.22 (June 19, 2002)

- Corrected serious new bug in Formula (x attribute unknown).

4.0.21 (June 12, 2002)

- Table: column names as variables.

4.0.20 (June 11, 2002)

- Table: scatter plot.

4.0.19 (June 4, 2002)

- New Table object for column statistics: Pearson's r , Kendall's τ - b , t -test.
- Formulas: refer to any matrices and tables.
- Scripting: possibility for using things like Sound_hello (x) or Table_everything [row, col].

4.0.18 (May 27, 2002)

- TextGrid: shift times, scale times.
- Pitch: removed crashing bug from dialogs for drawing.

4.0.17 (May 22, 2002)

- PitchTier: shift or multiply frequencies (also in ManipulationEditor).
- Spectrum: the sign of the Fourier transform has changed, to comply with common use in technology and physics. Old Spectrum files are converted when read.

4.0.16 (May 12, 2002)

- Improved manual.

4.0.14 (April 22, 2002)

- Corrected font bug in Classic Mac version.

4.0.13 (April 17, 2002)

- Many small corrections.

4.0.12 (April 3, 2002)

- Sound editor: formant report.
- Dialogs: layout improvements.

4.0.11 (March 25, 2002)

- Mono recording.
- Dialogs improved: range fields and option menus.
- Some jitter and shimmer measurements.

4.0.9 (February 22, 2002)

- Direct PostScript printing on MacOS X.

4.0.8 (February 21, 2002)

- T-test.

4.0.6 (February 7, 2002)

- Native MacOS X version.

4.0.5 (January 25, 2002)

- Removed licensing.
- Remember PostScript settings.

4.0.4 (November 30, 2001)

- Spectral moments.

4.0.3 (November 27, 2001)

- TableOfReal: Extract rows where column...
- Correlation: Confidence intervals...
- SSCP: Get diagonality (bartlett)...
- LPC: To Matrix
- CC: To Matrix

4.0.2 (November 12, 2001)

- Correlation: Confidence intervals...
- Removed bug that caused mean formants in log files to be zero.

4.0.1 (October 25, 2001)

- TableOfReal: Get correlation....
- TextGrid: bug removals in Modify menu (tier insertion and duplication).

4.0 (October 15, 2001)

- Sound: To Pitch (ac)...: pitch contour less dependent on time resolution. This improves the constancy of the contours in the editors when zooming.

3.9.37 (October 4, 2001)

- Editors: better visible pitch contour.
- Editors: Get minimum & maximum pitch; move cursor to min & max pitch.
- Special symbols: háč [IMAGE] ek.
- Removed a serious bug in TextGrid (Modify menu): Insert tier...

3.9.36 (September 24, 2001)

- TextGrid: tier insertion and deletion in Modify menu.
- ResultsMFC queries.

3.9.34 (August 16, 2001)

- Intensity: To IntensityTier (peaks, valleys).

3.9.33 (August 4, 2001)

- Removed a very old bug with multiple list replacement in the CategoriesEditor.
- Removed a very old bug that caused Praat to crash after interrupting a "Deepen band modulation" in progress.
- Removed a bug that prevented scripts to query analyses in sound windows immediately after creating the editor.

3.9.32 (August 1, 2001)

- Sounds: Concatenate recoverably. Creates a TextGrid whose interval labels are the original names of the sounds.
- Sound & TextGrid: Extract all intervals. The reverse of the previous command.

3.9.31 (July 18, 2001)

- Regular expressions.

3.9.29 (July 7, 2001)

- Improved random numbers and other numerical stuff.
- Corrected MFCC button bug (since 3.9.25).

3.9.28 (June 8, 2001)

- ExperimentMFC: multiple-forced-choice listening experiments.
- TextGrid and TextGridEditor: additions and improvements.

3.9.27 (May 19, 2001)

- Log files.
- Formatting in variable substitution, e.g. 'pitch:2' gives two digits after the decimal point.

3.9.26 (May 16, 2001)

- More TextGrid queries.

3.9.25 (May 15, 2001)

- New: Multidimensional scaling tutorial.
- New: filterbank analyses, MelFilter, BarkFilter and FormantFilter, by band filtering in the frequency domain.
- A change in the data structure for cepstral coefficients results in a change in the MFCC. As a side-effect the **Cepstrum** class is now renamed as LFCC. In a following version the Cepstrum object will return as the representation of the *complex cepstrum*.

3.9.24 (May 14, 2001)

- New: cepstrum and more statistics by David Weenink.
- Matrix: Get sum.
- PairDistribution: Get percentage correct (maximum likelihood, probability matching).
- OTGrammar & PairDistribution: Get percentage correct...

3.9.23 (May 6, 2001)

- Windows: improved text focus in TextGrid editor.
- Scripting: added hints about extra spaces in file names to error messages.
- Praatcon: protected against calling without a script name.

3.9.22 (April 23, 2001)

- Removed a bug that caused Praat to crash when opening a TextGrid window without a Sound.

3.9.21 (April 11, 2001)

- Added intensity contour to SoundEditor, LongSoundEditor, and TextGridEditor windows.
- Improved spectrogram drawing.

3.9.20 (April 4, 2001)

- Windows: corrected running audio cursor.

3.9.19 (April 2, 2001)

- Simplified selection and cursor in editor windows.
- Replaced Analysis and AnalysisEditor with Manipulation and ManipulationEditor.

3.9.18 (March 29, 2001)

- Removed a bug that caused editor windows to crash after the Preferences command.

3.9.17 (March 29, 2001)

- Removed a bug that caused editor windows to crash for very short sounds.

3.9.16 (March 28, 2001)

- Spectrogram, pitch contour, and formant contour available in the SoundEditor, LongSoundEditor, and TextGridEditor windows, including query commands.
- Scripting: corrected failing **choice** arguments in **execute** directive.

3.9.15 (March 7, 2001)

- SoundRecorder on Windows: worked around a Windows bug that caused the Stop button not to work.

3.9.13 (February 14, 2001)

- Macintosh: support for multiple audio input devices (sound cards).
- Repaired a memory leak in Manipulation-to-Sound.

3.9.12 (January 26, 2001)

- TextGridEditor: view spectrogram.

3.9.11 (January 10, 2001)

- TextGridEditor: improved scrolling strategy when navigating with Option-arrow.
- TextGridEditor: changed standard setting of shift-drag strategy.

3.9.9 (December 11, 2000)

- Macintosh: improved screen rendition of rotated text.

3.9.8 (December 7, 2000)

- Windows: corrected a small bug in scripts with relative paths to root directories.

3.9.7 (December 6, 2000)

- Enabled debugging-at-a-distance.

3.9.6 (November 30, 2000)

- Windows: removed a bug that limited "Spectrum: Draw (log freq)...".

3.9.5 (November 16, 2000)

- Implemented autoscaling in LongSound editor and in TextGrid editor.
- Added **fixed\$** to scripting language for formatting of numbers.
- Macintosh: removed bug that prevented scripting of picture file saving.

3.9.4 (October 30, 2000)

- Defended against drawing zero-width ellipses on PostScript.
- Worked around NT bug that prevents filling small circles (formant contour in editor invisible).
- TextGridEditor: Option-Shift-arrow for extending interval selection.

3.9.3 (October 26, 2000)

- Windows: corrected slow running cursor while sound was playing on some NT computers.

3.9.2 (October 25, 2000)

- Windows: corrected bug in some non-PostScript HP printers that caused wrong positioning of spectrograms.
- Enabled reading of Praat picture file if window has contents.

Praat 3.9, October 18, 2000

Editors:

- Shift-click and shift-drag extend or shrink selection in editor windows.
- Grouped editors can have separate zooming and scrolling (FunctionEditor preferences).
- Cursor follows playing sound in editors; interruption by Escape key moves the cursor.
- TextGridEditor: optimized for transcribing large corpora: text field, directly movable boundaries, more visible text in tiers, SpellingChecker, type while the sound is playing, complete keyboard navigation, control font size, control text alignment, shift-click near boundary adds interval to selection.
- Stereo display in LongSound and TextGrid editors.
- LongSoundEditor and TextGridEditor: write selection to audio file.
- SoundEditor: added command "Extract selection (preserve times)".
- IntervalTierEditor, DurationTierEditor.
- Added many query commands in editors.

Phonetics library:

- Sound: To Formant...: sample-rate-independent formant analysis.
- Sound: To Harmonicity (glottal-to-noise excitation ratio).
- Pitch: support for ERB units, draw all combinations of line/speckle and linear/logarithmic/semitones/mels/erbs, optionally with TextGrid, Subtract linear fit.
- Spectrum: Draw along logarithmic frequency axis.
- TextGrid: modification commands, Extract part, Shift to zero, Scale times (with Sound or LongSound).
- Matrix: To TableOfReal, Draw contour...
- Concatenate Sound and LongSound objects.
- File formats: save PitchTier in spreadsheet format, read CGN syntax files (XML version), text files now completely file-server-safe (independent from Windows/Macintosh/Unix line separators).

Statistics and numerics library:

- Principal component analysis.
- Discriminant analysis.
- Polynomial: drawing, root finding etc.
- TableOfReal: Draw box plots....
- Covariance: To TableOfReal (random sampling)....
- SSCP: Get sigma ellipse area....
- Query DTW for 'weighted distance' of time warp.
- Distributions: To Strings (exact)...
- Strings: Randomize.

Phonology library:

- OTGrammar: To PairDistribution.

Graphics:

- Full support for colour inkjet printers on Windows and Macintosh.
- Full support for high-resolution colour clipboards and metafiles for Windows and Macintosh programs that support them (this include MS Word for Windows, but unfortunately not MS Word for Macintosh).
- Colour in EPS files.
- Interpolating grey images, i.e. better zoomed spectrograms.
- Linux: support for 24-bits screens.

Audio:

- Asynchronous sound play.
- Linux: solved problems with /dev/mixer ("Cannot read MIC gain.") on many computers.
- Added possibility of zero padding for sound playing, in order to reduce clicks on some Linux and Sun computers.
- LongSound supports mono and stereo, 8-bit and 16-bit, μ -law and A-law, big-endian and little-endian, AIFC, WAV, NeXT/Sun, and NIST files.
- "Read two Sounds from stereo file..." supports 8-bit and 16-bit, μ -law and A-law, big-endian and little-endian, AIFC, WAV, NeXT/Sun, and NIST files.
- SoundRecorder writes to 16-bit AIFC, WAV, NeXT/Sun, and NIST mono and stereo files.
- Sound & LongSound: write part or whole to mono or stereo audio file.
- Read Sound from raw Alaw file.
- Artword & Speaker (& Sound) movie: real time on all systems.

Scripting:

- Formulas 4. Mathematical functions: added statistical functions: χ^2 , Student T, Fisher F, binomial, and their inverse functions.
- Windows: program **praatcon** for use as a Unix-style console application.

- Windows and Unix: Praat can be run with a command-line interface without quitting on errors.
- Unix & Windows: can use <stdout> as a file name (supports pipes for binary data).
- sendpraat now also for Macintosh.
- sendsocket.
- Read from file... recognizes script files if they begin with "#!".
- Script links in ManPages.

Documentation

- Tutorials on all subjects available through Intro.

Praat 3.8, January 12, 1999

Phonetics library

- New objects: LongSound (view and label long sound files), with editor; PairDistribution.
- PSOLA manipulation of voiceless intervals, version 2: quality much better now; target duration is exactly as expected from Duration tier or specified lengthening in Sound: Lengthen (PSOLA)...
- Audio: Escape key stops audio playing (on Mac also Command-period).
- SoundRecorder: allows multiple recordings without close; Play button; Write buttons; buffer size can be set.
- Reverse a Sound or a selection of a Sound.
- Sound: Get nearest zero crossing....
- Formant: "Scatter plot (reversed axes)..."
- TextGrid & Pitch: "Speckle separately..."
- "Extract Sound selection (preserve times)" in TextGridEditor.
- More query commands for Matrix, TableOfReal, Spectrum, PointProcess.

Phonology library

- 25-page OT learning tutorial.
- Made the OT learner 14 times as fast.

Systems

- May 23: Windows beta version.
- April 24: Windows alpha version.

Files

- Read more Kay, Sun (.au), and WAV sound files.
- "Read Strings from raw text file..."
- Create Strings as file list....
- "Read IntervalTier from Xwaves..."
- hidden "Read from old Windows Praat picture file..."

Graphics

- Use colours (instead of only greys) in "Paint ellipse..." etc.
- More true colours (maroon, lime, navy, teal, purple, olive).
- Direct printing from Macintosh to PostScript printers.
- Hyperpage printing to PostScript printers and PostScript files.
- Phonetic symbols: raising sign, lowering sign, script g, corner, ligature, pointing finger.

Shell

- November 4: all dialogs are modeless (which is new for Unix and Mac).
- September 27: sendpraat for Windows.

Scripting

- January 7: scriptable editors.
- October 7: file I/O in scripts.
- August 23: script language includes all the important functions for string handling.
- June 24: string variables in scripts.
- June 22: faster look-up of script variables.
- June 22: unlimited number of script variables.
- April 5: suspended chopping of trailing spaces.
- March 29: enabled formulas as arguments to dialogs (also interactive).

Praat 3.7, March 24, 1998

Editors:

- In all FunctionEditors: drag to get a selection.

Phonetics library:

- Many new query (**Get**) commands for Sound, Intensity, Harmonicity, Pitch, Formant, Ltas, PitchTier, IntensityTier, DurationTier, FormantTier.
- Many new modification commands.
- Many new interpolations.
- Sound enhancements: Sound: Lengthen (PSOLA)...., Sound: Deepen band modulation...
- Source-filter synthesis tutorial, Sound & IntensityTier: Multiply, Sound & FormantTier: Filter, Formant: Formula (frequencies)...., Sound: Pre-emphasize (in-line)....

Labelling

- TextGrid queries (**Get** times and labels in a script).
- TextGrid: Count labels....
- PointProcess: To TextGrid (vuv)....: get voiced/unvoiced information from a point process.
- IntervalTier to TableOfReal: labels become row labels.
- TextTier to TableOfReal.

Numerics and statistics library

- Multidimensional scaling (Kruskal, INDSCAL, etc).
- TableOfReal: Set value, Formula, Remove column, Insert column, Draw as squares, To Matrix.

Phonology library

- OT learning: new strategies: weighted symmetric plasticity (uncancelled or all).

Praat shell

- First Linux version.
- Eight new functions like e.g. *hertzToBark* in Formulas 4. Mathematical functions.
- Praat script: procedure arguments; object names.

Documentation:

- 230 more man pages (now 630).
- Hypertext: increased readability of formulas, navigation with keyboard.

Praat 3.6, October 27, 1997

Editors:

- Intuitive position of B and E buttons on left-handed mice.
- SoundEditor: copy *windowed* selection to list of objects.
- SoundEditor: undo Cut, Paste, Zero.
- SpectrumEditor: copy band-filtered spectrum or sound to list of objects.
- ManipulationEditor: LPC-based pitch manipulation.

Objects:

- Use '-', and '+' in object names.

Phonetics library

- LPC-based resynthesis in ManipulationEditor.
- Sound: direct modification without formulas (addition, multiplication, windowing)
- Sound: filtering in spectral domain by formula.
- Create a simple Pitch object from a PitchTier (for F_0) and a Pitch (for V/U).
- Semitones in PitchTier tables.
- PointProcess: transplant time domain from Sound.
- Much more...

Phonology library

- Computational Optimality Theory. See OT learning.

Hypertext

- You can use ManPages files for creating your own tutorials. These contains buttons for playing and recording sounds, so you can use this for creating an interactive IPA sound training course.

Scripting:

- Programmable Praat script language: variables, expressions, control structures, procedures, complete dialog box, exchange of information with Info window, continuation lines.
- Use system-independent relative file paths in Praat script.
- ScriptEditor: Run selection.

Graphics:

- Rotation and scaling while printing the Picture window.
- Apart from bold and italic, now also bold-italic (see Text styles).
- Rounded rectangles.
- Conversion of millimetres and world coordinates.
- Measurement of text widths (screen and PostScript).

Unix:

- Use the sendpraat program for sending messages to running Praat programs.

Mac:

- Praat looks best with the new and beautiful System 8.

Praat 3.5, May 27, 1997

New editors:

- **TextGridEditor** replaces and extends LabelEditor: edit points as well as intervals.
- **AnalysisEditor** replaces and extends PsolaEditor: view pitch, spectrum, formant, and intensity analyses in a single window, and allow pitch and duration resynthesis by **PSOLA** and more (would be undone in 3.9.19).
- **SpectrumEditor** allows you to view and edit spectra.

Praat shell:

- **History mechanism** remembers all the commands that you have chosen, and allows you to put them into a script.
- **ScriptEditor** allows you to edit and run any Praat script, and to put it under a button.
- All added and removed buttons are remembered across sessions.
- **ButtonEditor** allows you to make buttons visible or invisible.

Evaluations:

- In his 1996 doctoral thesis, Henning Reetz compared five pitch analysis routines; Sound: To Pitch (ac)... appeared to make the fewest errors. H. Reetz (1996): *Pitch Perception in Speech: a Time Domain Approach*, Studies in Language and Language Use **26**, IFOTT, Amsterdam (ICG Printing, Dordrecht).

Documentation:

- 140 more man pages (now 330).
- Tables and pictures in manual.
- Printing the entire manual.
- Logo.

New classes:

- Labelling & segmentation: **TextGrid**, **IntervalTier**, **TextTier**.
- Analysis & manipulation: **Analysis**.
- Statistics: **TableOfReal**, **Distributions**, **Transition**

File formats:

- Read and write rational numbers in text files.
- Read 8-bit .au sound files.
- Read and write raw 8-bit two's-complement and offset-binary sound files.

Audio:

- 16-bit interactive Sound I/O on Mac.
- Record sounds at 9.8 kHz on SGI.

New commands:

- Two more pitch-analysis routines.
- Sound to PointProcess: collect all maxima, minima, zero crossings.
- PointProcess: set calculus.
- TextGrid: extract time-point information.
- Compute pitch or formants at given time points.
- Put pitch, formants etc. in tables en get statistics.
- Many more...

Macintosh:

- 16-bit interactive sound I/O.
- Fast and interpolating spectrogram drawing.
- Phonetic Mac screen font included in source code (as a fallback to using SIL Doulos IPA).
- Keyboard shortcuts, text editor, help under question mark, etc.

Praat 3.3, October 6, 1996

- Documentation: hypertext help browser, including the first 190 man pages.
- New editors: class **TextTier** for labelling times instead of intervals.
- New actions: **Formant**: Viterbi tracker, Statistics menu, Scatter plot.
- Evaluation: For HNR analysis of speech, the cross-correlation method, which has a sensitivity of 60 dB and a typical time resolution of 12 milliseconds, must be considered better than the autocorrelation method, which has a better sensitivity (80 dB), but a much worse time resolution (30 ms). For pitch analysis, the autocorrelation method still beats the cross-correlation method because of its better resistance against noise and echos, and despite its marginally poorer resolution (15 vs. 12 ms).
- User preferences are saved across sessions.
- The phonetic X screen font included in the source code.
- Xwindows resources included in the source code
- Graphics: eight colours, small caps, text rotation.
- File formats: Sun/NexT mu-law files, raw matrix text files, Xwaves mark files.
- Accelerations: keyboard shortcuts, faster dynamic menu, Shift-OK keeps file selector on screen.
- Class changes: **StylPitch** and **MarkTier** are now called **PitchTier** and **TextTier**, respectively. Old files can still be read.
- Script warning: all times in dialogs are in seconds now: milliseconds have gone.

Praat 3.2, April 29, 1996

- Sound I/O for HPUNIX, Sun Sparc 5, and Sun Sparc LX.
- Cross-correlation pitch and HNR analysis.
- Facilities for generating tables from scripts.
- Editing and playing stylized pitch contours and point processes.
- PSOLA pitch manipulation.
- Spectral smoothing techniques: cepstrum and LPC.
- Time-domain pitch analysis with jitter measurement.
- Read and write Bell-Labs sound files and Kay CSL audio files.
- Replaced IpaTimes font by free SILDoulos-IPA font, and embedded phonetic font in PostScript picture.
- Completed main phonetic characters.

Praat 3.1, December 5, 1995

- Add and remove buttons dynamically.
- DataEditor (Inspect button).
- Initialization scripts.
- Logarithmic axes.
- Call remote ADDA server directly.

To do

- ManipulationEditor: LPC manipulation of duration and intensity.
- TextGrid & Sound: Extract intervals with margins.
- Spectrum: draw power, re, im, phase.
- Formant: To Spectrum (slice)... (combines Formant-to-LPC and LPC-to-Spectrum-slice)
- Read and/or write Matlab files, MBROLA files, Xwaves files, CHAT files.
- Matrix: draw numbers.
- Fractions with $\frac{a}{b}$.
- More fonts for manual.
- Move objects up and down list.
- Spectrogram cross-correlation.
- Spectral moments (done).
- Labels in AIFC file.
- Improve scrolling and add selection in hyperpages.
- Segment spectrograph?
- Phoneme-to-articulation conversion??

Known bugs in the Macintosh version

- (small) Pause window modal.
- (small) Cascade buttons grey after suspend+resume during progress window.
- Movie window cannot be closed.

Known bugs in the Windows version

- Cannot stand infinitesimal zooming in SpectrogramEditor.
- Clipboards with greys sometimes become black-and-white after use of colour.

Known bugs in the Unix versions

- (small) Motif messaging windows should have no parent and be modeless.

Known bugs in the Linux version

- Sounds shorter than 200 ms do not always play (workaround: add zeroes in prefs).
- Keyboard shortcuts do not work if NumLock is on.
- Progress window does not always disappear.

Known bugs in the Solaris version

- (serious for some) File names run out of the window in some Motif versions.

Known bugs in the HP version

- (serious for some) Sound recording should be from audioserver instead of local.
-

© *ppgb*, September 14, 2004

Record mono Sound...

A command in the New menu to record a Sound. Creates a SoundRecorder window, except on very old Macintoshes with 8-bit audio, where it presents a native Macintosh sound-recorder window.

Links to this page

- [Intro 1.1. Recording a sound](#)
-

© *ppgb*, December 12, 2002

New menu

The **New menu** is one of the menus in the Object window. You use this menu to create new objects from scratch. It contains the following commands:

- Record mono Sound...
- Record stereo Sound...
- Create Sound... (from a formula)
- Create Sound from tone complex...
- Create Sound from gamma-tone...
- Create Sound from Shepard tone...
- Create Matrix... (from a formula)
- Create simple Matrix... (from a formula)
- Create empty PointProcess...
- Create Poisson process...
- Create PitchTier...
- Create DurationTier...
- Create IntensityTier...
- Create FormantTier...
- Create Strings as file list...
- Create TextGrid...
- OT learning tutorial
- Create tongue-root grammar...

To create new objects from files on disk, use the Read menu instead. Objects can also often be create from other objects, with commands that start with **To**.

Links to this page

- Add to fixed menu...
- Articulatory synthesis
- Formulas
- Formulas 1.6. Formulas in settings windows
- Formulas 1.7. Formulas for creation
- Intro 1.1. Recording a sound
- Intro 1.3. Creating a sound from a formula
- Intro 7. Annotation
- Intro 8.3. Manipulation of intensity
- Objects
- OT learning 2.1. Viewing a grammar
- OT learning 2.3. Defining your own grammar
- OT learning 2.6. Variable output
- OT learning 3.1. Data from a pair distribution

- OT learning 3.2. Data from another grammar
 - OT learning 7. Learning from overt forms
 - OTAnyGrammar examples
 - OTGrammar
 - ScriptEditor
 - SoundRecorder
-

© *ppgb*, December 12, 2002

Object window

One of the two main windows in the PRAAT program.

Subdivision

To the left: the List of Objects.

To the right: the Dynamic menu.

Fixed buttons

The following buttons appear below the List of Objects:

- Rename...
- Info
- Copy...
- Remove
- Inspect

Menus

The Object window contains several fixed menus: the **Control** (or **Praat**), **New**, **Read**, and **Help** menus. It also contains the **Write** menu, whose contents vary with the kinds of selected objects, and must, therefore, be considered part of the dynamic menu.

The ‘Control’ menu (on MacOS X: the Praat menu)

- (Run script...)
- New Praat script: creates an empty ScriptEditor
- Open Praat script...: creates a ScriptEditor with a script from disk
- The **Goodies submenu**: for doing things (like using the Calculator) that do not create new objects and do not depend on the kinds of selected objects.
- The **Preferences submenu**: for program-wide preferences, like audio input and output settings.
- **Buttons...**: raises a ButtonEditor
- (Add menu command...)
- (Add action command...)
- Quit

Other menus

- The New menu: for creating objects from scratch.
- The Read menu: for reading objects from file into memory.
- The Write menu: for writing objects from memory to file.

- The **Help menu**: for viewing the manual.

Links to this page

- Action commands
 - Add to dynamic menu...
 - Add to fixed menu...
 - Calculator...
 - Control menu
 - Fixed menu commands
 - Hidden commands
 - Intro 1.1. Recording a sound
 - Intro 2. What to do with a sound
 - Objects
 - Scripting 1. My first script
 - TextGridEditor
-

© *ppgb*, May 28, 2003

SoundRecorder

One of the windows in PRAAT: a tool for recording a 16-bit sampled Sound. It appears on your screen if you choose Record mono Sound... or Record stereo Sound... from the New menu.

Depending on your system, the SoundRecorder window may allow you to choose the sampling frequency, the gain, and the input device (microphone, line, or digital). The sound input level is monitored continuously with two meters.

Usage

To record the sound, use the Record and Stop buttons in the SoundRecorder window. Click *To list* to copy the recorded sound to the List of Objects (or *Left to list* or *Right to list* to copy the left or right channel if you have a stereo sound). The name of the resulting Sound object will be taken from the text field next to the button clicked.

Size of the recording buffer

The size of the recording buffer determines how many seconds of sound you can record. For instance, if the recording buffer is 20 megabytes (the standard value), you can record 220 seconds in stereo (440 seconds in mono) at a sampling frequency of 22050 Hz, or 110 seconds in stereo (220 seconds in mono) at a sampling frequency of 44100 Hz. You can change the size of the recording buffer with **Sound input prefs...** from the Preferences submenu in the Control menu.

If you recorded a very long sound, it is probable that you cannot copy it to the list of objects. In such a case, you can still write the sound to disk with one of the Write commands in the File menu. You can then open such a long sound file in PRAAT with Open long sound file... from the Read menu.

Recording sounds on MacOS X

You can record from the combined microphone / line input. On some computers, these are separate.

Note that in MacOS X you cannot record from the internal CD. This is because the system provides you with something better. If you open the CD in the Finder, you will see the audio tracks as AIFC files! To open these audio tracks in PRAAT, use Read from file... or Open long sound file....

Calibration

Your computer's sound-recording software returns integer values between -32768 and 32767. Praat divides them by 32768 before putting them into a Sound object, so that the values in the Sound objects are always between -1 and +1.

The Praat program considers these numbers to be in units of Pascal.

If you record from your small Silicon Graphics microphone, and set your input-gain sliders at maximum sensitivity, the values in the resulting Sound object are smaller than the real sound pressure at the microphone. The difference is 4 dB for an Indigo, and 24 dB for an Indy. Therefore, the values in the Sound object represent the actual sound pressure at a distance of 1.6 times (Indigo) or 16 times (Indy) the distance between your mouth and your microphone at the time of recording.

For a speaker who holds the microphone at 30 centimetres (Indigo) or 3 centimetres (Indy), the dB values of sounds recorded from the microphone represent the actual sound pressure at 48 centimetres. To get the sound pressure at 1 metre, you divide by two (e.g., with `Formula... self/2`).

Links to this page

- [Intro 1.1. Recording a sound](#)
- [Intro 2.1. Writing a sound to disk](#)
- [Play](#)
- [Types of objects](#)
- [What's new?](#)

© *ppgb*, March 28, 2004

Record stereo Sound...

A command in the New menu to record a Sound. Creates a SoundRecorder window.

Links to this page

- [Intro 1.1. Recording a sound](#)
-

© *ppgb*, December 12, 2002

List of Objects

A list in the left-hand part of the Object window.

Purpose

If you select one or more objects in this list, the possible actions that you can perform with the selected objects will appear in the Dynamic menu.

How to select objects

To select one object (and deselect all the others), click on the object.

To extend the selection, drag the mouse (Unix, Windows) or use Shift-click (all systems).

To change the selection of one object (without changing the others), use Control-click (Unix, Windows) or Command-click (Macintosh).

Links to this page

- [ButtonEditor](#)
- [Editors](#)
- [Feedforward neural networks 2. Quick start](#)
- [Formulas 7. Attributes of objects](#)
- [Intro 1.1. Recording a sound](#)
- [Intro 2. What to do with a sound](#)
- [Intro 2.1. Writing a sound to disk](#)
- [Intro 3.4. The Spectrogram object](#)
- [Intro 3.7. The Spectrum object](#)
- [Intro 4.4. The Pitch object](#)
- [Intro 5.4. The Formant object](#)
- [Intro 6.4. The Intensity object](#)
- [Read menu](#)
- [TextGridEditor](#)

© *ppgb*, October 24, 1998

Edit

A command in the Dynamic menu of several types of objects.

This command puts an editor window on the screen, which shows the contents of the selected object. This window will allow your to view and modify the contents of this object.

Links to this page

- [Intro 1.1. Recording a sound](#)
- [Intro 2.2. Viewing and editing a sound](#)
- [Intro 3.1. Viewing a spectrogram](#)
- [Intro 3.7. The Spectrum object](#)
- [Intro 4.1. Viewing a pitch contour](#)
- [Intro 4.4. The Pitch object](#)
- [Intro 5.1. Viewing formant contours](#)
- [Intro 6.1. Viewing an intensity contour](#)
- [Intro 7. Annotation](#)
- [Intro 8.1. Manipulation of pitch](#)
- [Intro 8.2. Manipulation of duration](#)
- [Intro 8.3. Manipulation of intensity](#)
- [Scripting 7.2. Scripting an editor from within](#)

© *ppgb*, April 14, 2001

Read from file...

One of the commands in the Read menu.

Purpose

To read one or more objects from a file on disk.

Behaviour

Many kinds of files are recognized:

1. Text files that are structured as described under Write to text file...; these can contain an object of any class, or a collection of objects.
2. Files that were produced by Write to binary file... (any class).
3. Files in a LISP text format (only for classes that can be written to a LISP file).
4. Files that were made recognizable by the libraries built on Praat. For instance, the phonetics library adds recognizers for many kinds of sound files.

If the file contains more than one object, these objects will appear in the list, and their names will be the same as the names that they had when they were saved with **Write to text file...** or **Write to binary file...**

Examples

- If the file contains only one Pitch object and is called "hallo.pit", an object with the name "Pitch hallo" will appear in the list of objects. You may have more objects with the same name.
- If the file contains one object of class Pitch, named "hallo", and one object of class Polygon, named "kromme", there will appear two objects in the list, called "Pitch hallo" and "Polygon kromme".

Links to this page

- [AIFF and AIFC files](#)
- [Confusion](#)
- [Dissimilarity](#)
- [ExperimentMFC](#)
- [How to concatenate sound files](#)
- [Intro 1.2. Reading a sound from disk](#)
- [Intro 4.4. The Pitch object](#)
- [Intro 5.4. The Formant object](#)
- [Intro 7. Annotation](#)
- [Macintosh sound files](#)
- [ManPages](#)
- [Matrix](#)
- [NIST files](#)

- OT learning 2.2. Inside the grammar
 - OT learning 7. Learning from overt forms
 - Sesam/LVS files
 - Sound files 2.4. NeXT/Sun (.au) files
 - Sound files 3. Files that Praat can read
 - SoundRecorder
 - SpellingChecker
 - Strings
 - What's new?
 - WordList
 - Write menu
 - Write to short text file...
-

© *ppgb*, September 11, 1997

Create Sound...

A command in the New menu to create a Sound with a specified duration and sampling frequency, filled with values from a formula.

See the Formulas tutorial for explanations and examples.

Links to this page

- [Advanced spectrogram settings...](#)
- [ExperimentMFC](#)
- [Formulas 1.6. Formulas in settings windows](#)
- [Formulas 1.7. Formulas for creation](#)
- [Formulas 7. Attributes of objects](#)
- [Intro 1.3. Creating a sound from a formula](#)
- [Intro 3.2. Configuring the spectrogram](#)
- [Spectrum: Get centre of gravity...](#)

© *ppgb*, March 31, 2004

Dynamic menu

A column of buttons in the right-hand part of the Object window, plus the **Write** menu in the Object window.

If you select one or more objects in the list, the possible actions that you can perform with the selected objects will appear in the dynamic menu. These actions can include editing, writing, drawing, conversions to other types (including analysis and synthesis), and more.

Example of analysis:

Record a Sound, select it, and click on **To Pitch...** This will create a new Pitch object and put it in the list of objects. You can then edit, write, and draw this Pitch object.

Example of synthesis:

Create a **Speaker**, create and edit an **Artword**, and click on **To Sound...**

Links to this page

- Action commands
- Draw submenu
- Edit
- Editors
- Formants & LPC submenu
- History mechanism
- Intro 2. What to do with a sound
- List of Objects
- Modify
- Periodicity submenu
- Scripting 7. Scripting the editors

© ppgb, October 24, 1998

File menu

One of the menus in all editors, in the manual, and in the Picture window.

Links to this page

- [Intro 2.1. Writing a sound to disk](#)
 - [Intro 8.2. Manipulation of duration](#)
 - [ScriptEditor](#)
 - [Scripting 7.2. Scripting an editor from within](#)
-

© *ppgb*, December 4, 2002

Write menu

One of the menus in the Object window.

Purpose

With the Write menu, you write one or more selected objects from memory to a file on disk. The data can be read in again with one of the commands in the Read menu (most often simply with Read from file...).

Usage: save your work

You will often choose a command from this menu just before clicking the Remove button or choosing the Quit command.

Fixed commands

If no object is selected, the Write menu is empty. If any object is selected, it will at least contain the following commands:

- Write to console
- Write to text file...
- **Write to short text file...**
- Write to binary file...

Dynamic commands

Depending on the class of the selected object, the following commands may be available in the Write menu:

- Add to dynamic menu...
- How to concatenate sound files
- Intro 2.1. Writing a sound to disk
- Intro 4.4. The Pitch object
- Intro 5.4. The Formant object
- Sounds: Concatenate
- Write to short text file...

© ppgb, September 11, 1997

Open long sound file...

A command in the Read menu that creates a LongSound object.

The file will be opened for reading only. The file stays open until you remove the LongSound object.

Links to this page

- [How to concatenate sound files](#)
- [Intro 2.2. Viewing and editing a sound](#)
- [Sound files 3. Files that Praat can read](#)
- [SoundRecorder](#)

© *ppgb*, July 30, 1998

LongSound

One of the types of objects in PRAAT. See the Sound files tutorial.

A LongSound object gives you the ability to view and label a sound file that resides on disk. You will want to use it for sounds that are too long to read into memory as a Sound object (typically, a few minutes).

How to create a LongSound object

You create a LongSound object with Open long sound file... from the Read menu.

What you can do with a LongSound object

You can write a LongSound object to a new sound file, perhaps in a different format (AIFF, AIFC, WAV, NeXT/Sun, NIST) with the commands in the Write menu. You can also concatenate several LongSound objects in this way. See How to concatenate sound files.

How to view and edit a LongSound object

You can view a LongSound object in a LongSoundEditor by choosing LongSound: View. This also allows you to extract parts of the LongSound as Sound objects, or write these parts to a sound file. There are currently no ways to actually change the data in the file.

How to annotate a LongSound object

You can label and segment a LongSound object after the following steps:

1. Select the LongSound object.
2. Choose LongSound: To TextGrid... and specify your tiers.
3. Select the resulting TextGrid object together with the LongSound object, and click **Edit**.

A TextGridEditor will appear on the screen, with a copy of the LongSound object in it.

Limitations

The length of the sound file is limited to 2 gigabytes, which is 3 hours of CD-quality stereo, or 12 hours 16-bit mono sampled at 22050 Hz.

Links to this page

- [Get number of samples](#)
- [Get sample number from time...](#)
- [Get time from sample number...](#)

- Intro 2.2. Viewing and editing a sound
 - Intro 3.1. Viewing a spectrogram
 - Intro 4.1. Viewing a pitch contour
 - Intro 5.1. Viewing formant contours
 - Intro 6.1. Viewing an intensity contour
 - Intro 7. Annotation
 - Sound files 3. Files that Praat can read
 - SpellingChecker
 - What's new?
 - Write to AIFC file...
 - Write to AIFF file...
 - Write to NeXT/Sun file...
 - Write to NIST file...
 - Write to WAV file...
-

© ppgb, January 13, 2004

Extract visible spectrogram

One of the commands in the Spectrogram menu of the SoundEditor and the TextGridEditor.

See Intro 3. Spectral analysis

© *ppgb*, March 16, 2003

Show spectrogram

One of the commands in the Spectrogram menu of the SoundEditor and the TextGridEditor.

See Intro 3. Spectral analysis.

Links to this page

- [Intro 3.1. Viewing a spectrogram](#)

© *ppgb*, March 16, 2003

Sound: To Spectrogram...

A command that creates a Spectrogram from every selected Sound object. It performs a *short-term spectral analysis*, which means that for a number of time points in the Sound, Praat computes an approximation of the spectrum at that time. Each such spectrum is called an *analysis frame*.

For tutorial information, see Intro 3. Spectral analysis.

Arguments

Window length

the duration of the analysis window. If this is 0.005 seconds, Praat uses for each frame the part of the sound that lies between 0.0025 seconds before and 0.0025 seconds after the centre of that frame (for Gaussian windows, Praat actually uses a bit more than that). The window length determines the *bandwidth* of the spectral analysis, i.e. the width of the horizontal line in the spectrogram of a pure sine wave. For a Gaussian window, the -3 dB bandwidth is $2 \cdot \sqrt{6 \cdot \ln(2)} / (\pi \cdot \textit{Window length})$, or $1.2982804 / \textit{Window length}$. To get a ‘broad-band’ spectrogram (bandwidth 260 Hz), set *Window length* to 5 milliseconds; to get a ‘narrow-band’ spectrogram (bandwidth 43 Hz), set it to 30 milliseconds. The other window shapes give slightly different values.

Maximum frequency

the maximum frequency subject to analysis, e.g. 5000 Hertz. If it is higher than the Nyquist frequency of the Sound (which is half its sampling frequency), some values in the result will be zero (and will be drawn in white by Spectrogram: Paint...).

Time step

the distance between the centres of subsequent frames, e.g. 0.002 seconds. This determines the number of frames of the resulting Spectrogram. For instance, if the Sound is 1 second long, and the time step is 2 milliseconds, the Spectrogram will consist of almost 500 frames (not *exactly* 500, because no reliable spectrum can be measured near the beginning and end of the sound). See below for cases in which the time step of the resulting Spectrogram is different from what you supply here.

Frequency step

the frequency resolution, e.g. 20 Hertz. This determines the number of frequency bands (*bins*) of the resulting Spectrogram. For instance, if the *Maximum frequency* is 5000 Hz, and the frequency step is 20 Hz, the Spectrogram will consist of 250 frequency bands. See below for cases in which the frequency step of the resulting Spectrogram is different from what you supply here.

Window shape

determines the shape of the analysis window. You can choose from: Gaussian, Square (none, rectangular), Hamming (raised sine-squared), Bartlett (triangular), Welch (parabolic), and Hanning (sine-squared). The Gaussian window is superior, as it gives no sidelobes in your spectrogram; it analyzes a factor of 2 slower than the other window shapes, because the analysis is actually performed on twice as many samples per frame.

For purposes of computation speed, Praat may decide to change the time step and the frequency step. This is because the time step never needs to be smaller than $1/(8\sqrt{\pi})$ of the window length, and the frequency step never needs to be smaller than $(\sqrt{\pi})/8$ of the inverse of the window length. For instance, if the window length is 29 ms, the actual time step will be never be less than $29/(8\sqrt{\pi}) = 2.045$ ms. And if the

window length is 5 ms, the actual frequency step will never be less than $(\sqrt{\pi})/8/0.005 = 44.32$ Hz.

Tests of the bandwidth

You can check the bandwidth formula with the following procedure:

```
! create a 1000-Hz sine wave, windowed by a 0.2-seconds Gaussian
window.
Create Sound... gauss 0 1 22050 sin(2*pi*1000*x) *
exp(-3*((x-0.5)/0.1)^2)
! compute its spectrum and look at its bandwidth
To Spectrum (fft)
Draw... 980 1020 20 80 yes
Marks bottom every... 1 2 yes yes yes
Marks left every... 1 2 yes yes yes
! now you should see a peak at 1000 Hz with a 3 dB bandwidth of 7 Hz
(20 dB: 17 Hz)
! more precise: compute the position and width of the peak, and write
them to the console
Formula... if x<980 or x>1020 then 0 else self fi
To Formant (peaks)... 20
Write to console
! now you should be able to read that a peak was found at 999.99982 Hz
! with a bandwidth of 6.497 Hz; the theory above predicted 6.491 Hz
! The same, windowed by a 0.1-seconds Hamming window.
Create Sound... Hamming 0 1 22050 if x<0.4 or x>0.6 then 0 else
sin(2*pi*1000*x)*(0.54+0.46*cos(pi*(x-0.5)/0.1)) fi
To Spectrum (fft)
Formula... if x<970 or x>1030 then 0 else self fi
To Formant (peaks)... 20
Write to console
! peak at 999.99817 Hz, 3 dB bw 6.518 Hz, 20 dB bw 15 Hz, zero bw 20
Hz, sidelobe -42 dB
! The same, windowed by a 0.1-seconds rectangular window.
Create Sound... rectangular 0 1 22050 if x<0.4 or x>0.6 then 0 else
sin(2*pi*1000*x) fi
To Spectrum (fft)
Formula... if x<970 or x>1030 then 0 else self fi
To Formant (peaks)... 20
Write to console
! peak at 999.99506 Hz, 3 dB bw 4.440 Hz, 20 dB bw 27 Hz, zero bw 10
Hz, sidelobe -14 dB
! The same, windowed by a 0.1-seconds Hanning window.
Create Sound... Hanning 0 1 22050 if x<0.4 or x>0.6 then 0 else
sin(2*pi*1000*x)*(0.5+0.5*cos(pi*(x-0.5)/0.1)) fi
To Spectrum (fft)
```

```

Formula... if x<970 or x>1030 then 0 else self fi
To Formant (peaks)... 20
Write to console
! peak at 999.99945 Hz, 3 dB bw 7.212 Hz, 20 dB bw 16 Hz, zero bw 20
Hz, sidelobe -31 dB
! The same, windowed by a 0.1-seconds triangular window.
Create Sound... triangular 0 1 22050 if x<0.4 or x>0.6 then 0 else
sin(2*pi*1000*x)*(1-abs((x-0.5)/0.1)) fi
To Spectrum (fft)
Formula... if x<970 or x>1030 then 0 else self fi
To Formant (peaks)... 20
Write to console
! peak at 999.99933 Hz, 3 dB bw 6.384 Hz, 20 dB bw 15 Hz, zero bw 20
Hz, sidelobe -26 dB
! The same, windowed by a 0.1-seconds parabolic window.
Create Sound... parabolic 0 1 22050 if x<0.4 or x>0.6 then 0 else
sin(2*pi*1000*x)*(1-((x-0.5)/0.1)2) fi
To Spectrum (fft)
Formula... if x<970 or x>1030 then 0 else self fi
To Formant (peaks)... 20
Write to console
! peak at 999.99921 Hz, 3 dB bw 5.786 Hz, 20 dB bw 12 Hz, zero bw 15
Hz, sidelobe -21 dB

```

Links to this page

- [Fast Fourier Transform](#)
- [Intro 3.4. The Spectrogram object](#)
- [Sound: To Formant \(burg\)...](#)

© *ppgb*, March 30, 2004

Spectrogram

One of the types of objects in PRAAT. For tutorial information, see Intro 3. Spectral analysis.

A Spectrogram object represents an acoustic time-frequency representation of a sound: the power spectral density $P(f, t)$, expressed in Pa^2/Hz . It is sampled into a number of points centred around equally spaced times t_i and frequencies f_j .

Inside a Spectrogram

With Inspect, you will see the following attributes:

xmin

starting time, in seconds.

xmax

end time, in seconds.

nx

the number of times (≥ 1).

dx

time step, in seconds.

x1

the time associated with the first column, in seconds. This will usually be in the range $[xmin, xmax]$. The time associated with the last column (i.e., $x1 + (nx - 1) dx$) will also usually be in that range.

ymin

lowest frequency, in Hertz. Normally 0.

ymax

highest frequency, in Hertz.

ny

the number of frequencies (≥ 1).

dy

frequency step, in Hertz.

y1

the frequency associated with the first row, in Hertz. Usually $dy / 2$. The frequency associated with the last row (i.e., $y1 + (ny - 1) dy$) will often be $ymax - dy / 2$.

z_{ij} , $i = 1 \dots ny$, $j = 1 \dots nx$

the power spectral density, in Pa^2/Hz .

Links to this page

- Formulas 1.8. Formulas for modification
- Formulas 7. Attributes of objects
- Intro 3.4. The Spectrogram object
- Sound: To Spectrogram...
- Source-filter synthesis

- spectro-temporal representation
 - Spectrogram: Formula...
 - Spectrogram: Paint...
 - Spectrogram: To Spectrum (slice)...
 - Spectrum: To Spectrogram
 - time domain
-

© *ppgb*, March 16, 2003

LongSoundEditor

One of the Editors in PRAAT, for viewing a LongSound object.

This viewer allows you:

- to view and hear parts of the sound as it is on disk;
- to copy a selected part as a Sound object to the list of objects, so that you can perform analyses on it or save it to a smaller sound file;
- to copy a selected part as a Sound object to the Sound clipboard, so that you can paste it into another Sound object that you are viewing in a SoundEditor.

To label and segment the LongSound object, use the TextGridEditor instead (see LongSound).

The display and playback of the samples is restricted to 20 seconds at a time, for reasons of speed; the sound file itself can contain several hours of sound.

Links to this page

- [Formant analysis...](#)
- [Intro 3.1. Viewing a spectrogram](#)
- [LongSound: View](#)
- [Types of objects](#)

© *ppgb*, March 16, 2003

spectro-temporal representation

A representation (of a sound signal, for instance) as some sort of intensity as a function of time and frequency. In PRAAT, we have the Spectrogram, which is acoustic energy density as a function of time in seconds and frequency in Hz, and the Cochleagram, which is basilar membrane excitation as a function of time in seconds and frequency in Bark.

For tutorial information, see Intro 3.1. Viewing a spectrogram.

© *ppgb*, March 14, 2003

time

In normal life, time is how late the watch says it is.

In PRAAT, this definition is largely irrelevant. Sound files rarely tell us the absolute time of recording. So when you read a sound file into PRAAT and click **Edit**, you will see that the Sound starts at a time of 0 seconds, and if its duration is 3.5 seconds, you will see that the Sound finishes at a time of 3.5 seconds.

Besides sounds, many other types of objects in PRAAT have a time scale as well: spectrograms, pitch contours, formant contours, point processes, and so on. None of these are required to have a time domain that starts at 0 seconds. In the Sound editor window, for example, you can select the part that runs from 1.4 to 1.7 seconds, and "extract" it to the Objects window while "preserving the times". The resulting Sound object will have a starting time of 1.4 seconds and a finishing time of 1.7 seconds, as you can see when you click **Edit**. Spectrograms and pitch contours that you create from this sound will also have a time domain from 1.4 to 1.7 seconds. This time domain is preserved if you save these objects to a text file or to a binary file and read them into PRAAT again later. Only if you save the Sound object to an audio file (WAV, AIFF), the time information is not preserved in that file; if you read such an audio file into PRAAT again, the time domain of the new Sound object will run from 0 to 0.3 seconds.

In order to prevent confusion, PRAAT always requires times to be expressed in seconds. So if you want to supply a window length of 5 milliseconds (5 ms), you fill in 0.005 or 5e-3. For 83.2 microseconds (83.2 μ s), you say 0.0000832, or better 83.2e-6 or 8.32e-5.

On a clock, time runs around in circles. In PRAAT's editor windows, time runs from left to right. You can often see only a part of the time scale in the window. To see another part, you *scroll* backward or forward.

Links to this page

- [Intro 3.1. Viewing a spectrogram](#)
- [spectro-temporal representation](#)

© ppgb, March 14, 2003

frequency

Frequency is how often something happens in a certain time, for instance the number of times the PRAAT home page www.praat.org is visited every day.

In PRAAT, frequency is the number of vibration cycles per second. Although one can sometimes see the abbreviation *cps*, PRAAT always uses Hz (short for *Hertz*), which means the same.

Unfortunately, there are two very distinct kinds of vibrations in speech analysis. For pitch, frequency is the number of glottal cycles per second, and for spectral analysis, frequency is the number of sine wave cycles per second. Quite some bit of the training of an acoustic phonetician goes into the understanding of the difference between the ideas behind F0 and F1, and many years can be spent on understanding the influence they have on each other in production, acoustics, perception, or measurement...

In order to prevent confusion, PRAAT always requires frequency to be expressed in Hz. So if you want to supply a sampling frequency of 20 kiloHertz (20 kHz), you fill in 20000 or 2e4 or 20e3. If you want to switch off pre-emphasis in some spectral analysis, you supply 1 GigaHertz (GHz) for its "from-frequency", by typing 1e9.

In PRAAT editor windows, frequency usually runs from bottom to top, since time already has to run from left to right. This goes for spectrograms, pitch contours, and formant contours. In spectral slices, frequency runs from left to right, since these have no time axis.

Links to this page

- [Intro 3.1. Viewing a spectrogram](#)
- [spectro-temporal representation](#)

© *ppgb*, March 14, 2003

Ladefoged (2001)

Peter Ladefoged (2001). *Vowels and consonants: an introduction to the sounds of languages*. Oxford: Blackwell.

A very readable introduction to phonetics, mainly acoustic and articulatory. Has lots of spectrograms of the sounds of the world's languages. Comes with a CD that has all those sounds and includes training material for transcription (from another book).

For a more encyclopaedic treatment of the sounds of the world's languages, see Ladefoged & Maddieson (1996) instead.

Links to this page

- [Intro 3.1. Viewing a spectrogram](#)
-

© ppgb, March 16, 2003

Ladefoged & Maddieson (1996)

Peter Ladefoged & Ian Maddieson (1996). *The sounds of the world's languages*. Oxford: Blackwell.

An extensive reference source for the articulation and acoustics of 'all' vowels and consonants that occur in the world's languages. If you don't find the answer in this book, you will find the answer in the articles referred to in this book. The book uses lots of spectrograms, palatograms, and other techniques.

The book is not an introductory text. For that, see Ladefoged (2001) instead.

Links to this page

- [Intro 3.1. Viewing a spectrogram](#)
-

© ppgb, March 16, 2003

Spectrogram settings...

A command in the Spectrogram menu of the SoundEditor and TextGridEditor windows. See Intro 3.2. Configuring the spectrogram.

© *ppgb*, March 16, 2003

Advanced spectrogram settings...

Optimization

Number of time steps

the maximum number of points along the time window for which PRAAT has to compute the spectrum. If your screen is not wider than 1200 pixels, then the standard of 1000 is appropriate, since there is no point in computing more than one spectrum per one-pixel-wide vertical line. If you have a really wide screen, you may see improvement if you raise this number to 1500.

Number of frequency steps

the maximum number of points along the frequency axis for which PRAAT has to compute the spectrum. If your screen is not taller than 768 pixels, then the standard of 250 is appropriate, since there is no point in computing more than one spectrum per one-pixel-height horizontal line. If you have a really tall screen, you may see improvement if you raise this number.

For purposes of computation speed, Praat may decide to change the time step and the frequency step. This is because the time step never needs to be smaller than $1/(8\sqrt{\pi})$ of the window length, and the frequency step never needs to be smaller than $(\sqrt{\pi})/8$ of the inverse of the window length. For instance, if the window length is 30 ms, the actual time step will never be less than $30/(8\sqrt{\pi}) = 2.116$ ms. And if the window length is 5 ms, the actual frequency step will never be less than $(\sqrt{\pi})/8/0.005 = 44.32$ Hz.

Spectrogram analysis settings

Method

there is currently only one method available in this window for computing a spectrum from a sound: the Fourier transform.

Window shape

the shape of the analysis window. To compute the spectrum at, say, 3.850 seconds, samples that lie close to 3.850 seconds are given more weight than samples further away. The relative extent to which each sample contributes to the spectrum is given by the window shape. You can choose from: Gaussian, Square (none, rectangular), Hamming (raised sine-squared), Bartlett (triangular), Welch (parabolic), and Hanning (sine-squared). The Gaussian window is superior, as it gives no *sidelobes* in your spectrogram (see below); it analyzes a factor of 2 slower than the other window shapes, because the analysis is actually performed on twice as many samples per frame.

Sidelobes; anybody wants to win a cake?

The Gaussian window is the only shape that we can consider seriously as a candidate for the analysis window. To see this, create a 1000-Hz sine wave with Create Sound... by typing $1/2 * \sin(2*\pi*1000*x)$ as the formula, then click **Edit**. If the window shape is Gaussian, the spectrogram will show a horizontal black line. If the window shape is anything else, the spectrogram will show many horizontal grey lines (*sidelobes*), which do not represent anything that is available in the signal. They are artifacts of the window shapes.

We include these other window shapes only for pedagogical purposes and because the Hanning and Hamming windows have traditionally been used in other programs before computers were as fast as they are now (a spectrogram is computed twice as fast with these other windows). Several other programs still use these inferior window shapes, and you are likely to run into people who claim that the Gaussian window has disadvantages. We promise such people a large cake if they can come up with sounds that look better with Hanning or Hamming windows than with a Gaussian window. An example of the reverse is easy to find; we have just seen one.

Spectrogram blackness settings

Autoscaling

Maximum (dB/Hz)

all parts of the spectrogram that have a power above *maximum* (after preemphasis) will be drawn in black. The standard maximum is 100 dB/Hz, but if *autoscaling* is on (which is the standard), PRAAT will use the maximum of the visible part of the spectrogram instead; this ensures that the window will always look well, but it also means that the blackness of a certain part of the spectrogram will change as you scroll.

Preemphasis (dB/octave)

determines the steepness of a high-pass filter, i.e., how much the power of higher frequencies will be raised before drawing, as compared to lower frequencies. Since the spectral slope of human vowels is approximately -6 dB per octave, the standard value for this setting is +6 dB per octave, so that the spectrum is flattened and the higher formants look as strong as the lower ones. When you raise the preemphasis, frequency bands above 1000 Hz will become darker, those below 1000 Hz will become lighter.

Dynamic compression

determines how much stronger weak spectra should be made before drawing. Normally, this parameter is between 0 and 1. If it is 0 (the standard value), there is no dynamic compression. If it is 1, all spectra will be drawn equally strong, i.e., all of them will contain frequencies that are drawn in black. If this parameter is 0.4 and the global maximum is at 80 dB, then a spectrum with a maximum at 20 dB (which will normally be drawn all white if the dynamic range is 50 dB), will be raised by $0.4 * (80 - 20) = 24$ dB, so that its maximum will be seen at 44 dB (thus making this frame visible).

Links to this page

- [Intro 3.2. Configuring the spectrogram](#)
-

© ppgb, September 17, 2003

Picture window

One of the two main windows in PRAAT.

File menu

- o Read from Praat picture file..., Write to Praat picture file...
- o Write to Mac PICT file..., Copy to clipboard
- o PostScript settings...
- o Write to EPS file...
- o Print...

Edit menu

- o Undo, Erase all

Margins menu

- o Draw inner box
- o Text left/right/top/bottom...
- o Marks left/right/top/bottom every...
- o One mark left/right/top/bottom...
- o Marks left/right/top/bottom...
- o Logarithmic marks left/right/top/bottom...
- o One logarithmic mark left/right/top/bottom...
- o Axes...

World menu

- o Text...
- o Axes...

Select menu

- o Select inner viewport..., Select outer viewport..., Viewport text...

Pen menu

Font menu

Links to this page

- [Add to fixed menu...](#)
- [Draw submenu](#)
- [File menu](#)
- [Fixed menu commands](#)
- [Formant: Draw tracks...](#)
- [Formant: Speckle...](#)
- [Hidden commands](#)
- [Intro 3.4. The Spectrogram object](#)
- [Intro 3.7. The Spectrum object](#)
- [Intro 4.4. The Pitch object](#)
- [Intro 5.4. The Formant object](#)
- [Intro 6.4. The Intensity object](#)
- [Matrix: Draw as squares...](#)

- Matrix: Paint cells...
 - OT learning 2.7. Tableau pictures
 - OTAnyGrammar examples
 - Phonetic symbols
 - Phonetic symbols: consonants
 - Phonetic symbols: diacritics
 - Phonetic symbols: vowels
 - Pitch: Draw...
 - PointProcess: Draw...
 - Scripting 1. My first script
 - Special symbols
 - Spectrogram: Paint...
 - Text styles
 - TextGrid
 - What's new?
 - Write to Windows metafile...
-

© *ppgb*, September 5, 2004

Spectrogram: Paint...

A command to draw the selected Spectrogram object(s) into the Picture window in shades of grey.

Arguments

From time, To time (seconds)

the time domain along the x axis.

From frequency, To frequency (Hertz)

the frequency domain along the y axis.

Dynamic range (dB)

The global maximum of the spectrogram (after preemphasis) will always be drawn in black; all values that are more than *Dynamic range* dB below this maximum (after dynamic compression) will be drawn in white. Values in-between have appropriate shades of grey.

Preemphasis (dB/octave)

determines the steepness of a high-pass filter, i.e., how much the power of higher frequencies will be raised before drawing, as compared to lower frequencies. Since the spectral slope of human vowels is approximately -6 dB per octave, the standard value for this parameter is +6 dB per octave, so that the spectrum is flattened and the higher formants look as strong as the lower ones.

Dynamic compression

determines how much stronger weak time frames should be made before drawing. Normally, this parameter is between 0 and 1. If it is 0, there is no dynamic compression. If it is 1, all time frames (vertical bands) will be drawn equally strong, i.e., all of them will contain frequencies that are drawn in black. If this parameter is 0.4 and the global maximum is at 80 dB, then a frame with a maximum at 20 dB (which will normally be drawn all white if the dynamic range is 50 dB), will be raised by $0.4 * (80 - 20) = 24$ dB, so that its maximum will be seen at 44 dB (thus making this frame visible).

Links to this page

- [Intro 3.4. The Spectrogram object](#)
- [Sound: To Spectrogram...](#)

© ppgb, September 16, 2003

TextGridEditor

One of the Editors in PRAAT, for editing a TextGrid object.

You can optionally include a copy of a Sound or LongSound in this editor, by selecting both the TextGrid and the Sound or LongSound before clicking **Edit**. The Sound or LongSound is shown in the upper part of the window, the tiers in the lower part. A text window at the top shows the text of the *selected* interval or point, i.e. the interval or point at the location of the cursor. All tiers are visible, and if you do not zoom in, all boundaries, points, and texts are visible, too. You can do many of the same things that you can do with a SoundEditor or LongSoundEditor.

Positioning the cursor or the selection marks

To position the cursor hair, click in the **Sound**, on a boundary, on a point, or inside an interval.

To select any part of the time domain, use the time selection mechanism; if you do this by clicking in a tier, the selected time domain will snap to the nearest boundary or point.

Creating new intervals, boundaries, points, or tiers

To create a new interval, create a new boundary in an interval tier.

To create a new boundary or point in a tier, click inside the cursor circle in that tier, or choose one of the commands in the Boundary/Point menu to insert a boundary at the cursor time on the selected tier (shortcut: Enter) or on any tier (shortcuts: Command-F1 through Command-F9). The original text in the interval that is split, is divided up between the two resulting intervals, depending on the position of the text cursor in the text window.

To create a new tier, choose **Add interval tier** or **Add point tier** from the **Tier** menu.

Playing an entire interval, or part of it

As in many other editors, you can play a stretch of sound by clicking in any of the rectangles around the drawing area.

To play an interval of an interval tier, you first click inside it. This will make the interval *selected*, which means that the visible part of the interval will be drawn in yellow. The cursor will be positioned at the start of the interval, and the time selection will comprise exactly the interval. This means that you can use the Tab key to play the interval. If you press it while a sound is playing, the Tab key will halt the playing sound, and the cursor will move to the time at which the sound stopped playing. This helps you to divide up a long sentence into parts that you can remember long enough to write them down.

The Tab key will play the selected interval.

Editing the text in an interval or at a point

To edit the label text of an interval or point:

1. Select that interval or point by clicking in or on it. The text currently in the interval or point will appear in the text window.
2. Just type the text, and use the mouse and the arrow keys to navigate the text window. Everything you type will become visible immediately in the text window as well as in the selected interval or point.

You can use all the Special symbols that you can use elsewhere in Praat, including mathematical symbols, Greek letters, superscripts, and phonetic symbols.

Selecting a tier

To select a tier, click anywhere inside it. Its number and name will be drawn in red, and a pointing finger symbol () will appear on its left.

Selecting a boundary or point

To select a boundary on an interval tier, click in its vicinity or inside the following interval; the boundary will be drawn in red. The text in the interval will appear in the text window.

To select a point on a point tier, click in its vicinity; it will be drawn in red. The text of the point will appear in the text window.

Moving one or more boundaries or points

To move a boundary or point to another time position, drag it with the mouse.

To move all the boundaries and points with the same time (on different tiers) to another time position, Shift-drag them.

To move boundaries or points to the exact time position of a boundary or point on an other tier, drag them into that other tier and into the vicinity of that boundary or point.

To move boundaries or points to the exact time position of the cursor, drag them into the vicinity of the cursor.

Removing a boundary, point, or tier

To remove a selected **boundary**, choose **Remove** from the **Boundary** menu. This creates a new interval which is the union of the two intervals originally adjoining the boundary; the new text of this interval is the concatenation of the two original texts, except if these were equal, in which case the new text equals both original texts.

To remove a selected **point**, choose **Remove** from the **Point** menu.

To remove a selected **tier**, choose **Remove entire tier** from the **Tier** menu.

Extracting a part of the sound

To copy the selected part of the Sound or LongSound as a Sound to the List of Objects, choose **Extract sound selection** from the **File** menu. You can specify whether you want the time domain of the resulting Sound to match the starting and finishing times of the selection or whether you want the time domain of the resulting Sound to start at zero seconds.

If you are viewing a LongSound, you can save the selected part of it to a 16-bit sound file (AIFF, AIFC, WAV, NeXT/Sun, NIST) with a command from the File menu.

Accelerations

To write the TextGrid object to a text file without going to the Object window: choose **Write TextGrid to text file...** from the **File** menu.

Searching

The Search menu contains the command **Find** (Command-F), which will allow you to specify a text whose first occurrence will then be looked for in the currently selected tier (starting from the currently selected text in the currently selected interval). The command **Find again** (Command-G) will search for the next occurrence of the same search text.

Checking the spelling

You can check the spelling of the intervals in your tiers by including a SpellingChecker object as you launch the editor: select TextGrid + (Long)Sound + SpellingChecker, then click Edit. The Search menu will contain the command **Check spelling** (Command-L), which will search for the next word that does not occur in its lexicon.

Links to this page

- [Advanced pitch settings...](#)
- [Extract visible formant contour](#)
- [Extract visible intensity contour](#)
- [Extract visible pitch contour](#)
- [Extract visible spectrogram](#)
- [Formant analysis...](#)
- [Get first formant](#)
- [Get pitch](#)
- [Get second formant](#)
- [IntervalTier](#)
- [Intro 3.5. Viewing a spectral slice](#)

- Intro 4.3. Querying the pitch contour
 - Intro 4.4. The Pitch object
 - Intro 5.3. Querying the formant contours
 - Intro 5.4. The Formant object
 - Intro 6.1. Viewing an intensity contour
 - Intro 7. Annotation
 - Log files
 - Phonetic symbols
 - Phonetic symbols: consonants
 - Phonetic symbols: diacritics
 - Phonetic symbols: vowels
 - Pitch settings...
 - Play
 - Show formant
 - Show intensity
 - Show pitch
 - Show pulses
 - Show spectrogram
 - Spectrogram settings...
 - TextTier
 - Time step settings...
 - Types of objects
-

© *ppgb*, March 16, 2003

Spectrum

One of the types of objects in PRAAT. A Spectrum object represents the complex spectrum as a function of frequency. If the spectrum was created from a sound (which is expressed in units of Pascal), the complex values are expressed in units Pa/Hz (Pascal per Hertz).

Spectrum commands

Creation:

- Sound: To Spectrum (fft)

Queries:

- Spectrum: Get centre of gravity...
- Spectrum: Get standard deviation...
- Spectrum: Get skewness...
- Spectrum: Get kurtosis...
- Spectrum: Get central moment...

Modification:

- Spectrum: Filter (pass Hann band)...
- Spectrum: Filter (stop Hann band)...
- Formula...

Conversion:

- Spectrum: To Ltas (1-to-1)
- Spectrum: To Spectrogram

Synthesis:

- Spectrum: To Sound (fft)

Inside the Spectrum object

With Inspect, you can see that a Spectrum object has the following attributes:

x_{min}
bottom of frequency domain, in Hertz. Usually 0.

x_{max}
top of frequency domain, in Hertz.

- n_x
the number of frequencies (≥ 1).
- dx
frequency step, in Hertz.
- x_1
the frequency associated with the first bin, in Hertz. Often 0. The frequency associated with the last bin (i.e., $x_1 + (n_x - 1) dx$) will often be equal to x_{max} .
- n_y
the number of ‘rows’ (if you regard a Spectrum as a sort of Matrix). Always 2: the real part and the imaginary part.
- $z_{1i}, i = 1 \dots n_x$
the real part of the spectrum, often in Pa/Hz.
- $z_{2i}, i = 1 \dots n_x$
the imaginary part of the spectrum, often in Pa/Hz.

Links to this page

- [Band filtering in the frequency domain](#)
- [Formulas 1.8. Formulas for modification](#)
- [Formulas 7. Attributes of objects](#)
- [Intro 3.5. Viewing a spectral slice](#)
- [Intro 3.7. The Spectrum object](#)
- [LPC: To Spectrum \(slice\)...](#)
- [Polynomial: To Spectrum...](#)
- [Sound: Deepen band modulation...](#)
- [Sound: To Spectrum \(dft\)](#)
- [Spectra: Multiply](#)
- [Spectrogram: To Spectrum \(slice\)...](#)
- [Spectrum: Conjugate](#)
- [Spectrum: Formula...](#)
- [SpectrumEditor](#)

© *ppgb*, March 16, 2003

SpectrumEditor

One of the editors in PRAAT. It allows you to view, zoom, and play a Spectrum object.

Clicking on one of the (maximally) 8 rectangles above or below the drawing area lets you play a Sound that is synthesized from a band-filtered part of the Spectrum. You can also copy the Spectrum, band-filtered with the frequency selection, or the Sound synthesized from this Spectrum, to the list of objects.

Links to this page

- [Intro 3.5. Viewing a spectral slice](#)
- [Types of objects](#)
- [What's new?](#)

© *ppgb*, March 16, 2003

Time selection

The ways to select a part of the time (or frequency) domain in some Editors in the **Praat program**, namely those that contain a function of time (or frequency).

The **time selection** is used for selecting the time (or frequency) interval that will be played, copied, cut, modified, or questioned:

1. How to make a selection

The easiest way is to drag the mouse across the part that you want to select. This is analogous to how selection works in a text processor.

Also see point 3 below.

2. How to extend or shrink a selection

The easiest way is to click with the Shift key pressed. The nearest edge of the selection will move to the time position where you clicked. This is analogous to how extending a selection works in a text processor.

For instance, if the currently selected time interval runs from 2 to 5 seconds, and you shift-click at a time position of 4 seconds, the end of the selection will move from 5 to 4 seconds, thus shrinking the selection.

You can also shift-drag, i.e. hold the Shift key and the mouse button down while moving the mouse.

Also see point 3 below.

3. How to move beginning and end separately

Clicking with the *middle* mouse button (on Mac: clicking while keeping the Option key pressed) inside the drawing area will move the "B" hair to the time where you click. "B" stands for *Begin selection*.

Clicking with the *right* mouse button (on Mac: clicking while keeping the Command key pressed) will move the "E" hair. "E" stands for *End selection*.

If you use a left-hand mouse (on SGI: *Desktop - Customize - Mouse*), the left button will be "B" and the middle button will be "E".

To move one of the marks to the other mark, or to an absolute or relative position, use the Select menu.

Links to this page

- [Frequency selection](#)
- [Intro 3.5. Viewing a spectral slice](#)
- [ManipulationEditor](#)

- PitchEditor
 - PitchTierEditor
 - PointEditor
 - SoundEditor
 - TextGridEditor
-

© *ppgb*, February 16, 2003

Sound: To Spectrum (fft)

A command to create a Spectrum object from every selected Sound object, by an over-all spectral analysis.

Algorithm

The algorithm is the continuous interpretation of the Fast Fourier Transform, with a negative exponent:

$$X(f) = \int_0^T x(t) e^{-2\pi ift} dt$$

where T is the duration of the sound, and t is taken relative to the time associated with the first sample.

If the Sound is expressed in Pascal (Pa), the Spectrum is expressed in Pa·s, or Pa/Hz. The frequency integral over the Spectrum equals the time integral over the Sound.

Behaviour

If you perform Spectrum: To Sound (fft) on the resulting Spectrum object, a Sound is created that is equal to the original Sound.

Links to this page

- [Intro 3.6. Configuring the spectral slice](#)
- [Intro 3.7. The Spectrum object](#)
- [Sound: Filter \(formula\)...](#)
- [Sound: Filter \(pass Hann band\)...](#)
- [Sound: Filter \(stop Hann band\)...](#)
- [Spectrum: Get centre of gravity...](#)

© ppgb, September 10, 2003

Extract visible pitch contour

One of the commands in the Pitch menu of the SoundEditor and the TextGridEditor.

See Intro 4. Pitch analysis

Links to this page

- [Intro 4.4. The Pitch object](#)

© *ppgb*, March 16, 2003

Pitch

One of the types of objects in PRAAT. For tutorial information, see Intro 4. Pitch analysis.

A Pitch object represents periodicity candidates as a function of time. It does not mind whether this periodicity refers to acoustics, perception, or vocal-cord vibration. It is sampled into a number of *frames* centred around equally spaced times.

Pitch commands

Creation:

- Sound: To Pitch...: preferred method (autocorrelation).
- Sound: To Pitch (ac)...: autocorrelation method (all parameters).
- Sound: To Pitch (cc)...: cross-correlation method.

Drawing:

- Pitch: Draw...

Viewing and editing:

- PitchEditor

Synthesis:

- Pitch: To PointProcess: create points in voiced intervals.
- Sound & Pitch: To PointProcess (cc): near locations of high amplitude.
- Sound & Pitch: To PointProcess (peaks)...: near locations of high amplitude.

Conversion:

- Pitch: To PitchTier: time-stamp voiced intervals.
- Pitch & TextTier: To PitchTier...: interpolate values at specified times.

Inside a Pitch object

With Inspect, you will see the following attributes:

x_{min}

starting time, in seconds.

x_{max}

end time, in seconds.

n_x

the number of frames (≥ 1).

dx

time step = frame length = frame duration, in seconds.

x₁

the time associated with the first frame, in seconds. This will usually be in the range [*xmin*, *xmax*].
The time associated with the last frame (i.e., $x_1 + (n_x - 1) dx$) will also usually be in that range.

ceiling

a frequency above which a candidate is considered voiceless.

frame_i, $i = 1 \dots n_x$

the frames (see below).

Attributes of a pitch frame

Each frame contains the following attributes:

nCandidates

the number of candidates in this frame (at least one: the ‘unvoiced’ candidate).

candidate_j, $j = 1 \dots nCandidates$

the information about each candidate (see below).

Attributes of each candidate

Each candidate contains the following attributes:

frequency

the candidate’s frequency in Hz (for a voiced candidate), or 0 (for an unvoiced candidate).

strength

the degree of periodicity of this candidate (between 0 and 1).

Interpretation

The current pitch contour is determined by the path through all first candidates. If the first candidate of a certain frame has a frequency of 0, or a frequency above *ceiling*, this frame is considered voiceless.

Links to this page

- [FAQ: Pitch analysis](#)
- [Formulas 7. Attributes of objects](#)
- [Get frame number from time...](#)
- [Get number of frames](#)
- [Get time from frame number...](#)
- [Get time step](#)
- [Intro 4.4. The Pitch object](#)
- [Pitch: Interpolate](#)
- [Pitch: Smooth...](#)
- [PitchTier: Get mean \(curve\)...](#)

- PitchTier: Get standard deviation (points)...
 - Play
 - Sound & Pitch: Change gender...
 - Sound & Pitch: To FormantFilter...
 - Sound: To Pitch (shs)...
 - time domain
 - What's new?
-

© *ppgb*, March 16, 2003

Show pitch

One of the commands in the Pitch menu of the SoundEditor and the TextGridEditor.

See Intro 4. Pitch analysis.

Links to this page

- [Intro 4.1. Viewing a pitch contour](#)

© *ppgb*, March 16, 2003

Pitch settings...

A command in the Pitch menu of the SoundEditor and TextGridEditor windows. See Intro 4.2. Configuring the pitch contour.

Links to this page

- [Advanced pitch settings...](#)
 - [Voice 5. Comparison with other programs](#)
-

© *ppgb*, March 16, 2003

Advanced pitch settings...

A command in the **Pitch** menu of the SoundEditor or TextGridEditor windows. Before changing the advanced pitch settings, make sure you understand Intro 4.2. Configuring the pitch contour.

View range different from analysis range

Normally, the range of pitch values that can be seen in the editor window is equal to the range of pitch values that the analysis algorithm can determine. If you set the analysis range from 75 to 500 Hz, this will be the range you see in the editor window as well. If the pitch values in the curve happen to be between 350 and 400 Hz, you may want to zoom in to the 350-400 Hz pitch region. You will usually do this by changing the pitch range in the Pitch settings... dialog. However, the analysis range will also change in that case, so that the curve itself may change. If you do not want that, you can change the *View range* settings from "0.0 (= auto)" - "0.0 (=auto)" to something else, perhaps "350" - "400".

Pitch analysis settings

For information about these, see Sound: To Pitch (ac).... The standard settings are best in most cases. For some pathological voices, you will want to set the voicing threshold to much less than the standard of 0.45, in order to get pitch values even in irregular parts of the signal.

© ppgb, September 16, 2003

Get pitch

One of the commands in the Query menu of the SoundEditor and the TextGridEditor.

Links to this page

- [Intro 4.3. Querying the pitch contour](#)
-

© *ppgb*, April 17, 2001

Info window

A text window into which many query commands write their answers.

You can select text from this window and copy it to other places.

In a Praat script, you can bypass the Info window by having a query command writing directly into a script variable.

Apart from the Info command, which writes general information about the selected object, the following more specific commands also write into the Info window:

- Categories: Difference
- Intro 4.3. Querying the pitch contour
- Intro 5.3. Querying the formant contours
- Log files
- Query
- Scripting 6.3. Writing to the Info window
- TextGrid: Count labels...
- undefined

© *ppgb*, May 28, 2003

Sound: To Pitch...

A command that creates a Pitch object from every selected Sound object.

Purpose

to perform a pitch analysis, optimized for speech.

Arguments

The arguments that control the recruitment of the candidates are:

Time step (standard value: 0.0)

the measurement interval (frame duration), in seconds. If you supply 0, PRAAT will use a time step of $0.75 / (\textit{pitch floor})$, e.g. 0.01 seconds if the pitch floor is 75 Hz; in this example, PRAAT computes 100 pitch values per second.

Pitch floor (standard value: 75 Hz)

candidates below this frequency will not be recruited. This parameter determines the length of the analysis window: it will be 3 longest periods long, i.e., if the pitch floor is 75 Hz, the window will be $3/75 = 0.04$ seconds long.

Note that if you set the time step to zero, the analysis windows for consecutive measurements will overlap appreciably: PRAAT will always compute 4 pitch values within one window length, i.e., the degree of *oversampling* is 4.

A post-processing algorithm seeks the cheapest path through the candidates. The argument that determines the cheapest path is:

Pitch ceiling (standard value: 600 Hz)

candidates above this frequency will be ignored.

Algorithm

This is the algorithm described at [Sound: To Pitch \(ac\)...](#), with all the parameters not mentioned above set to their standard values.

Links to this page

- [FAQ: Pitch analysis](#)
- [Intro 4.4. The Pitch object](#)
- [Manipulation](#)
- [Periodicity submenu](#)
- [Script for listing F0 statistics](#)
- [Script for listing time\F0\intensity](#)
- [Sound: To FormantFilter...](#)

- Sound: To Pitch (cc)...
 - Time step settings...
-

© *ppgb*, September 16, 2003

Periodicity submenu

A submenu that occurs in the Dynamic menu for a Sound.

This submenu contains commands for analysing the pitch contour of the selected Sound:

- Sound: To Pitch...
- Sound: To Pitch (ac)...
- Sound: To Pitch (cc)...
- Sound: To Harmonicity (cc)...
- Sound: To Harmonicity (ac)...

Links to this page

- [Intro 4.4. The Pitch object](#)
-

© *ppgb*, April 17, 2001

PitchEditor

One of the Editors in PRAAT, for viewing and modifying a Pitch object.

What the Pitch editor shows

In the window of the PitchEditor, you will see the following features:

- Digits between 0 and 9 scattered all over the drawing area. Their locations represent the pitch *candidates*, of which there are several for every time frame. The digits themselves represent the goodness of a candidate, multiplied by ten. For instance, if you see a "9" at the location (1.23 seconds, 189 Hertz), this means that in the time frame at 1.23 seconds, there is a pitch candidate with a value of 189 Hertz, and its goodness is 0.9. The number 0.9 may be the relative height of an autocorrelation peak, a cross-correlation peak, or a spectral peak, depending on the method by which the Pitch object was computed.
- A *path* of red disks. These disks represent the best path through the candidates, i.e. our best guess at what the pitch contour is. The path will usually have been determined by the *path finder*, which was called by the pitch-extraction algorithm, and you can change the path manually. The path finder takes into account the goodness of each candidate, the intensity of the sound in the frame, voiced-unvoiced transitions, and frequency jumps. It also determines whether each frame is voiced or unvoiced.
- A *voicelessness bar* at the bottom of the drawing area. If there is no suitable pitch candidate in a frame, the frame is considered voiceless, which is shown as a blue rectangle in the voicelessness bar.
- A line of digits between 0 and 9 along the top. These represent the relative intensity of the sound in each frame.

Moving the marks

To move the cursor hair or the beginning or end of the selection, use the time selection mechanism.

Changing the path

To change the path through the candidates manually, click on the candidates of your choice. The changes will immediately affect the Pitch object that you are editing. To make a voiced frame voiceless, click on the voicelessness bar.

To change the path automatically, choose 'Path finder...' from the 'Edit' menu; this will have the same effect as filling in different values in the Sound: To Pitch (ac)... dialog, but is much faster because the candidates do not have to be determined again.

Resynthesis

To hum any part of the pitch contour, click on one of the buttons below or above the data area (there can be 1 to 8 of these buttons), or use a Play command from the View menu.

Changing the ceiling

To change the ceiling, but not the path, choose ‘Change ceiling...’ from the ‘Edit’ menu; if the new ceiling is lower than the old ceiling, some formerly voiced frames may become unvoiced; if the new ceiling is higher than the old ceiling, some formerly unvoiced frames may become voiced.

Links to this page

- [Intro 4.4. The Pitch object](#)
 - [Types of objects](#)
-

© *ppgb*, March 16, 2003

Read menu

One of the menus in the Object window.

With the Read menu, you read one or more objects from a file on disk into memory. The resulting object(s) will appear in the List of Objects.

The Read menu contains the command Read from file..., which recognizes most file types, and perhaps several other commands for reading unrecognizable file types (e.g., raw sound data), or for interpreting known file types in a different way (e.g., reading two sounds from a stereo sound file, which is normally read as mono):

- How to concatenate sound files
- Intro 4.4. The Pitch object
- Intro 5.4. The Formant object
- LongSound
- New menu
- Open long sound file...
- Sound files 3. Files that Praat can read
- Write menu

© ppgb, September 11, 1997

Draw submenu

A submenu that occurs in the Dynamic menu for many objects.

This submenu contains commands for drawing the object to the Picture window, which will allow you to print the drawing or to copy it to your word processor.

Links to this page

- [Intro 4.4. The Pitch object](#)
- [Intro 5.4. The Formant object](#)

© *ppgb*, April 17, 2001

Extract visible formant contour

One of the commands in the Formant menu of the SoundEditor and the TextGridEditor.

See Intro 5. Formant analysis

Links to this page

- [Intro 5.4. The Formant object](#)

© *ppgb*, March 16, 2003

Show formant

One of the commands in the Formant menu of the SoundEditor and the TextGridEditor.

See Intro 5. Formant analysis.

Links to this page

- [Intro 5.1. Viewing formant contours](#)

© *ppgb*, March 16, 2003

Sound: To Formant (burg)...

A command that creates a Formant object from every selected Sound object. It performs a short-term spectral analysis, approximating the spectrum of each analysis frame by a number of formants.

Arguments

Time step (seconds)

the time between the centres of consecutive analysis frames. If the sound is 2 seconds long, and the time step is 0.01 seconds, there will be approximately 200 analysis frames. The actual number is somewhat lower (usually 195), because we cannot measure very well near the edges. If you set the time step to 0.0 (the standard), Praat will use a time step that is equal to 25 percent of the analysis window length (see below).

Maximum number of formants

for most analyses of human speech, you will want to extract 5 formants per frame. This, in combination with the *Maximum formant* argument, is the only way in which this procedure will give you results compatible with how people tend to interpret formants for vowels, i.e. in terms of vowel height (F1) and vowel place (F2).

Maximum formant (Hertz)

the ceiling of the formant search range. It is crucial that you set this argument to a value suitable for your speaker. The standard value of 5500 Hz is suitable for an average adult female. For a male, use 5000 Hz; if you use 5500 Hz for an adult male, you may end up with too few formants in the low frequency region, e.g. analysing an [u] as having a single formant near 500 Hz whereas you want two formants at 300 and 600 Hz. For a young child, use a value much higher than 5500 Hz, for instance 8000 Hz (experiment with it on steady vowels).

Window length (seconds)

the effective duration of the analysis window. The actual length is twice this value, because Praat uses a Gaussian-like analysis window with sidelobes below -120 dB. For instance, if the *Window length* is 0.025 seconds, the actual Gaussian window duration is 0.050 seconds. This window has values below 4% outside the central 0.025 seconds, and its frequency resolution (-3 dB point) is $1.298 / (0.025 \text{ s}) = 51.9 \text{ Hz}$, as computed with the formula given at Sound: To Spectrogram.... This is comparable to the bandwidth of a Hamming window of 0.025 seconds, which is $1.303 / (0.025 \text{ s}) = 52.1 \text{ Hz}$, but that window (which is the window most often used in other analysis programs) has three spectral lobes of about -42 dB on each side.

Pre-emphasis from (Hertz)

the +3 dB point for an inverted low-pass filter with a slope of +6 dB/octave. If this value is 50 Hz, then frequencies below 50 Hz are not enhanced, frequencies around 100 Hz are amplified by 6 dB, frequencies around 200 Hz are amplified by 12 dB, and so forth. The point of this is that vowel spectra tend to fall by 6 dB per octave; the pre-emphasis creates a flatter spectrum, which is better for formant analysis because we want our formants to match the local peaks, not the global spectral slope. See the source-filter synthesis tutorial for a technical explanation, and Sound: Pre-emphasize (in-line)... for the algorithm.

Algorithm

The sound will be resampled to a sampling frequency of twice the value of *Maximum formant*, with the algorithm described at [Sound: Resample....](#) After this, pre-emphasis is applied with the algorithm described at [Sound: Pre-emphasize \(in-line\)....](#) For each analysis window, Praat applies a Gaussian-like window, and computes the LPC coefficients with the algorithm by Burg, as given by Childers (1978) and Press et al. (1992).

This algorithm can initially find formants at very low or high frequencies. In order for you to be able to identify the traditional F1 and F2, all formants below 50 Hz and all formants above *Maximum formant* minus 50 Hz, are removed. If you don't want this, you may experiment with [Sound: To Formant \(keep all\)...](#) instead. If you prefer an algorithm that always yields the requested number of formants, nicely distributed across the frequency domain, you may try the otherwise rather unreliable Split-Levinson procedure [Sound: To Formant \(sl\)...](#)

Links to this page

- [FAQ: Formant analysis](#)
- [Formant analysis...](#)
- [Formant: Track...](#)
- [Formants & LPC submenu](#)
- [Intro 5.2. Configuring the formant contours](#)
- [Intro 5.4. The Formant object](#)
- [Sound: To LPC \(autocorrelation\)...](#)
- [Sound: To LPC \(burg\)...](#)
- [Sound: To LPC \(covariance\)...](#)
- [Sound: To LPC \(marple\)...](#)
- [Time step settings...](#)

© ppgb, October 3, 2003

Get first formant

One of the commands in the Query menu of the SoundEditor and the TextGridEditor.

Links to this page

- [Intro 5.3. Querying the formant contours](#)
-

© *ppgb*, November 7, 2001

Get second formant

One of the commands in the Query menu of the SoundEditor and the TextGridEditor.

Links to this page

- [Intro 5.3. Querying the formant contours](#)
-

© *ppgb*, November 7, 2001

Formant

One of the types of objects in PRAAT. A Formant object represents spectral structure as a function of time: a *formant contour*. Unlike the time-stamped FormantTier object, it is sampled into a number of *frames* centred around equally spaced times, Each frame contains frequency and bandwidth information about several formants.

Inside a Formant object

With Inspect, you will see the following attributes:

xmin

starting time, in seconds.

xmax

end time, in seconds.

nx

the number of frames (≥ 1).

dx

time step = frame length = frame duration, in seconds.

x1

the time associated with the first frame, in seconds. This will usually be in the range [*xmin*, *xmax*].
The time associated with the last frame (i.e., $x1 + (nx - 1) dx$) will also usually be in that range.

frame_i, $i = 1 \dots nx$

the frames (see below).

Attributes of a formant frame

Each *frame_i* contains the following attributes:

intensity

an indication of the maximum intensity (a squared sound amplitude) in this frame.

nFormants

the number of formants in this frame (usually between 2 and 6).

formant_j, $j = 1 \dots nFormants$

the information about each formant (see below).

Attributes of each formant

Each *formant_j* contains the following attributes:

frequency

the formant's centre frequency (in Hz).

bandwidth

the formant's bandwidth (in Hz).

See also

Linear Prediction

Links to this page

- [Formant: Down to FormantTier](#)
- [Formant: Draw tracks...](#)
- [Formant: Formula \(bandwidths\)...](#)
- [Formant: Formula \(frequencies\)...](#)
- [Formant: Get bandwidth at time...](#)
- [Formant: Get maximum...](#)
- [Formant: Get mean...](#)
- [Formant: Get minimum...](#)
- [Formant: Get number of formants](#)
- [Formant: Get quantile...](#)
- [Formant: Get standard deviation](#)
- [Formant: Get time of maximum...](#)
- [Formant: Get time of minimum...](#)
- [Formant: Get value at time...](#)
- [Formant: Speckle...](#)
- [Formant: Track...](#)
- [Formulas 7. Attributes of objects](#)
- [Get frame number from time...](#)
- [Get number of frames](#)
- [Get time from frame number...](#)
- [Get time step](#)
- [Intro 5.4. The Formant object](#)
- [Sound & Formant: Filter](#)
- [Sound & Formant: Filter \(no scale\)](#)
- [Sound: To Formant \(burg\)...](#)
- [Sound: To Formant \(keep all\)...](#)
- [Sound: To Formant \(sl\)...](#)
- [Source-filter synthesis](#)
- [time domain](#)
- [What's new?](#)

Formants & LPC submenu

A submenu that occurs in the Dynamic menu for a Sound.

This submenu contains commands for analysing the formant contours of the selected Sound:

- Sound: To Formant (burg)...
- Sound: To Formant (keep all)...
- Sound: To Formant (sl)...
- Sound: To LPC (autocorrelation)...
- Sound: To LPC (covariance)...
- Sound: To LPC (burg)...
- Sound: To LPC (marple)...
- Sound: To MFCC...

Links to this page

- [Intro 5.4. The Formant object](#)
-

© *ppgb*, November 7, 2001

Extract visible intensity contour

One of the commands in the Intensity menu of the SoundEditor and the TextGridEditor.

See Intro 6. Intensity analysis

© *ppgb*, March 16, 2003

Show intensity

One of the commands in the Intensity menu of the SoundEditor and the TextGridEditor.

See Intro 6. Intensity analysis.

© *ppgb*, March 16, 2003

Intensity

One of the types of objects in PRAAT.

An Intensity object represents an intensity contour at linearly spaced time points $t_i = t_1 + (i - 1) dt$, with values in dB SPL, i.e. dB relative to $2 \cdot 10^{-5}$ Pascal, which is the normative auditory threshold for a 1000-Hz sine wave.

Links to this page

- [Formulas 7. Attributes of objects](#)
- [Get frame number from time...](#)
- [Get number of frames](#)
- [Get time from frame number...](#)
- [Get time step](#)
- [Intensity & TextTier: To IntensityTier...](#)
- [Intensity: Get maximum...](#)
- [Intensity: Get mean...](#)
- [Intensity: Get minimum...](#)
- [Intensity: Get standard deviation...](#)
- [Intensity: Get time of maximum...](#)
- [Intensity: Get time of minimum...](#)
- [Intensity: Get value at time...](#)
- [Intensity: Get value in frame...](#)
- [Intensity: To IntensityTier](#)
- [Intro 6.4. The Intensity object](#)
- [Sound & IntensityTier: Multiply](#)
- [Sound: To Intensity...](#)
- [Source-filter synthesis](#)
- [time domain](#)
- [What's new?](#)

© *ppgb*, March 16, 2003

Sound: To Intensity...

A command to create an Intensity object from every selected Sound.

Arguments

Minimum pitch (Hz)

specifies the minimum periodicity frequency in your signal. If you set *Minimum pitch* too high, you will end up with a pitch-synchronous intensity modulation. If you set it too low, your intensity contour may appear smeared, so you should set it as high as allowed by the signal if you want a sharp contour.

Time step (s)

the time step of the resulting intensity contour. If you set it to zero, the time step is computed as one quarter of the effective window length, i.e. as $0.8 / (\textit{Minimum pitch})$

Algorithm

The values in the sound are squared and convolved with a Kaiser-20 window (sidelobes below -190 dB). The effective length of this window is $3.2 / (\textit{Minimum pitch})$, which will guarantee that a periodic signal is analysed with a pitch-synchronous intensity ripple not greater than our 4-byte floating-point precision (i.e., < 0.00001 dB).

Links to this page

- [Intro 6.4. The Intensity object](#)
- [Script for listing time\--F0\--intensity](#)
- [Time step settings...](#)

© ppgb, August 22, 2003

TextGrid

One of the types of objects in PRAAT, used for *annotation* (segmentation and labelling). For tutorial information, see Intro 7. Annotation.

Description

A **TextGrid** object consists of a number of *tiers*. There are two kinds of tiers: an *interval tier* is a connected sequence of labelled intervals, with *boundaries* in between. A *point tier* is a sequence of labelled points.

How to create a TextGrid

From scratch:

- Sound: To TextGrid... (takes the time domain from the Sound)
- LongSound: To TextGrid... (takes the time domain from the LongSound)
- PointProcess: To TextGrid... (takes the time domain from the PointProcess)
- PointProcess: To TextGrid (vuv)... (labels voiced and unvoiced intervals)
- Create TextGrid...

From merging any number of existing tiers:

- TextTier(s): To TextGrid
- IntervalTier(s): To TextGrid
- TextTier(s) & IntervalTier(s): To TextGrid

From merging an existing TextGrid with existing tiers or TextGrids:

- TextGrid & TextTier: Append
- TextGrid & IntervalTier: Append
- TextGrid & TextGrid: Merge

Conversion from an old-style Label object:

- Label & Sound: To TextGrid

How to edit a TextGrid

You select a TextGrid alone or together with a Sound or LongSound, and click **Edit**. A TextGridEditor will appear on your screen, containing the TextGrid and an optional copy of the Sound or LongSound.

How to draw a TextGrid

You can draw a TextGrid to the Picture window with:

- TextGrid: Draw...**
- TextGrid & Sound: Draw...**
- TextGrid & Pitch: Draw...**
- TextGrid & Pitch: Draw separately...**

Links to this page

- [IntervalTier](#)
- [SpellingChecker](#)
- [TextGrid: Count labels...](#)
- [TextGrid: Extend time...](#)
- [TextTier](#)
- [time domain](#)

© *ppgb*, April 13, 2004

Sound: To TextGrid...

A command to create a TextGrid without any labels, copying the time domain from the selected Sound.

Arguments

Tier names

a list of the names of the tiers that you want to create, separated by spaces.

Point tiers

a list of the names of the tiers that you want to be *point tiers*; the rest of the tiers will be *interval tiers*.

Example

If *Tier names* is "a b c", and "Point tiers" is "b", the resulting TextGrid object will contain an interval tier named "a", a point tier named "b", and another interval tier named "c".

Links to this page

- [Intro 7. Annotation](#)
- [LongSound: To TextGrid...](#)

© ppgb, July 30, 1998

LongSound: To TextGrid...

A command to create a TextGrid without any labels, copying the time domain from the selected LongSound.

See Sound: To TextGrid... for the arguments.

Links to this page

- [Intro 7. Annotation](#)

© *ppgb*, July 30, 1998

Create TextGrid...

A command to create a TextGrid from scratch.

Arguments

From time (seconds)

the starting time, usually 0 seconds.

To time (seconds)

the end time, usually the duration.

Tier names

a list of the names of the tiers that you want to create, separated by spaces.

Point tiers

a list of the names of the tiers that you want to be *point tiers*; the rest of the tiers will be *interval tiers*.

Links to this page

- [Intro 7. Annotation](#)
- [New menu](#)

© ppgb, March 16, 1997

Write to text file...

One of the commands in the Write menu.

Availability

You can choose this command after selecting one or more objects.

Behaviour

The Object window will ask you for a file name. After you click OK, the objects will be written to a text file on disk.

File format

The format is the same as used by Write to console, except for the first two lines:

- if you selected a single object, e.g., of class Pitch, the file will start with the lines:

```
File type = "ooTextFile"
Class = "Pitch"
```

- if you selected more than one object, e.g., 'Pitch hallo' and 'Polygon kromme', the file will look like:

```
File type = "ooTextFile"
Class = "Collection"
size = 2
item []:
  item [1]:
    class = "Pitch"
    name = "hallo"
    (pitch data...)
  item [2]:
    class = "Polygon"
    name = "kromme"
    (polygon data...)
```

The file can be read again with Read from file..., which, by the way, does not need the verbosity of the above example. The following minimal format will also be read correctly:

```
"ooTextFile"
"Collection" 2
"Pitch" "hallo" (pitch data...)
"Polygon" "kromme" (polygon data...)
```

Thus, all text that is not a free-standing number and is not enclosed in double quotes or <>, is considered a comment, as is all text following an exclamation mark (!) on the same line.

Links to this page

- ExperimentMFC
- Intro 7. Annotation
- OT learning 2.2. Inside the grammar
- OT learning 2.3. Defining your own grammar
- Sound
- TableOfReal
- Write to short text file...

© *ppgb*, September 11, 1997

Write to short text file...

One of the commands in the Write menu.

Availability

You can choose this command after selecting one or more objects.

Behaviour

The Object window will ask you for a file name. After you click OK, the objects will be written to a text file on disk.

File format

The format is much shorter than the one described at Write to text file.... Most of the comments are gone, and there is normally one piece of data per line.

The file can be read again with the all-purpose Read from file....

Links to this page

- [Intro 7. Annotation](#)

© *ppgb*, November 24, 1998

Write to binary file...

One of the commands in the Write menu.

Availability

You can choose this command after selecting one or more objects.

Behaviour

The Object window will ask you for a file name. After you click OK, the objects will be written to a binary file on disk.

Usage

The file can be read again with Read from file....

File format

These files are in a device-independent binary format, and can be written and read on any machine.

Links to this page

- [Intro 7. Annotation](#)
- [Sound](#)
- [WordList](#)

© *ppgb*, September 11, 1997

Labelling

See Intro 7. Annotation.

© *ppgb*, April 8, 2001

Segmentation

See Intro 7. Annotation.

© *ppgb*, April 8, 2001

Manipulation

One of the types of objects in PRAAT, for changing the pitch and duration contours of a sound.

Inside a manipulation object

With Inspect, you will see the following attributes:

timeStep

the time step (or *frame length*) used in the pitch analysis. A common value is 0.010 seconds.

minimumPitch

the minimum pitch frequency considered in the pitch analysis. A common value is 75 Hertz.

maximumPitch

the maximum pitch frequency considered in the pitch analysis. A common value is 600 Hertz.

A Manipulation object also contains the following smaller objects:

1. The original Sound.
2. A PointProcess representing glottal pulses.
3. A PitchTier.
4. A DurationTier.

Analysis

When a Manipulation object is created from a sound, the following steps are performed:

1. A pitch analysis is performed on the original sound, with the method of Sound: To Pitch.... This uses the time step, minimum pitch, and maximum pitch parameters.
2. The information of the resulting pitch contour (frequency and voiced/unvoiced decisions) is used to posit glottal pulses where the original sound contains much energy. The method is the same as in Sound & Pitch: To PointProcess (cc).
3. The pitch contour is converted to a pitch tier with many points (targets), with the method of Pitch: To PitchTier.
4. An empty DurationTier is created.

Resynthesis

A Manipulation object can produce Sound input. This Sound can be computed in several ways:

- o PSOLA: from original sound + pulses + pitch tier + duration tier;
- o LPC: from LPC (from original sound) + pulses + pitch tier;
- o from the pulses only, as a pulse train or hummed;
- o from the pitch tier only, as a pulse train or hummed.

Links to this page

- [Intro 8.1. Manipulation of pitch](#)
- [Intro 8.2. Manipulation of duration](#)
- [Manipulation: Extract duration tier](#)
- [Manipulation: Extract original sound](#)
- [Manipulation: Extract pitch tier](#)
- [Manipulation: Extract pulses](#)
- [Manipulation: Get resynthesis \(PSOLA\)](#)
- [Manipulation: Play \(PSOLA\)](#)
- [Manipulation: Replace duration tier](#)
- [Manipulation: Replace original sound](#)
- [Manipulation: Replace pitch tier](#)
- [Manipulation: Replace pulses](#)
- [ManipulationEditor](#)
- [What's new?](#)

© *ppgb*, March 16, 2003

ManipulationEditor

One of the Editors in PRAAT, for viewing and manipulating a Manipulation object.

Objects

The editor shows:

- The original Sound.
- The PointProcess that represents the glottal *pulses*. You can edit it for improving the pitch analysis.
- A pitch contour based on the locations of the pulses, for comparison (drawn as grey dots). Changes shape if you edit the pulses.
- The PitchTier that determines the pitch contour of the resynthesized Sound (drawn as blue circles). At the creation of the Manipulation object, it is computed from the original pitch contour. You can manipulate it by simplifying it (i.e., removing targets), or by moving parts of it up and down, and back and forth.
- A DurationTier for manipulating the relative durations of the voiced parts of the sound.

Playing

To play (a part of) the *resynthesized* sound (by any of the methods shown in the **Synth** menu, like PSOLA and **LPC**), click on any of the 1 to 8 buttons below and above the drawing area or use the Play commands from the View menu.

To play the *original* sound instead, use **Shift-click**.

Pulses

To add:

click at the desired time location, and choose **Add pulse at cursor** or type **Command-p**.

To remove:

make a time selection, and choose **Remove pulse(s)** or type **Option-Command-p**. If there is no selection, the pulse nearest to the cursor is removed.

Pitch points

To add one at a specified *time and frequency*:

click at the desired time-frequency location, and choose **Add pitch point at cursor** or type **Command-t**.

To add one at a specified *time* only:

click at the desired time, and choose **Add pitch point at time slice**. ManipulationEditor tries to compute the frequency from the intervals between the pulses, basically by a median-of-three method.

To remove:

make a time selection, and choose **Remove pitch point(s)** or type **Option-Command-t**. If there is no selection, the pitch point nearest to the cursor is removed.

To move *some*:

make a time selection (the points become red) and **Shift-drag** the points across the window. You cannot drag them across adjacent points, or below 50 Hz, or above the maximum frequency. You can only drag them horizontally if the *dragging strategy* is **All** or **Only horizontal**, and you can drag them vertically if the dragging strategy is not **Only horizontal**. You can change the dragging strategy with **Set pitch dragging strategy...** from the **Pitch** menu.

To move *one*:

drag that point across the window. You can only drag it horizontally if the dragging strategy is not **Only vertical**, and you can drag it vertically if the dragging strategy is not **Only horizontal**.

Duration points

Work pretty much the same as pitch points.

Stylization

Before editing the Pitch points, you may want to reduce their number by choosing any of the **Stylize** commands from the **Pitch** menu.

Links to this page

- [Intro 8.1. Manipulation of pitch](#)
- [Intro 8.2. Manipulation of duration](#)
- [Intro 8.3. Manipulation of intensity](#)
- [Source-filter synthesis](#)
- [Types of objects](#)
- [What's new?](#)

© ppgb, March 16, 2003

PitchTier

One of the types of objects in PRAAT. A PitchTier object represents a time-stamped pitch contour, i.e. it contains a number of (*time*, *pitch*) points, without voiced/unvoiced information. For instance, if your PitchTier contains two points, namely 150 Hz at a time of 0.5 seconds and 200 Hz at a time of 1.5 seconds, then this is to be interpreted as a pitch contour that is constant at 150 Hz for all times before 0.5 seconds, constant at 200 Hz for all times after 1.5 seconds, and linearly interpolated for all times between 0.5 and 1.5 seconds (i.e. 170 Hz at 0.7 seconds, 210 Hz at 1.1 seconds, and so on).

PitchTier commands

Creation:

From scratch:

- o Create PitchTier...
- o PitchTier: Add point...

Copy from another object:

- o Pitch: To PitchTier: trivial copying of voiced frames.
- o PointProcess: Up to PitchTier...: single value at specified times.
- o Pitch & TextTier: To PitchTier...: copying interpolated values at specified points.

Synthesize from another object:

- o PointProcess: To PitchTier...: periodicity analysis.

Extract from a Manipulation object:

- o Manipulation: Extract pitch tier

Viewing and editing:

- PitchTierEditor: with or without a Sound.
- ManipulationEditor

Conversion:

- PitchTier: Down to PointProcess: copy times.

Synthesis:

- PitchTier: To PointProcess: area-1 pulse generation (used in PSOLA).
- Manipulation: Replace pitch tier

Queries:

- Get low index from time...
- Get high index from time...
- Get nearest index from time...

Modification:

- [Remove point...](#)
- [Remove point near...](#)
- [Remove points between...](#)
- [PitchTier: Add point...](#)

Links to this page

- [Get area...](#)
- [Intro 8.1. Manipulation of pitch](#)
- [Pitch: To PointProcess](#)
- [PitchTier: Get mean \(curve\)...](#)
- [PitchTier: Get mean \(points\)...](#)
- [PitchTier: Get standard deviation \(curve\)...](#)
- [PitchTier: Get standard deviation \(points\)...](#)
- [Source-filter synthesis](#)
- [time domain](#)
- [What's new?](#)

© *ppgb*, March 16, 2003

Create PitchTier...

A command in the New menu to create an empty PitchTier object.

The resulting object will have the specified name and time domain, but contain no formant points. To add some points to it, use PitchTier: Add point...

For an example, see Source-filter synthesis.

Links to this page

- [Intro 8.1. Manipulation of pitch](#)
- [Manipulation: Replace pitch tier](#)

© *ppgb*, December 4, 2002

PitchTier: Add point...

A command to add a point to each selected PitchTier.

Arguments

Time (s)

the time at which a point is to be added.

Pitch (Hz)

the pitch value of the requested new point.

Behaviour

The tier is modified so that it contains the new point. If a point at the specified time was already present in the tier, nothing happens.

Links to this page

- [Create PitchTier...](#)
- [Intro 8.1. Manipulation of pitch](#)
- [Source-filter synthesis](#)

© *ppgb*, April 10, 2001

DurationTier

One of the types of objects in PRAAT. A DurationTier object contains a number of (*time*, *duration*) points, where *duration* is to be interpreted as a relative duration (e.g. the duration of a manipulated sound as compared to the duration of the original). For instance, if your DurationTier contains two points, one with a duration value of 1.5 at a time of 0.5 seconds and one with a duration value of 0.6 at a time of 1.1 seconds, this is to be interpreted as a relative duration of 1.5 (i.e. a slowing down) for all original times before 0.5 seconds, a relative duration of 0.6 (i.e. a speeding up) for all original times after 1.1 seconds, and a linear interpolation between 0.5 and 1.1 seconds (e.g. a relative duration of 1.2 at 0.7 seconds, and of 0.9 at 0.9 seconds).

See Intro 8.2. Manipulation of duration and Create DurationTier....

Links to this page

- [DurationTier: Add point...](#)
- [DurationTier: Get target duration...](#)
- [DurationTierEditor](#)
- [Get area...](#)
- [Get high index from time...](#)
- [Get low index from time...](#)
- [Get nearest index from time...](#)
- [Manipulation](#)
- [Manipulation: Extract duration tier](#)
- [Manipulation: Replace duration tier](#)
- [ManipulationEditor](#)
- [Remove point near...](#)
- [Remove point...](#)
- [Remove points between...](#)
- [time domain](#)
- [What's new?](#)

© ppgb, March 16, 2003



A command that is available if you select a Sound, Pitch, or PointProcess object. It gives you an acoustic representation of the selected object, if your loudspeakers are on and you did not "mute" your computer sound system.

A Play button is also available in the SoundRecorder window and in the View menu of the SoundEditor or TextGridEditor. In the editors, you will usually play a sound by clicking on any of the rectangles around the data.

Links to this page

- [Intro 8.2. Manipulation of duration](#)
-

© *ppgb*, December 12, 2002

View

One of the menus in several editors and in the manual.

Links to this page

- [Intro 8.2. Manipulation of duration](#)
- [Play](#)

© *ppgb*, May 12, 2001

IntensityTier

One of the types of objects in PRAAT. An IntensityTier object represents a time-stamped intensity contour, i.e., it contains a series of (*time*, *intensity*) points. The intensity values are in dB.

For examples, see Source-filter synthesis.

IntensityTier commands

Creation:

From scratch:

- o Create IntensityTier...
- o IntensityTier: Add point...

Copy from another object:

- o Intensity: To IntensityTier: trivial copying of linearly spaced points.
- o Intensity & TextTier: To IntensityTier...: copying interpolated values at specified points.
- o PointProcess: Up to IntensityTier...: equal values at specified points.

Viewing and editing:

- IntensityTierEditor

Conversion:

- IntensityTier: Down to PointProcess: copy times.

Synthesis (see Source-filter synthesis):

- Sound & IntensityTier: Multiply

Queries:

- Get low index from time...
- Get high index from time...
- Get nearest index from time...

Modification:

- Remove point...
- Remove point near...
- Remove points between...
- IntensityTier: Add point...

Links to this page

- [Get area...](#)
- [Intro 8.3. Manipulation of intensity](#)
- [What's new?](#)

© *ppgb*, March 16, 2003

Create IntensityTier...

A command in the New menu to create an empty IntensityTier object.

The resulting object will have the specified name and time domain, but contain no formant points. To add some points to it, use IntensityTier: Add point...

For an example, see Source-filter synthesis.

Links to this page

- [Intro 8.3. Manipulation of intensity](#)

© *ppgb*, December 4, 2002

Sound & IntensityTier: Multiply

A command to create a new Sound from the selected Sound and Intensity objects.

The resulting Sound equals the original sound, multiplied by a linear interpolation of the intensity. Afterwards, the resulting Sound is scaled so that its maximum absolute amplitude is 0.9.

Links to this page

- [IntensityTier](#)
- [Intro 8.3. Manipulation of intensity](#)
- [What's new?](#)

© *ppgb*, July 24, 2000

Show pulses

One of the commands in the Pulses menu of the SoundEditor and the TextGridEditor.

See Voice.

Links to this page

- [Voice 2. Jitter](#)
- [Voice 3. Shimmer](#)

© *ppgb*, March 16, 2003

Voice report

A command in the **Pulses** menu that will write to the Info window an extensive report about many voice parameters. See the Voice tutorial.

© *ppgb*, March 16, 2003

Voice 1. Voice breaks

Normal voices can easily maintain phonation for some time when saying [a]. Some pathological voices have trouble with it. This can be measured in PRAAT in two ways.

Fraction of locally unvoiced pitch frames

This is the fraction of pitch frames that are analysed as unvoiced (MDVP calls it DUV). If the pitch floor is 75 Hz, your Sound editor window will contain pitch measurements that are 0.01 seconds apart, so that if you select one second, there will be 100 pitch frames. If 86 of these are locally voiced, the Fraction will be 14 percent.

The usual pitch analysis contains a *path finder* that searches for a smooth path through the local pitch candidates. This path finder is temporarily switched off to determine the fraction of locally unvoiced frames. A frame is regarded as *locally* unvoiced if it has a voicing strength below the *voicing threshold* (whose standard value is 0.45), or a local peak below the *silence threshold* (whose standard value is 0.03).

In the voice report, the fraction of unvoiced frames will be reported as follows:

```
Fraction of locally unvoiced frames: 14.000% (14/100)
```

The numbers between parentheses are the number of unvoiced frames and the total number of frames, respectively (in MDVP, these are called NUV and SEG, respectively).

The normative value for the fraction of unvoiced frames is 0, i.e., normal healthy voices should have no trouble maintaining voicing during a sustained vowel. Every non-zero value can be considered a sign of pathology (like a common cold). Naturally, you will not select the leading and trailing silences when measuring this parameter.

Number of voice breaks

The number of distances between consecutive pulses that are longer than 1.25 divided by the pitch floor. Thus, if the pitch floor is 75 Hz, all inter-pulse intervals longer than 16.6667 milliseconds are regarded as voice breaks.

Degree of voice breaks

This is the total duration of the breaks between the voiced parts of the signal, divided by the total duration of the analysed part of the signal (MDVP calls it DVB). Since silences at the beginning and the end of the signal are not considered breaks, you will probably not want to select these silences when measuring this parameter.

In the voice report, the degree of voice breaks will be reported like this:

Degree of voice breaks: 29.529% (1.163061 s / 3.938685 s)

The numbers between parentheses are the total duration of the voice breaks and the duration of the analysed part of the signal, respectively.

Links to this page

- [Voice](#)
-

© *ppgb*, September 16, 2003

Voice 2. Jitter

You can measure jitter in the Sound editor window, after choosing Show pulses from the **Pulses** menu. You will see blue lines that can be thought of as representing the glottal closures. Use the Pulse menu to get the jitter in the selected part. You typically perform jitter measurements only on long sustained vowels. The voice report gives five kinds of jitter measurements.

Jitter (local)

This is the average absolute difference between consecutive periods, divided by the average period. MDVP calls this parameter *Jitt*, and gives 1.040% as a threshold for pathology.

Jitter (local, absolute)

This is the average absolute difference between consecutive periods. MDVP calls this parameter *Jita*, and gives 83.200 μ s as a threshold for pathology.

Jitter (rap)

This is the Relative Average Perturbation, the average absolute difference between a period and the average of it and its two neighbours, divided by the average period. MDVP gives 0.680% as a threshold for pathology.

Jitter (ppq5)

This is the five-point Period Perturbation Quotient, the average absolute difference between a period and the average of it and its four closest neighbours, divided by the average period. MDVP calls this parameter *PPQ*, and gives 0.840% as a threshold for pathology.

Jitter (ddp)

This is the average absolute difference between consecutive differences between consecutive periods, divided by the average period. This is PRAAT's original **Get jitter**. The value is three times RAP.

Links to this page

- [PointProcess: Get jitter \(ddp\)...](#)
- [PointProcess: Get jitter \(local\)...](#)
- [PointProcess: Get jitter \(local, absolute\)...](#)
- [PointProcess: Get jitter \(ppq5\)...](#)
- [PointProcess: Get jitter \(rap\)...](#)
- [Voice](#)

© *ppgb*, May 31, 2003

Voice 3. Shimmer

You can measure shimmer in the Sound editor window, after choosing Show pulses from the **Pulses** menu. You will see blue lines that can be thought of as representing the glottal closures. Use the Pulse menu to get the shimmer in the selected part. You typically perform shimmer measurements only on long sustained vowels. The voice report gives six kinds of shimmer measurements.

Shimmer (local)

This is the average absolute difference between the amplitudes of consecutive periods, divided by the average amplitude. MDVP calls this parameter *Shim*, and gives 3.810% as a threshold for pathology.

Shimmer (local, dB)

This is the average absolute base-10 logarithm of the difference between the amplitudes of consecutive periods, multiplied by 20. MDVP calls this parameter *ShdB*, and gives 0.350 dB as a threshold for pathology.

Shimmer (apq3)

This is the three-point Amplitude Perturbation Quotient, the average absolute difference between the amplitude of a period and the average of the amplitudes of its neighbours, divided by the average amplitude.

Shimmer (apq5)

This is the five-point Amplitude Perturbation Quotient, the average absolute difference between the amplitude of a period and the average of the amplitudes of it and its four closest neighbours, divided by the average amplitude.

Shimmer (apq11)

This is the 11-point Amplitude Perturbation Quotient, the average absolute difference between the amplitude of a period and the average of the amplitudes of it and its ten closest neighbours, divided by the average amplitude. MDVP calls this parameter *APQ*, and gives 3.070% as a threshold for pathology.

Shimmer (ddp)

This is the average absolute difference between consecutive differences between the amplitudes of consecutive periods. This is PRAAT's original **Get shimmer**. The value is three times APQ3.

Links to this page

- [Voice](#)
-

© *ppgb*, May 21, 2003

Voice 4. Additive noise

For a signal that can be assumed periodic (i.e., a sustained vowel), the signal-to-noise ratio equals the harmonics-to-noise ratio, which you get can get by selecting a Sound and choosing one of the "To Harmonicity..." commands from the Periodicity submenu (for the algorithm, see Sound: To Harmonicity (ac)... or Sound: To Harmonicity (cc)...). These are the world's most sensitive HNR measurements (up to 90 dB). For more information, see the Harmonicity manual page.

Links to this page

- [Voice](#)

© ppgb, December 6, 2002

Voice 5. Comparison with other programs

Voicing, jitter, and shimmer measurements made by PRAAT cannot always be compared directly with those made by other programs such as MDVP. The causes are the voicing decision strategy and the accuracy of period and peak determination.

Voicing decision strategy

Different programs use very different methods for deciding whether an irregular part of the signal is voiced or not. A comparison of Boersma (1993) for PRAAT and Deliyski (1993) for MDVP leads to the following considerations. Both PRAAT and MDVP use an autocorrelation method for pitch analysis, but MDVP quantizes the amplitudes into the values -1, 0, and +1 before computing the autocorrelation, whereas PRAAT uses the original amplitude. Also, PRAAT corrects the autocorrelation function by dividing it by the autocorrelation function of the window, unlike any other program. Lastly, PRAAT uses sinc interpolation to compute an accurate estimate of the height of the autocorrelation peaks, unlike any other program. All three of these differences (and there are more) influence the measurement of the height of the autocorrelation peak at $1/F_0$. This height is generally taken as a criterion for voicing: if it is more than the *voicing threshold* (which you can change with Pitch settings..., the frame is considered voiced, otherwise voiceless. In PRAAT, the standard voicing threshold is 0.45, in MDVP it is 0.29, which suggests that MDVP tends to regard more frames as voiced than PRAAT. But the difference between these two numbers may partly be explained by MDVP's failure to correct the autocorrelation function and by MDVP's failure to do an accurate sinc interpolation: both of these failures cause the measured height of the peak at $1/F_0$ (in MDVP) to be lower than the real height, as explained by Boersma (1993).

Peak determination

- Voice
-

© ppgb, September 16, 2003

Sound files 1. General structure

This is chapter 1 of the Sound files tutorial. It describes the general structure of the headers and data parts of sound files.

- 1.1. Sampling (sampling frequency)
- 1.2. Quantization (linear, μ -law, A-law)
- 1.3. Channels (mono, stereo)
- 1.4. The header
- 1.5. Size
- 1.6. Compression

© *ppgb*, January 23, 2000

Sound files 1.1. Sampling

When a sound signal from a microphone or on a tape needs to be read into a computer, it is *digitized*, which means that it is *sampled* and *quantized*.

Sampling is the discretization of the time domain of the signal: each second of the signal is divided up into 11025, 22050, or 44100 slices (or any other suitable number), and a *sample value* is associated with each slice. For instance, a continuous 377-Hz sine wave is expressed by

$$x(t) = \sin(2\pi 377 t)$$

If the *sampling frequency* (or *sample rate*) is 44100 Hz, this sine wave will be sampled at points spaced $\Delta t = 1/44100$ second apart, and the sample values will be

$$x_i = \sin(2\pi 377 (t_0 + i \Delta t))$$

where t_0 is the time after which sampling begins. Δt is called the *sample period*.

Quantization is handled in the next section (\SS1.2).

Links to this page

- [Sound files](#)
- [Sound files 1. General structure](#)
- [Sound files 1.4. The header](#)

© ppgb, March 30, 2004

Sound files 1.2. Quantization

Apart from sampling (\SS1.1), digitization also involves quantization, which is the conversion of a sample value into a number that fits into 8 or 16 bits.

Links to this page

- [Sound files](#)
- [Sound files 1. General structure](#)
- [Sound files 1.4. The header](#)

© *ppgb*, January 26, 2000

Sound files 1.3. Channels

For most file types, Praat supports sounds with one channel (mono) and two channels (stereo).

Links to this page

- [Sound files](#)
- [Sound files 1. General structure](#)
- [Sound files 1.4. The header](#)

© *ppgb*, January 26, 2000

Sound files 1.4. The header

The bulk of most sound files is formed by the samples. The sample part is usually preceded by a header, which contains information about:

1. The type of file (WAV, AIFF, AIFC, NeXT/Sun, NIST, Kay,...).
2. The sampling frequency (\SS1.1).
3. The encoding (linear, 8 or 16 bit, byte order, μ -law, A-law, see \SS1.2).
4. The number of channels (mono, stereo, see \SS1.3).
5. The duration (usually the number of samples).

Links to this page

- [Sound files](#)
- [Sound files 1. General structure](#)

© *ppgb*, February 23, 2004

Sound files 1.5. Size

The size of a sound file is equal to the size of the header plus the product of the number of samples, the quantization size, and the number of channels.

For instance, a stereo sound with a duration of 3 seconds, sampled at 44100 Hz, would, when written into a 16-bit NeXT/Sun file, take up a disk space of

$$28 + 2 * 3.0 * 44100 * 2 = 529228 \text{ bytes}$$

whereas the same sound, when averaged to mono, downsampled to 8000 Hz, and written into a μ -law NeXT/Sun file, take up only a disk space of

$$28 + 1 * 3.0 * 8000 * 1 = 24028 \text{ bytes}$$

The first example is typical of CD quality, the second of telephone speech.

Links to this page

- [Sound files](#)
- [Sound files 1. General structure](#)

© *ppgb*, January 26, 2000

Sound files 1.6. Compression

Praat used to be able to read some compressed sound file formats (shortened NIST, Polyphone), but because of licensing problems (Praat went open source, Shorten did not), you now need to use other (freely available) programs to do the conversion before reading them into Praat. For MP3-encoded audio files, one can e.g. use iTunes® on the Macintosh.

Links to this page

- [Sound files](#)
- [Sound files 1. General structure](#)
- [Sound files 2.5. NIST files](#)

© *ppgb*, March 8, 2003

Sound files 2. File types

- 2.1. WAV files
- 2.2. AIFF files
- 2.3. AIFC files
- 2.4. NeXT/Sun (.au) files
- 2.5. NIST files

Links to this page

- [Sound files](#)
-

© *ppgb*, February 23, 2004

Sound files 2.1. WAV files

The audio file type most commonly used on Windows computers, also very common on the Internet.

Links to this page

- [Sound files](#)
- [Sound files 2. File types](#)

© *ppgb*, February 23, 2004

Sound files 2.2. AIFF files

AIFF stands for: Audio Interchange File Format.

This standard format for sound files was defined by Apple. It is also the format of the sound files on the Iris Indigo, where each sample is quantized into 16 bits.

Links to this page

- [Sound files](#)
- [Sound files 2. File types](#)
- [Sound files 2.3. AIFC files](#)

© *ppgb*, February 23, 2004

Sound files 2.3. AIFC files

AIFC is short for AIFF(C) or AIFF-C, i.e. the Audio Interchange File Format (\SS2.2) with optional compression.

Praat reads and write uncompressed AIFC files, but does not support compressed AIFC files.

Links to this page

- [Sound files](#)
- [Sound files 2. File types](#)

© *ppgb*, February 23, 2004

Sound files 2.4. NeXT/Sun (.au) files

This is the format of the sound files on the Sun.

Reading

To read a **Sound** from a Sun audio file on disk, use Read from file....Praat then asks you for a file name. After you click OK, Praat determines the encoding of the file. If the encoding is 16-bit linear, the 16-bit sample values are divided by 32768 so that the amplitude of the resulting Sound is between -1.0 and +1.0. If the encoding is 8-bit μ -law, the 16-bit sample value is determined by table look-up first.

The resulting Sound will appear in the list of objects; its name will be equal to the file name, without extension.

Writing

Use Write to NeXT/Sun file.... The samples of the Sound are multiplied by 32768 and quantized between -32768 and 32767; the result is written to the file in 16-bit mono Sun audio format.

To avoid clipping, keep the absolute amplitude below 1.000. If the maximum sound pressure level is 91 dB (top = 32767), the quantization threshold is (top = 1/2) -5 dB.

Links to this page

- [Sound files](#)
- [Sound files 2. File types](#)

© ppgb, February 23, 2004

Sound files 2.5. NIST files

An audio file type used by speech researchers. Used, for instance, in the TIMIT database. Praat reads several kinds of NIST files: big-endian, little-endian, μ -law, A-law, Polyphone. NIST files compressed with **shorten** are no longer supported (see \SS1.6)

Links to this page

- [Sound files](#)
- [Sound files 2. File types](#)

© *ppgb*, February 23, 2004

Sound files 3. Files that Praat can read

Praat can read five types of standard sound files in several formats, and a number of proprietary types of sound files as well.

Standard sound files

The Read menu contains two commands for opening sound files:

- With Read from file..., you read the entire file into memory. A Sound object will appear in the list. This is appropriate if your sound is not too long to fit into memory. The advantage of having a Sound object is that you can perform analysis on it directly.
- With Open long sound file..., you open a sound file that is too long to read into memory completely. A LongSound object will appear in the list. You will typically choose View to view the contents of this sound. Praat will only read so much of the file into memory as is needed to play or display parts of it. In order to perform analyses on a part of the LongSound, you must select this part and copy it as a Sound object to the list.

Both commands understand the following five standard audio file formats:

- WAV:
 - linear 16-bit little-endian
 - 8-bit μ -law
 - 8-bit A-law
 - linear 8-bit unsigned
- AIFF or AIFC:
 - linear 16-bit big-endian
 - linear 8-bit signed
- NeXT/Sun (.au):
 - linear 16-bit big-endian
 - 8-bit μ -law
 - 8-bit A-law
 - linear 8-bit signed
- NIST:
 - linear 16-bit little-endian
 - linear 16-bit big-endian
 - 8-bit μ -law
 - 8-bit A-law
 - linear 8-bit signed

Links to this page

- [Sound files](#)
-

© *ppgb*, February 23, 2004

Sound files 4. Files that Praat can write

Praat can write five types of standard sound files in an appropriate linear 16-bit formats, and a number of proprietary types of sound files as well:

- Write to WAV file... (16-bit little-endian)
- Write to AIFF file... (16-bit big-endian)
- Write to AIFC file... (16-bit big-endian)
- Write to NeXT/Sun file... (16-bit big-endian)
- Write to NIST file... (16-bit little-endian)

Links to this page

- [Sound files](#)
-

© *ppgb*, February 23, 2004

Write to AIFC file...

With this command, you write one or more selected Sound and/or LongSound objects to a single 16-bit big-endian AIFC file on disk. See the Sound files tutorial for more information.

Links to this page

- [How to concatenate sound files](#)
- [Sound files 4. Files that Praat can write](#)

© *ppgb*, January 26, 2000

Write to AIFF file...

With this command, you write one or more selected Sound and/or LongSound objects to a single 16-bit big-endian AIFF file on disk. See the Sound files tutorial for more information.

Links to this page

- [AIFF and AIFC files](#)
- [How to concatenate sound files](#)
- [Sound files 4. Files that Praat can write](#)

© *ppgb*, January 26, 2000

Write to NeXT/Sun file...

With this command, you write one or more selected Sound and/or LongSound objects to a single 16-bit big-endian NeXT/Sun (.au) file on disk. See the Sound files tutorial for more information.

Links to this page

- [How to concatenate sound files](#)
- [Sound files 2.4. NeXT/Sun \(.au\) files](#)
- [Sound files 4. Files that Praat can write](#)

© *ppgb*, January 26, 2000

Write to NIST file...

With this command, you write one or more selected Sound and/or LongSound objects to a single 16-bit little-endian NIST audio file on disk. See the Sound files tutorial for more information.

Links to this page

- [How to concatenate sound files](#)
- [Sound files 4. Files that Praat can write](#)

© *ppgb*, January 26, 2000

Write to WAV file...

With this command, you write one or more selected Sound and/or LongSound objects to a single 16-bit little-endian WAV file on disk. See the Sound files tutorial for more information.

Links to this page

- [How to concatenate sound files](#)
- [Sound files 4. Files that Praat can write](#)

© *ppgb*, January 26, 2000

Spectrum: Filter (pass Hann band)...

A command to modify every selected Spectrum object.

The complex values in the **Spectrum** are multiplied by real-valued sine shapes and straight lines, according to the following figure:

[sorry, no pictures yet in the web version of this manual]

Arguments

From frequency (standard value: 500 Hz)

the lower edge of the pass band (f_1 in the figure). The value zero is special: the filter then acts as a low-pass filter.

To frequency (standard value: 1000 Hz)

the upper edge of the pass band (f_2 in the figure). The value zero is special: the filter then acts as a high-pass filter.

Smoothing (standard value: 100 Hz)

the width of the region between pass and stop (w in the figure).

Usage

Because of its symmetric Hann-like shape, the filter is especially useful for decomposing the Spectrum into consecutive bands. For instance, we can decompose the spectrum into the bands 0-500 Hz, 500-1000 Hz, 1000-2000 Hz, and 2000-"0" Hz:

[sorry, no pictures yet in the web version of this manual]

By adding the four bands together, we get the original spectrum again.

A complementary filter is described at Spectrum: Filter (stop Hann band)...

See the Filtering tutorial for information on the need for smoothing and a comparative discussion of various filters.

Links to this page

- [Sound: Filter \(pass Hann band\)...](#)

© ppgb, September 16, 2003

Spectrum: Filter (stop Hann band)...

A command to modify every selected Spectrum object.

The complex values in the **Spectrum** are multiplied by real-valued sine shapes and straight lines, according to the following figure:

[sorry, no pictures yet in the web version of this manual]

Arguments

From frequency (standard value: 500 Hz)

the lower edge of the stop band (f_1 in the figure). The value zero is special: the filter then acts as a high-pass filter.

To frequency (standard value: 1000 Hz)

the upper edge of the stop band (f_2 in the figure). The value zero is special: the filter then acts as a low-pass filter.

Smoothing (standard value: 100 Hz)

the width of the region between stop and pass (w in the figure).

Usage

This filter is the complement from the pass-band filter (Spectrum: Filter (pass Hann band)...). For instance, we can decompose the spectrum into the above stop-band spectrum and a band from 500 to 1000 Hz:

[sorry, no pictures yet in the web version of this manual]

By adding the two spectra together, we get the original spectrum again.

See the Filtering tutorial for information on the need for smoothing and a comparative discussion of various filters.

Links to this page

- [Sound: Filter \(stop Hann band\)...](#)

© ppgb, September 16, 2003

Sound: Filter (pass Hann band)...

A command to convert every selected Sound object into a filtered sound.

The filtering is done in the frequency domain. This command is equivalent to the following sequence:

1. Sound: To Spectrum (fft)
2. Spectrum: Filter (pass Hann band)...
3. Spectrum: To Sound (fft)

For a comparative discussion of various filtering methods, see the [Filtering tutorial](#).

For a complementary filter, see [Sound: Filter \(stop Hann band\)...](#)

Links to this page

- [Sound: Filter \(formula\)...](#)

© *ppgb*, May 26, 2002

Sound: Filter (stop Hann band)...

A command to convert every selected Sound object into a filtered sound.

The filtering is done in the frequency domain. This command is equivalent to the following sequence:

1. Sound: To Spectrum (fft)
2. Spectrum: Filter (stop Hann band)...
3. Spectrum: To Sound (fft)

For a comparative discussion of various filtering methods, see the [Filtering tutorial](#).

For a complementary filter, see [Sound: Filter \(pass Hann band\)...](#)

© *ppgb*, May 26, 2002

Sound: Filter (formula)...

A command to convert every selected Sound object into a filtered sound.

The filtering is done in the frequency domain. This command is equivalent to the following sequence:

1. Sound: To Spectrum (fft)
2. Matrix: Formula...
3. Spectrum: To Sound (fft)

For a comparative discussion of various filtering methods, see the Filtering tutorial.

The example formula is the following:

```
if x<500 or x>1000 then 0 else self fi; rectangular band
```

This formula represents a rectangular pass band between 500 Hz and 1000 Hz (x is the frequency). Rectangular bands are *not* recommended, since they may lead to an appreciable amount of *ringing* in the time domain. The transition between stop and pass band should be smooth, as e.g. in Sound: Filter (pass Hann band)....

© ppgb, September 16, 2003

Band filtering in the frequency domain

We describe how band filtering in the frequency domain is performed.

We start with a Sound and end with a filter bank representation of this sound. We assume a standard analysis context: a sound divided into frames according to a certain *window length* and *time step*. We will simulate a filterbank with N filters.

The algorithm for each sound frame proceeds in the following way:

1. Apply a Gaussian window to the sound frame.
2. Convert the windowed frame into a Spectrum object.
3. Convert the spectral amplitudes to *energy* values by squaring the real and imaginary parts and multiplying by df , the frequency distance between two successive frequency points in the spectrum. Since the Spectrum object only contains positive frequencies, we have to multiply all energy values, except the first and the last frequency, by another factor of 2 to compensate for negative frequencies.
4. For each of the N filters in the filter bank: determine the inner product of its filter function with the energies as determined in the previous step. The result of each inner product is the energy in the corresponding filter.
5. Convert the energies in each filter to power by dividing by the *window length*.
6. Correct the power, due to the windowing of the frame, by dividing by the integral of the *squared* windowing function.
7. Convert all power values to *dB*'s according to $10 * \log_{10}(\text{power} / 4 \cdot 10^{-10})$.

Links to this page

- [Filtering](#)
- [Sound & Pitch: To FormantFilter...](#)
- [Sound: To BarkFilter...](#)
- [Sound: To FormantFilter...](#)
- [Sound: To MelFilter...](#)
- [What's new?](#)

© djmw, April 4, 2001

Sound: Filter (one formant)...

A command to filter every selected Sound object, with a single formant of a specified frequency and bandwidth.

Algorithm

Two recursive filter coefficients are computed as follows:

$$p = -2 \exp(-\pi \textit{bandwidth} \textit{dt}) \cos(2\pi \textit{frequency} \textit{dt})$$

$$q = \exp(-2\pi \textit{bandwidth} \textit{dt})$$

where dt is the sample period. The new signal y is then computed from the old signal x and itself as

$$y_1 := x_1$$

$$y_2 := x_2 - p y_1$$

$$\forall n \geq 3: y_n := x_n - p y_{n-1} - q y_{n-2}$$

After filtering, the sound y is scaled so that its absolute extremum is 0.9.

For a comparative discussion of various filtering methods, see the Filtering tutorial.

This filter has an in-line version: [Sound: Filter with one formant \(in-line\)...](#)

© ppgb, March 9, 2003

Sound: Filter (pre-emphasis)...

A command to filter each selected Sound object. The resulting Sound object has a higher spectral slope.

The reverse of Sound: Filter (de-emphasis)...

Argument

From frequency (Hz)

the frequency F above which the spectral slope will increase by 6 dB/octave.

Algorithm

The pre-emphasis factor α is computed as

$$\alpha = \exp(-2 \pi F \Delta t)$$

where Δt is the sampling period of the sound. The new sound y is then computed as:

$$y_i = x_i - \alpha x_{i-1}$$

Links to this page

- [Filtering](#)
- [Sound: Pre-emphasize \(in-line\)...](#)

© ppgb, March 9, 2003

Sound: Filter (de-emphasis)...

A command to filter every selected Sound object. The resulting Sound object has a lower spectral slope.

The reverse of Sound: Filter (pre-emphasis).... For an example, see Source-filter synthesis.

Argument

From frequency (Hz)

the frequency F above which the spectral slope will decrease by 6 dB/octave.

Algorithm

The de-emphasis factor α is computed as

$$\alpha = \exp(-2 \pi F \Delta t)$$

where Δt is the sampling period of the sound. The new sound y is then computed recursively as:

$$y_1 = x_1$$

$$y_i = x_i + \alpha y_{i-1}$$

Links to this page

- [Filtering](#)
- [Sound: De-emphasize \(in-line\)...](#)

© ppgb, March 9, 2003

Sound: Filter with one formant (in-line)...

A command to filter every selected Sound object in-line, with a single formant of a specified frequency and bandwidth.

This is the in-line version of Sound: Filter (one formant)..., i.e. it does not create a new Sound object but modifies the selected object.

Links to this page

- [Filtering](#)
-

© *ppgb*, March 9, 2003

Sound: Pre-emphasize (in-line)...

A command to change the spectral slope of every selected Sound object.

The reverse of Sound: De-emphasize (in-line)...

This is the in-line version of Sound: Filter (pre-emphasis)..., i.e., it does not create a new Sound object but modifies an existing object.

Algorithm

The pre-emphasis factor α is computed as

$$\alpha = \exp(-2 \pi F \Delta t)$$

where Δt is the sampling period of the sound. Every sample x_i of the sound, except x_1 , is then changed, going down from the last sample:

$$x_i = x_i - \alpha x_{i-1}$$

Links to this page

- [Filtering](#)
- [Sound: To Formant \(burg\)...](#)
- [What's new?](#)

© *ppgb*, March 9, 2003

Sound: De-emphasize (in-line)...

A command to change the spectral slope of every selected Sound object.

The reverse of Sound: Pre-emphasize (in-line).... For an example, see Source-filter synthesis.

This is the in-line version of Sound: Filter (de-emphasis)...., i.e., it does not create a new Sound object but modifies an existing object.

Argument

From frequency (Hz)

the frequency F above which the spectral slope will decrease by 6 dB/octave.

Algorithm

The de-emphasis factor α is computed as

$$\alpha = \exp(-2 \pi F \Delta t)$$

where Δt is the sampling period of the sound. Every sample x_i of the sound, except x_1 , is then changed, going up from the second sample:

$$x_i = x_i + \alpha x_{i-1}$$

Links to this page

- [Filtering](#)

© ppgb, March 9, 2003

Sounds: Convolve

A command to convolve two Sound objects with each other.

The convolution of two time signals $A(t)$ and $B(t)$ is defined as

$$(A * B)(t) \equiv \int A(\tau) B(t-\tau) d\tau$$

Convolution is commutative, i.e. the convolution of A and B equals the convolution of B and A .

Algorithm

Since convolution in the time domain amounts to multiplication in the frequency domain, both sounds are FFT-ed, the resulting spectra are multiplied, and the resulting product spectrum is FFT-ed back to give the convoluted sound.

Links to this page

- Filtering
- Source-filter synthesis

© ppgb, March 23, 1998

Sound & Formant: Filter

A command to create a new Sound from the selected Sound and Formant objects.

For examples, see Source-filter synthesis.

The resulting Sound is scaled so that its maximum absolute amplitude is 0.99. If you don't want this, use Sound & Formant: Filter (no scale) instead.

Links to this page

- [Filtering](#)
- [Sound: To Formant \(keep all\)...](#)

© *ppgb*, November 19, 1999

Sound & FormantTier: Filter

A command to create a new Sound from the selected Sound and FormantTier objects.

For examples, see Source-filter synthesis.

The resulting Sound is scaled so that its maximum absolute amplitude is 0.99. If you don't want this, use Sound & FormantTier: Filter (no scale) instead.

Links to this page

- [Filtering](#)
- [What's new?](#)

© *ppgb*, November 20, 1999

LPC & Sound: Filter...

A command that creates a new Sound object from one Sound and one LPC object which have been selected together.

Settings

Use LPC gain

Determines whether the gain from the LPC is used in the synthesis.

Behaviour

Filters the selected Sound by the selected LPC-filter.

When the LPC-gain is used the samples in the new Sound will be multiplied with the square root of the corresponding LPC-gain value.

In **Z**-domain notation: $\mathbf{O}(z) = \mathbf{H}(z) \cdot \mathbf{E}(z)$, where $\mathbf{E}(z)$ is the selected filter input Sound, $\mathbf{H}(z)$ the selected LPC filter, and, $\mathbf{O}(z)$ the filter output (the new Sound that will appear in the List of objects).

Links to this page

- [Filtering](#)
- [Source-filter synthesis](#)

© *djmw*, April 7, 2004

LPC & Sound: Filter (inverse)

A command that creates a new Sound object from one Sound and one LPC object which have been selected together.

Behaviour

Given a filter (the selected LPC) and its output (the selected Sound), its input is reconstructed (the new Sound that will appear in the List of objects).

In Z-domain notation: $\mathbf{E}(z) = \mathbf{O}(z) / \mathbf{H}(z)$, where $\mathbf{O}(z)$ is the filter output Sound, $\mathbf{H}(z)$ the LPC filter, and, $\mathbf{E}(z)$ the filter input Sound. (Selecting this newly generated Sound and the LPC, choosing the option 'Filter...' generates a Sound that is identical to the Sound that originated the LPC.)

Links to this page

- Filtering

© djmw, January 26, 1997

Sound: LPC analysis

You can perform this analysis by selecting one or more Sound objects and choosing the appropriate command to generate an LPC.

The acronym LPC stands for Linear Predictive Coding.

In the LPC analysis one tries to predict x_n on the basis of the p previous samples,

$$x'_n = \sum a_k x_{n-k}$$

then $\{a_1, a_2, \dots, a_p\}$ can be chosen to minimize the prediction power Q_p where

$$Q_p = E[|x_n - x'_n|^2].$$

Several different algorithms exist for minimizing Q_p :

- To LPC (autocorrelation)...
- To LPC (covariance)...
- To LPC (burg)...
- To LPC (marple)...

Links to this page

- [Source-filter synthesis](#)
-

© djmw, January 26, 1997

Sound: Resample...

A command that creates new Sound objects from the selected Sounds.

Purpose

High-precision resampling from any sampling frequency to any other sampling frequency.

Arguments

Sampling frequency

the new sampling frequency, in Hertz.

Precision

the depth of the interpolation, in samples (standard is 50). This determines the quality of the interpolation used in resampling.

Algorithm

If *Precision* is 1, the method is linear interpolation, which is inaccurate but fast.

If *Precision* is greater than 1, the method is $\sin(x)/x$ ("sinc") interpolation, with a depth equal to *Precision*. For higher *Precision*, the algorithm is slower but more accurate.

If *Sampling frequency* is less than the sampling frequency of the selected sound, an anti-aliasing low-pass filtering is performed prior to resampling.

Behaviour

A new Sound will appear in the list of objects, bearing the same name as the original Sound, followed by the sampling frequency. For instance, the Sound "hallo" will give a new Sound "hallo_10000".

Links to this page

- [Sound: To Formant \(burg\)...](#)
- [Sound: To LPC \(autocorrelation\)...](#)
- [Sound: To LPC \(burg\)...](#)
- [Sound: To LPC \(covariance\)...](#)
- [Sound: To LPC \(marple\)...](#)
- [Source-filter synthesis](#)

Sound: To LPC (burg)...

With this command you create a new LPC from every selected Sound, using "Burg's" method.

Warning

You are advised not to use this command for formant analysis. For formant analysis, instead use **Sound: To Formant (burg)...**, which also works via LPC (linear predictive coding). This is because **Sound: To Formant (burg)...** lets you specify a maximum frequency, whereas the **To LPC** commands automatically use the Nyquist frequency as their maximum frequency. If you do use one of the **To LPC** commands for formant analysis, you may therefore want to downsample the sound first. For instance, if you want five formants below 5500 Hz but your Sound has a sampling frequency of 44100 Hz, you have to downsample the sound to 11000 Hz with the **Sound: Resample...** command. After that, you can use the **To LPC** commands, with a prediction order of 10 or 11.

Settings

Prediction order

the number of linear prediction coefficients, also called the *number of poles*. Choose this number at least twice as large as the number of spectral peaks that you want to detect.

Analysis window duration (s)

the effective duration of each analysis frame, in seconds.

Time step (s)

the time step between two consecutive analysis frames.

Pre-emphasis frequency (Hz)

a +6dB / octave filtering will be applied above this frequency. A pre-emphasis frequency of 48.47 Hz for a signal with a sampling frequency of 10 kHz approximately corresponds to a value of $a = 0.97$ for the filter $y_n = x_n - a \cdot x_{n-1}$. The relation between a and the pre-emphasis frequency is: $a = \exp(-2 \cdot \pi \cdot \text{preemphasisFrequency} / \text{samplingFrequency})$. If you do not want pre-emphasis, choose a frequency greater than the Nyquist frequency.

Algorithm

Burg's algorithm is described in Anderson (1978)

Links to this page

- [Formants & LPC submenu](#)
 - [Sound: LPC analysis](#)
 - [Source-filter synthesis](#)
-

© *David Weenink & Paul Boersma, April 7, 2004*

LPC

One of the types of objects in PRAAT.

An object of type LPC represents filter coefficients as a function of time. The coefficients are represented in frames with constant sampling period.

LPC commands

Creation:

- Sound: To LPC (autocorrelation)...
- Sound: To LPC (covariance)...
- Sound: To LPC (burg)...
- Sound: To LPC (marple)...

Conversion

- To LFCC...
- To Spectrogram...
- To Spectrum (slice)...
- To Polynomial (slice)...

Links to this page

- LPC & Sound: Filter (inverse)
- LPC & Sound: Filter...
- LPC: Draw gain...
- LPC: Draw poles...
- LPC: To Formant
- LPC: To Matrix
- LPC: To VocalTract (slice)...
- Sound: LPC analysis
- Source-filter synthesis

© *djmw*, June 10, 1999

Inspect

One of the fixed buttons in the Object window.

You can use this command after selecting one object in the list.

The contents of the selected object will become visible in a Data Editor. You can then view and change the data in the object, but beware: changing the data directly in this way may render them inconsistent.

Changes that you make to the data with another Editor (e.g., a SoundEditor), or with the commands under *Modify*, are immediately reflected in the top-level Data Editor; any subeditors are destroyed, however, because they may now refer to invalid data.

Changes that you make to the data with a Data Editor, are immediately reflected in any open type-specific Editors (e.g., a SoundEditor).

Links to this page

- [BarkFilter](#)
- [Categories](#)
- [Configuration](#)
- [Confusion](#)
- [Editors](#)
- [Eigen](#)
- [Excitation](#)
- [Formant](#)
- [FormantFilter](#)
- [FormantTier](#)
- [Get frame number from time...](#)
- [Get number of frames](#)
- [Get number of samples](#)
- [Get sample number from time...](#)
- [Get sampling period](#)
- [Get time from frame number...](#)
- [Get time from sample number...](#)
- [Get time step](#)
- [IntervalTier](#)
- [Ltas](#)
- [Manipulation](#)
- [Matrix](#)
- [MelFilter](#)
- [OT learning 7. Learning from overt forms](#)
- [OTAnyGrammar examples](#)

- OTGrammar
 - Pattern
 - PCA
 - Pitch
 - Proximity
 - Sound
 - Source-filter synthesis
 - Spectrogram
 - Spectrum
 - SSCP
 - time domain
 - WordList
-

© *ppgb*, September 4, 1996

LPC: To Spectrogram...

You can choose this command after selecting 1 or more LPC objects.

Settings

Minimum frequency resolution (Hz)

successive frequencies in the Spectrum will be maximally this distance apart

Bandwidth reduction (Hz)

formants with small bandwidths show up very well as darker regions in the spectrogram because the poles lie close to the contour along which a spectrum is computed (the unit circle in the z-plane).

Peak enhancement can be realized by computing a spectrum in the z-plane along a contour of radius $r = \exp(-\pi \cdot \text{BandwidthReduction} / \text{samplingFrequency})$.

De-emphasis frequency (Hz)

Performs de-emphasis when value is in the interval (0, Nyquist frequency)

Algorithm

For each LPC_Frame the corresponding Spectrum will be calculated according to the algorithm explained in LPC: To Spectrum (slice).... For each frequency the power, i.e., the square of the complex values, will be stored in the corresponding area in the Spectrogram.

Links to this page

- Source-filter synthesis

© djmw, April 7, 2004

LPC: To Formant

You can choose this command after selecting 1 or more LPC objects.

Behaviour

For each LPC_Frame, the zeros of the linear prediction polynomial are extracted. Zeros that are outside the unit circle are reflected into it. Next, formant frequencies and bandwidths are calculated from all the roots that have the imaginary part positive, i.e., that lie in the upper half of the unit circle. Formant frequencies smaller than 50 Hz or larger than (*Nyquist_frequency* - 50) are discarded. The remaining frequencies and bandwidths are sorted and copied to the Formant_Frame. Finally, the *gain* field of the LPC is copied to the *intensity* field of the Formant_Frame.

Algorithm

The root finder is Laguerre's method followed by root polishing, see Press et al. (1992).

Warning

- The formant values can be very inaccurate if you did not resample the Sound before the LPC-analysis (consult the Source-filter synthesis tutorial).
- The results of the root finder may not always be accurate when more than 30 roots have to be found.

© djmw, January 23, 1997

Formant: Speckle...

A command to draw the selected Formant objects to the Picture window.

Behaviour

Every formant value is drawn as a small circle, filled with the current colour.

Arguments

From time (s), *To time* (s)

the time domain of the drawing. If *To time* is not greater than *From time*, the entire formant contour is drawn.

Maximum frequency (Hz)

the height of the y axis. For speech, 5000 Hz is a usual value.

Dynamic range (dB)

determines the signal intensity (as stored in each formant frame) below which no formants will be drawn. If zero, all formants will be drawn. The standard value is 30 dB, which would mean that formants in frames with intensities less than the maximum intensity minus 30 dB will not be drawn.

Garnish

determines whether axes, numbers, and texts ("Time", "Formant frequency") will be drawn in the margins around the picture. Turn this button off if you prefer to garnish you picture by yourself with the Margins menu.

Links to this page

- [Formant: Draw tracks...](#)
- [Source-filter synthesis](#)

© ppgb, September 16, 2003

Nyquist frequency

The *Nyquist frequency* is the bandwidth of a sampled signal, and is equal to half the sampling frequency of that signal. If the sampled signal should represent a continuous spectral range starting at 0 Hz (which is the most common case for speech recordings), the Nyquist frequency is the highest frequency that the sampled signal can unambiguously represent.

Example

If a speech signal is sampled at 22050 Hz, the highest frequency that we can expect to be present in the sampled signal is 11025 Hz. This means that to heed this expectation, we should run the continuous signal through a low-pass filter with a cut-off frequency below 11025 Hz; otherwise, we would experience the phenomenon of aliasing.

Of course, with a sampling frequency of 22050 Hz we could also represent a signal band-limited between, say, 40000 Hz and 51025 Hz, but this seems less useful in speech research.

Links to this page

- [Create Sound from gamma-tone...](#)
- [Create Sound from tone complex...](#)
- [Formulas 7. Attributes of objects](#)
- [LPC: To Spectrogram...](#)
- [LPC: To Spectrum \(slice\)...](#)
- [Polynomial: To Spectrum...](#)
- [Sound: To LPC \(autocorrelation\)...](#)
- [Sound: To LPC \(burg\)...](#)
- [Sound: To LPC \(covariance\)...](#)
- [Sound: To LPC \(marple\)...](#)
- [Source-filter synthesis](#)
- [Spectrum: Get centre of gravity...](#)
- [Spectrum: Get standard deviation...](#)
- [Vector peak interpolation](#)
- [Vector value interpolation](#)

© ppgb, March 31, 2004

FormantTier

One of the types of objects in PRAAT.

A FormantTier object represents spectral structure as a function of time: a *formant contour*. Unlike the evenly sampled Formant object, it consists of a number of formant *points* (or *targets*), sorted by time. Each point contains several formant/bandwidth pairs.

For examples, see Source-filter synthesis.

FormantTier commands

Creation:

From scratch:

- o Create FormantTier...
- o FormantTier: Add point...

Copy from another object:

- o Formant: Down to FormantTier: trivial copying of frames to points.

Conversion:

- FormantTier: Down to PointProcess: copy times.

Synthesis:

- Sound & FormantTier: Filter: see Source-filter synthesis.

Queries:

- Get low index from time...
- Get high index from time...
- Get nearest index from time...

Modification:

- Remove point...
- Remove point near...
- Remove points between...
- FormantTier: Add point...

FormantTier attributes

With Inspect, you will see the following attribute:

points

when you open this, you see the *size* (the number of points) and a series of points (*item [i], i = 1...n*).

Attributes of a formant point

Each point contains the following attributes:

time

the time associated with this formant target, in seconds. Also used to sort the targets.

numberOfFormants

the number of formant/bandwidth pairs (never more than 10).

formant [0..*numberOfFormants*-1]

the formant frequencies in Hertz. **Caution:** the frequency of the *ith* formant is in *formant [i-1]*!

bandwidth [0..*numberOfFormants*-1]

the formant bandwidths in Hertz. **Caution:** the bandwidth of the *ith* formant is in *bandwidth [i-1]*!

Links to this page

- [Sound & FormantTier: Filter \(no scale\)](#)
- [What's new?](#)

© ppgb, March 16, 2003

Formant: Down to FormantTier

A command for converting each selected Formant object into a FormantTier object.

The resulting FormantTier contains a point for each original frame. The number of formants in the result is limited to 10. The intensity information is lost.

Links to this page

- [Source-filter synthesis](#)

© *ppgb*, January 1, 1998

Formant: Formula (frequencies)...

A command to modify each selected Formant object with a specified formula.

For what you can do with formulas, see [Matrix: Formula...](#) The i th row contains the values of the i th frequency contour.

For an example, see [Source-filter synthesis](#).

Links to this page

- [Formant: Formula \(bandwidths\)...](#)
- [What's new?](#)

© *ppgb*, December 21, 1998

Formant: Formula (bandwidths)...

A command to modify each selected Formant object with a specified formula.

For what you can do with formulas, see [Matrix: Formula...](#) The i th row contains the values of the i th bandwidth contour.

See [Formant: Formula \(frequencies\)...](#) for more information.

Links to this page

- [Source-filter synthesis](#)

© *ppgb*, March 23, 1998

FormantTier: Add point...

A command to add a point to each selected FormantTier.

For examples, see Source-filter synthesis.

Arguments

Time (s)

the time at which a point is to be added.

Frequencies and bandwidths (Hz)

the frequency and bandwidth values of the requested new point. To get three formants at 500, 1500, and 2500 Hz with bandwidths of 50, 100, and 150 Hz, respectively, you specify "500 50 1500 100 2500 150".

Behaviour

The tier is modified so that it contains the new point. If a point at the specified time was already present in the tier, nothing happens.

Links to this page

- [Create FormantTier...](#)

© ppgb, December 21, 1998

Remove point...

A command to remove one point from every selected time-based tier object (DurationTier, FormantTier, IntensityTier, PitchTier, TextTier).

Argument

Point number

the index of the point you want to remove.

Behaviour

If *point number* is 3, the third point counted from the start of the tier (if it exists) is removed from the tier.

Links to this page

- [Source-filter synthesis](#)
-

© ppgb, February 16, 2003

Remove point near...

A command to remove one point from every selected time-based tier object (DurationTier, FormantTier, IntensityTier, PitchTier, TextTier).

Argument

Time (s)

the time near which you want to remove a point.

Behaviour

The point nearest to *time* (if there is any point) is removed from the tier.

Links to this page

- [Source-filter synthesis](#)
-

© *ppgb*, February 16, 2003

Remove points between...

A command to remove some points from every selected time-based tier object (DurationTier, FormantTier, IntensityTier, PitchTier, TextTier).

Arguments

From time (s), To time (s)
the times between which you want to remove all points.

Behaviour

Any points between *tmin* and *tmax* (inclusive) are removed from the tier.

Links to this page

- [Source-filter synthesis](#)
-

© ppgb, February 16, 2003

Create FormantTier...

A command in the New menu to create an empty FormantTier object.

The resulting object will have the specified name and time domain, but contain no formant points. To add some points to it, use FormantTier: Add point...

For an example, see Source-filter synthesis.

© *ppgb*, December 4, 2002

PointProcess

One of the types of objects in PRAAT.

A PointProcess object represents a *point process*, which is a sequence of *points* t_i in time, defined on a domain $[t_{min}, t_{max}]$. The index i runs from 1 to the number of points. The points are sorted by time, i.e. $t_{i+1} > t_i$.

PointProcess commands

Creation from scratch:

- Create empty PointProcess...
- Create Poisson process...

Creation of a pulse train from a pitch contour:

- PitchTier: To PointProcess: area-1 along entire time domain.
- Pitch: To PointProcess: same, but excludes voiceless intervals.
- Sound & Pitch: To PointProcess (cc): "pitch-synchronous": near locations of high amplitude.
- Sound & Pitch: To PointProcess (peaks)...: "pitch-synchronous": near locations of high amplitude.
- Sound: To PointProcess (periodic, cc)...: near locations of high amplitude.
- Sound: To PointProcess (periodic, peaks)...: near locations of high amplitude.

Creation from converting another object:

- **Matrix: To PointProcess**
- TextTier: Down to PointProcess
- PitchTier: Down to PointProcess
- IntensityTier: Down to PointProcess

Hearing:

- PointProcess: Play: pulse train.
- PointProcess: Hum: pulse train with formants.

Drawing:

- PointProcess: Draw...

Editing:

- **PointProcess: Edit**: invokes a PointEditor.
- **PointProcess & Sound: Edit**: invokes a PointEditor.
- Inside a ManipulationEditor.

Queries:

- PointProcess: Get jitter (local)...: periodic jitter.
- PointProcess: Get jitter (local, absolute)...: periodic jitter.
- PointProcess: Get jitter (rap)...: periodic jitter.
- PointProcess: Get jitter (ppq5)...: periodic jitter.
- PointProcess: Get jitter (ddp)...: periodic jitter.
- PointProcess: Get low index...: index of nearest point not after specified time.
- PointProcess: Get high index...: index of nearest point not before specified time.
- PointProcess: Get nearest index...: index of point nearest to specified time.
- PointProcess: Get interval...: duration of interval around specified time.

Set calculations:

- PointProcesses: Union: the union of two point processes.
- PointProcesses: Intersection: the intersection of two point processes.
- PointProcesses: Difference: the difference of two point processes.

Modification:

- PointProcess: Add point...: at a specified time.
- PointProcess: Remove point...: at specified index.
- PointProcess: Remove point near...: near specified time.
- PointProcess: Remove points...: between specified indices.
- PointProcess: Remove points between...: between specified times.

Analysis:

- PointProcess: To PitchTier...: pitch values in interval centres.
- **PointProcess & Sound: To Manipulation**

Synthesis:

- PointProcess: To Sound (pulse train)...
- PointProcess: To Sound (hum)...

Conversion:

- **PointProcess: To Matrix**
- PointProcess: Up to TextTier...
- PointProcess: Up to PitchTier...
- PointProcess: Up to IntensityTier...

Links to this page

- [FormantTier: Down to PointProcess](#)
 - [Formulas 7. Attributes of objects](#)
 - [Manipulation](#)
 - [Manipulation: Extract pulses](#)
 - [Manipulation: Replace pulses](#)
 - [Play](#)
 - [PointProcess: To TextGrid \(vuv\)...](#)
 - [PointProcess: To TextGrid...](#)
 - [Source-filter synthesis](#)
 - [time domain](#)
 - [What's new?](#)
-

© *ppgb*, May 21, 2003

PitchTier: To PointProcess

A command that uses a PitchTier object to generate a PointProcess.

Purpose

to interpret an acoustic periodicity contour as the frequency of an underlying point process (such as the sequence of glottal closures in vocal-fold vibration).

Algorithm

Points are generated along the entire time domain of the **PitchTier**, because there is no voiced/unvoiced information. The area between two adjacent points under the linearly interpolated pitch contour, is always 1.

Links to this page

- [Pitch: To PointProcess](#)
- [PSOLA](#)
- [Source-filter synthesis](#)

© *ppgb*, September 15, 1996

PointProcess: Remove points between...

A command to remove a range of points from every selected PointProcess.

Arguments

From time (seconds)

the start of the domain from which all points are to be removed.

To time (seconds)

the end of the domain from which all points are to be removed.

Behaviour

All points that originally fell in the domain [*fromTime*, *toTime*], including the edges, are removed, and the other points stay the same.

Links to this page

- [Source-filter synthesis](#)

© ppgb, December 12, 2002

PointProcess: To Sound (pulse train)...

A command to convert every selected PointProcess into a Sound.

Algorithm

A pulse is generated at every point in the point process. This pulse is filtered at the Nyquist frequency of the resulting **Sound** by converting it into a sampled **sinc** function.

Arguments

Sampling frequency

the sampling frequency of the resulting Sound object, e.g. 22050 Hertz.

Adaptation factor

the factor by which a pulse height will be multiplied if the pulse time is not within *adaptationTime* from the pre-previous pulse, and by which a pulse height will again be multiplied if the pulse time is not within *adaptationTime* from the previous pulse. This factor is against abrupt starts of the pulse train after silences, and is 1.0 if you do want abrupt starts after silences.

Adaptation time

the minimal period that will be considered a silence, e.g. 0.05 seconds.

Interpolation depth

the extent of the sinc function to the left and to the right of the peak, e.g. 2000 samples.

Example: if *adaptationFactor* is 0.6, and *adaptationTime* is 0.02 s, then the heights of the first two pulses after silences of at least 20 ms will be multiplied by 0.36 and 0.6, respectively.

Links to this page

- [PointProcess: Play](#)
- [PointProcess: To Sound \(hum\)...](#)
- [Source-filter synthesis](#)

© ppgb, March 31, 2004

Sound: Formula...

A command for changing the data in all selected Sound objects.

See the Formulas tutorial for examples and explanations.

Links to this page

- [Source-filter synthesis](#)

© *ppgb*, December 6, 2002

Sound & Formant: Filter (no scale)

A command to create a new Sound from the selected Sound and Formant objects.

For examples, see Source-filter synthesis.

Unlike what happens in Sound & Formant: Filter, the resulting Sound is not scaled. This allows generation of a series of signals with controlled relative intensities.

© *ppgb*, November 19, 1999

Sound & FormantTier: Filter (no scale)

A command to create a new Sound from the selected Sound and FormantTier objects.

For examples, see Source-filter synthesis.

Unlike what happens in Sound & FormantTier: Filter, the resulting Sound is not scaled. This allows generation of a series of signals with controlled relative intensities.

© *ppgb*, November 19, 1999

Boersma (1998)

Paul Boersma (1998): *Functional Phonology* [LOT International Series 11]. The Hague: Holland Academic Graphics. Pages i-ix, 1-493. [Doctoral thesis, University of Amsterdam]

This book can be downloaded as a PDF file from <http://www.fon.hum.uva.nl/paul/>, where you can also find many Praat scripts for the simulations and pictures in this book. A paperback version is also available from the author (paul.boersma@hum.uva.nl).

Links to this page

- Articulatory synthesis
- Boersma (1997a)
- Boersma (1997b)
- Boersma (2000)
- OT learning 1. Kinds of OT grammars
- OT learning 2.4. Evaluation
- OT learning 3.2. Data from another grammar
- OT learning 6. Shortcut to OT learning
- OTGrammar & 2 Strings: Learn...
- OTGrammar: Learn one...
- phonToDifferenceLimens

© ppgb, October 17, 2000

Speaker

One of the types of objects in PRAAT. See Articulatory synthesis.

Speaker commands

- Create Speaker...
 - Artword & Speaker: To Sound...: articulatory synthesis
-

© *ppgb*, February 1, 1998

Artword

One of the types of objects in PRAAT. See Articulatory synthesis.

An object of class Artword object represents the activities of several speech muscles as functions of time.

Artword commands

- Create Artword...: creates an Artword with relaxed muscles
- Artword & Speaker: To Sound...: articulatory synthesis

© *ppgb*, March 16, 2003

Create Speaker...

A command to create a Speaker object. See Articulatory synthesis.

Arguments

Name

the name that you give to the created object. The standard name is "artword", but you should give it a more sensible name, possibly something that represents the utterance that it is supposed to generate.

Duration (seconds)

the duration of the resulting Artword. Should be as long as the utterance that you want to generate with it. The standard value is 1 second.

© ppgb, September 16, 2003

Create Artword...

A command to create an Artword object with all muscle activities set to zero. See Articulatory synthesis.

Arguments

Name

the name that you give to the created object. The standard name is "speaker", but if you work with multiple Speaker objects, give them sensible names to reduce confusion.

© ppgb, September 16, 2003

Artword & Speaker: To Sound...

A command to synthesize a Sound object from the selected Speaker and the selected Artword.

This is the command that performs the actual articulatory synthesis. See Articulatory synthesis.

Arguments

Sampling frequency (Hz)

the number of times per second that the equilibrium widths and lengths and the tensions of the muscles are recomputed from the Artword. This will also be the sampling frequency of the resulting sound and of the optional resulting tube widths, air pressures, and air velocities. The standard value is 22050 Hz.

Oversampling

the number of times that the aerodynamic quantities and the state of the tube walls will be recomputed during each sample period. The standard value is 25.

Width 1, Width 2, Width 3

the numbers (see below) of the tubes whose widths you want to monitor. E.g., if *Width 1* is 36, the synthesizer will create a Sound object named "width36", which contains the width of tube 36 (the lower glottis) as a function of time, expressed in metres. To prevent the creation of a "width" object, specify "0" (the standard value).

Pressure 1, Pressure 2, Pressure 3

the numbers (see below) of the tubes whose air pressures you want to monitor. E.g., if *Pressure 3* is 37, the synthesizer will create a Sound object named "pressure37", which contains the air pressure of tube 37 (the upper glottis) as a function of time, expressed in Pascal. To prevent the creation of a "pressure" object, specify "0" (the standard value).

Velocity 1, Velocity 2, Velocity 3

the numbers (see below) of the tubes whose air velocities you want to monitor. E.g., if *Velocity 1* is 60, the synthesizer will create a Sound object named "velocity60", which contains the air velocity of tube 60 (in the mouth) as a function of time, expressed in metres per second. To prevent the creation of a "velocity" object, specify "0" (the standard value).

Stability

The internal sampling frequency for the aerodynamics is the specified *sampling rate*, multiplied by the specified *oversampling*. With the standard settings, this is 22050 times 25 = 550750 Hz.

To ensure the stability of the synthesis, this internal sampling frequency should not be less than the velocity of sound (353 m/s) divided by the length of the shortest tube. For the standard "Female", "Male", and "Child" speakers, the shortest tube is the upper glottis, which has a length of 0.7, 1.0, and 0.3 millimetres, respectively. The minimum internal sampling frequencies, therefore, are 504286, 353000, and 1176667 Hertz, respectively.

Time resolution

To capture the microscopic pressure changes in the glottis, you will want maximum time resolution. For a female speaker, you could set *sampling frequency* to 550750 Hz, and *oversampling* to 1.

Tube numbers

Here are the tube numbers that you can use for the *width*, *pressure*, and *velocity* arguments:

- 1..23: lungs (from bottom to top)
- 24..29: bronchi (from bottom to top)
- 30..35: trachea (from bottom to top)
- 36: lower glottis
- 37: upper glottis (not for a one-mass model)
- 38..49: pharynx (from bottom to top)
- 50..51: nasopharyngeal branching
- 52..64: mouth (from back to front)
- 65..78: nose (from back to front)
- 79..86: conus elasticus (only for a 10-mass model)
- 87..89: glottal shunt between the arytenoids (from bottom to top)

Some structural properties:

- Tube 1 is closed at the bottom.
- Tubes 64 (lips) and 78 (nostrils) radiate into the air.
- The nasopharyngeal branch is at tubes 50, 51, and 65. They are constrained to have equal lengths.
- For a one-mass model of the vocal cords, tube 36 is connected to 38.
- For a 10-mass model, tubes 32..35 are replaced with 79..86, so that tube 31 is connected to 79, and 86 is connected to 36.
- A glottal shunt will be implemented if the speaker's "shunt.Dx" attribute is not zero. A branch is then made from tubes 34 and 35 (or 85 and 86) to 87, and from tube 89 to 38 and 39.

© ppgb, March 31, 2004

OT learning 1. Kinds of OT grammars

This is chapter 1 of the OT learning tutorial.

According to Prince & Smolensky (1993), an Optimality-Theoretic (OT) grammar consists of a number of ranked constraints. For every possible input (usually an underlying form), GEN (the generator) generates a (possibly very large) number of *output candidates*, and the ranking order of the constraints determines the winning candidate, which becomes the single optimal output.

In OT, ranking is *strict*, i.e., if a constraint *A* is ranked higher than the constraints *B*, *C*, and *D*, a candidate that violates only constraint *A* will always be beaten by any candidate that respects *A* (and any higher constraints), even if it violates *B*, *C*, and *D*.

Ordinal OT grammars

Because only the ranking order of the constraints plays a role in evaluating the output candidates, Prince & Smolensky took the grammar to contain no absolute ranking values, i.e., they accepted only an ordinal relation between the constraint rankings. For such a grammar, Tesar & Smolensky (1998) devised a learning algorithm (Error-Driven Constraint Demotion, EDCD) that changes the ranking order whenever the form produced by the learner is different from the adult form. Such a learning step can sometimes lead to a large change in the behaviour of the grammar.

Stochastic OT grammars

The EDCD algorithm is fast and convergent. As a model of language acquisition, however, its drawbacks are that it is extremely sensitive to errors in the learning data and that it does not show realistic gradual learning curves. For these reasons, Boersma (1997b) / Boersma (1998) / Boersma (2000) proposed stochastic constraint grammars in which every constraint has a *ranking value* along a continuous ranking scale, and a small amount of *noise* is added to this ranking value at evaluation time. The associated error-driven learning algorithm (Gradual Learning Algorithm, GLA) effects small changes in the ranking values of the constraints with every learning step. An added virtue of the GLA is that it can learn languages with optionality and variation, which was something that EDCD could not do.

Ordinal OT grammars can be seen as a special case of the more general stochastic OT grammars: they have integer ranking values (*strata*) and zero evaluation noise. In PRAAT, therefore, every constraint is taken to have a ranking value, so that you can do stochastic as well as ordinal OT.

An OT grammar is implemented as an OTGrammar or OTAnyGrammar object. In an OTGrammar object, you specify all the constraints, all the possible inputs and all their possible outputs. This makes the OTGrammar object the simplest of the two for most problems. In this tutorial, we will only discuss OTGrammar objects.

© ppgb, November 5, 2002

OTGrammar

One of the types of objects in PRAAT. See the OT learning tutorial.

Inside an OTGrammar

With Inspect, you will see the following attributes:

constraints

a list of constraints. Each constraint contains the following attributes:

name

the fixed name of the constraint, for instance "PARSE".

ranking

the continuous ranking value; will change during learning.

disharmony

the effective ranking value during stochastic evaluation; with a non-zero evaluation noise, this will be different from *ranking*.

fixedRankings

an often empty list of locally ranked pairs of constraints. Each local-ranking pair contains the following attributes:

higher

the index of the universally higher-ranked of the two constraints, a number between 1 and the number of constraints.

lower

the index of the universally lower-ranked of the two constraints.

tableaus

a list of tableaus. Each tableau contains the following attributes:

input

the input string of the tableau. For production grammars, the underlying form of the utterance, for example [IMAGE] an+pa[IMAGE] or [IMAGE] b[IMAGE] [IMAGE] [IMAGE] + PAST[IMAGE] .

candidates

a list of output candidates. Each output candidate consists of:

output

the output string of the tableau. In generative phonology: the surface form of the utterance, for example [anpa] or [ampa] or [b[IMAGE] [IMAGE] :t] or [b[IMAGE] æ[IMAGE]]. In functional phonology: the combination of the articulatory and the perceptual results, for example [anpa]-/anpa/ or [ampa]-/ampa/ or [b[IMAGE] [IMAGE] :t]-/b[IMAGE] [IMAGE] :t/ or [b[IMAGE] æ[IMAGE]]-/b[IMAGE] æ[IMAGE] /.

marks

a list of the number of violations of each constraint for this output form. If there are 13 constraints, this list will contain 13 integer numbers for each candidate.

For a more computational approach to OT grammars, see the `OTAnyGrammar` class.

OTGrammar creation

You can easily create some **OTGrammar** examples from the New menu, or type your own grammars into a text file and read the file into Praat. See the OT learning tutorial.

OTGrammar actions

You can perform the following actions on selected **OTGrammar** objects:

- OTGrammar: Generate inputs...
- OTGrammar: Input to output...
- OTGrammar: Input to outputs... (compute the output distribution for a given input)
- OTGrammar: To output Distributions...
- OTGrammar & Strings: Inputs to outputs...
- OTGrammar: Learn one...
- OTGrammar & 2 Strings: Learn...

You can view an **OTGrammar** in an `OTGrammarEditor`.

Links to this page

- [Create tongue-root grammar...](#)
- [OT learning 1. Kinds of OT grammars](#)
- [OT learning 2.1. Viewing a grammar](#)
- [OT learning 2.2. Inside the grammar](#)
- [OT learning 2.7. Tableau pictures](#)
- [OT learning 2.9. Output distributions](#)
- [OT learning 3.1. Data from a pair distribution](#)
- [OT learning 3.2. Data from another grammar](#)
- [OT learning 4. Learning an ordinal grammar](#)
- [OT learning 6. Shortcut to OT learning](#)
- [OT learning 7. Learning from overt forms](#)
- [OTAnyGrammar examples](#)
- [Robust Interpretive Parsing](#)

© *ppgb*, March 16, 2003

OT learning 2. The grammar

This is chapter 2 of the OT learning tutorial.

We can ask the grammar to produce an output form for any input form that is in its list of tableaux.

- 2.1. Viewing a grammar (NOCODA example, OTGrammarEditor)
- 2.2. Inside the grammar (saving, inspecting)
- 2.3. Defining your own grammar
- 2.4. Evaluation (noise)
- 2.5. Editing a grammar
- 2.6. Variable output (place assimilation example)
- 2.7. Tableau pictures (printing, EPS files)
- 2.8. Asking for one output
- 2.9. Output distributions

Links to this page

- [OT learning 3. Generating language data](#)
-

© *ppgb*, January 22, 2000

OT learning 2.1. Viewing a grammar

Consider a language where the underlying form /pat/ leads to the surface form [pa], presumably because the structural constraint NOCODA outranks the faithfulness constraint PARSE.

To create such a grammar in PRAAT, choose **Create NoCoda grammar** from the Optimality Theory submenu of the New menu. An OTGrammar object will then appear in the list of objects. If you click Edit, an OTGrammarEditor will show up, containing:

1. the constraint list, sorted by *disharmony* (= ranking value + noise):

*ranking value*disharmony

NOCODA100.000100.000

PARSE90.00090.000

2. the tableaux for the two possible inputs /pat/ and /pa/:

[sorry, no pictures yet in the web version of this manual]

[sorry, no pictures yet in the web version of this manual]

From the first tableau, we see that the underlying form /pat/ will surface as [pa], because the alternative [pat] violates a constraint (namely, NOCODA) with a higher disharmony than does [pa], which only violates PARSE, which has a lower disharmony.

Note the standard OT tableau layout: asterisks (*) showing violations, exclamation marks (!) showing crucial violations, greying of cells that do not contribute to determining the winning candidate, and a finger () pointing to the winner (this may look like a plus sign (+) if you don't have the Zapf Dingbats font installed on your computer or printer).

The second tableau shows that /pa/ always surfaces as [pa], which is no wonder since this is the only candidate. All cells are grey because none of them contributes to the determination of the winner.

Links to this page

- [OT learning](#)
- [OT learning 2. The grammar](#)

© *ppgb*, December 4, 2002

OTGrammarEditor

One of the editors in PRAAT, for viewing and editing the grammar in an OTGrammar object.

See the OT learning tutorial for examples.

Usage

The menu command that you will probably use most often if you investigate variation, is the **Evaluate (noise 2.0)** command, which you can invoke from the Edit menu or by pressing Command-2.

This command performs a new evaluation with the current ranking values. Some noise is added to the ranking values, so that the *disharmonies* of the constraint will change. This may cause a change in the ranking order of the constraints, which in its turn may cause a different candidate to win in some tableaux.

Links to this page

- [OT learning 2. The grammar](#)
- [OT learning 2.1. Viewing a grammar](#)
- [OT learning 2.8. Asking for one output](#)
- [OT learning 3.2. Data from another grammar](#)
- [Types of objects](#)

© *ppgb*, March 16, 2003

OT learning 2.2. Inside the grammar

You can write an OTGrammar grammar into a text file by choosing Write to text file... from the Write menu of the Objects window. For the NOCODA example, the contents of the file will look like:

```
File type = "ooTextFile"
Object class = "OTGrammar"

2 constraints
constraint [1]: "N\s{O}C\s{ODA}" 100 100 ! NOCODA
constraint [2]: "P\s{ARSE}" 90 90 ! PARSE

0 fixed rankings

2 tableaux
input [1]: "pat" 2
  candidate [1]: "pa" 0 1
  candidate [2]: "pat" 1 0
input [2]: "pa" 1
  candidate [1]: "pa" 0 0
```

To understand more about this data structure, consult the OTGrammar class description or click **Inspect** after selecting the OTGrammar object. The "`\s{...}`" braces ensure that the constraint names show up with their traditional small capitals (see Text styles).

You can read this text file into Praat again with Read from file... from the Read menu in the Objects window.

Links to this page

- OT learning
- OT learning 2. The grammar

© ppgb, December 30, 1998

OT learning 2.3. Defining your own grammar

By editing a text file created from an example in the New menu, you can define your own OT grammars.

As explained at Write to text file..., Praat is quite resilient about its text file formats. As long as the strings and numbers appear in the correct order, you can redistribute the data across the lines, add all kinds of comments, or leave the comments out. For the NOCODA example, the text file could also have looked like:

```
"ooTextFile"  
"OTGrammar"  
2  
"N\s{O}C\s{ODA}" 100 100  
"P\s{ARSE}" 90 90  
0 ! number of fixed rankings  
2 ! number of accepted inputs  
"pat" 2 ! input form with number of output candidates  
  "pa" 0 1 ! first candidate with violations  
  "pat" 1 0 ! second candidate with violations  
"pa" 1 ! input form with number of candidates  
  "pa" 0 0
```

To define your own grammar, you just provide a number of constraints and their rankings, and all the possible input forms with all their output candidates, and all the constraint violations for each candidate. The order in which you specify the constraints is free (you don't have to specify the highest-ranked first), as long as the violations are in the same order; you could also have reversed the order of the two input forms, as long as the corresponding candidates follow them; and, you could also have reversed the order of the candidates within the /pat/ tableau, as long as the violations follow the output forms. Thus, you could just as well have written:

```
"ooTextFile"  
"OTGrammar"  
2  
"P\s{ARSE}" 90 90  
"N\s{O}C\s{ODA}" 100 100  
0  
2  
"pa" 1  
  "pa" 0 0  
"pat" 2  
  "pat" 0 1  
  "pa" 1 0
```

Links to this page

- [OT learning](#)
- [OT learning 2. The grammar](#)

© *ppgb*, December 4, 2002

OT learning 2.4. Evaluation

In an Optimality-Theoretic model of grammar, *evaluation* refers to the determination of the winning candidate on the basis of the constraint ranking.

In an ordinal OT model of grammar, repeated evaluations will yield the same winner again and again. We can simulate this behaviour with our NOCODA example. In the editor, you can choose **Evaluate (zero noise)** or use its keyboard shortcut Command-0 (= Command-zero). Repeated evaluations (keep Command-0 pressed) will always yield the following grammar:

ranking valuedisharmony

NOCODA100.000100.000

PARSE90.00090.000

In a stochastic OT model of grammar, repeated evaluations will yield different disharmonies each time. To see this, choose **Evaluate (noise 2.0)** or use its keyboard shortcut Command-2. Repeated evaluations will yield grammars like the following:

ranking valuedisharmony

NOCODA100.000100.427

PARSE90.00087.502

and

ranking valuedisharmony

NOCODA100.000101.041

PARSE90.00090.930

and

ranking valuedisharmony

NOCODA100.00096.398

PARSE90.00089.482

The disharmonies vary around the ranking values, according to a Gaussian distribution with a standard deviation of 2.0. The winner will still be [pa] in almost all cases, because the probability of bridging the gap between the two ranking values is very low, namely 0.02 per cent according to Boersma (1998), page 332.

With a noise much higher than 2.0, the chances of PARSE outranking NOCODA will rise. To see this, choose **Evaluate...** and supply 5.0 for the noise. Typical outcomes are:

ranking valuedisharmony

NOCODA100.00092.634

PARSE90.00086.931

and

ranking valuedisharmony

NOCODA100.000101.162

PARSE90.00085.311

and

ranking valuedisharmony

PARSE90.00099.778

NOCODA100.00098.711

In the last case, the order of the constraints has been reversed. You will see that [pat] has become the winning candidate:

[sorry, no pictures yet in the web version of this manual]

However, in the remaining part of this tutorial, we will stick with a noise with a standard deviation of 2.0. This specific number ensures that we can model fairly rigid rankings by giving the constraints a ranking difference of 10, a nice round number. Also, the learning algorithm will separate many constraints in such a way that the differences between their ranking values are in the vicinity of 10.

Links to this page

- [OT learning](#)
- [OT learning 2. The grammar](#)
- [OTGrammar: Input to output...](#)
- [OTGrammar: Input to outputs...](#)

© *ppgb*, November 5, 2002

OT learning 2.5. Editing a grammar

In the NOCODA example, the winning candidate for the input /pat/ was always [pa].

To make [pat] the winner instead, NOCODA should be ranked lower than PARSE. To achieve this even with zero noise, go to the editor and select the NOCODA constraint by clicking on it (a spade symbol \sp will mark the selected constraint), and choose **Edit ranking...** from the Edit menu, or use the keyboard shortcut Command-E.

In the resulting dialog, we lower the ranking of the constraint from 100 to 80, and click OK. This is what you will see in the editor:

ranking valuedisharmony

\sp **NOCODA**80.000103.429

PARSE90.00088.083

[sorry, no pictures yet in the web version of this manual]

Nothing has happened to the tableau, because the disharmonies still have their old values. So choose **Evaluate (noise 2.0)** (Command-2) or **Evaluate (zero noise)** (Command-0). The new disharmonies will centre around the new ranking values, and we see that [pat] becomes the new winner:

ranking valuedisharmony

PARSE90.00090.743

NOCODA80.00081.581

[sorry, no pictures yet in the web version of this manual]

- OT learning
 - OT learning 2. The grammar
-

© *ppgb*, February 2, 2000

OT learning 2.6. Variable output

Each time you press Command-2, which invokes the command **Evaluate (noise 2.0)** from the Edit menu, you will see the disharmonies changing. If the distance between the constraint rankings is 10, however, the winning candidates will very probably stay the same.

So starting from the NOCODA example, we edit the rankings of the constraints again, setting the ranking value of PARSE to 88 and that of NOCODA to 85. If we now press Command-2 repeatedly, we will get [pat] in most of the cases, but we will see the finger pointing at [pa] in 14 percent of the cases:

ranking valuedisharmony

PARSE88.00087.421

NOCODA85.00085.585

[sorry, no pictures yet in the web version of this manual]

but

ranking valuedisharmony

NOCODA85.00087.128

PARSE88.00085.076

[sorry, no pictures yet in the web version of this manual]

As a more functionally oriented example, we consider nasal place assimilation. Suppose that the underlying sequence /an+pa/ surfaces as the assimilated [ampa] in 80 percent of the cases, and as the faithful [anpa] in the remaining 20 percent, while the non-nasal stop /t/ never assimilates. This can be achieved by having the articulatory constraint *GESTURE ranked at a short distance above *REPLACE (n, m):

```

"ooTextFile"
"OTGrammar"
3 constraints
"*G\{ESTURE}" 102.7 0
"*R\{EPLACE} (n, m)" 100.0 0
"*R\{EPLACE} (t, p)" 112.0 0
0 fixed rankings
2 tableaux
"an+pa" 2
  "anpa" 1 0 0
  "ampa" 0 1 0
"at+ma" 2
  "atma" 1 0 0
  "apma" 0 0 1

```

You can create this grammar with **Create place assimilation grammar** from the New menu. In the editor, it will often look like follows:

ranking valuedisharmony

***REPLACE (t, p)**112.000109.806

***GESTURE**102.700102.742

***REPLACE (n, m)**100.000101.044

[sorry, no pictures yet in the web version of this manual]

[sorry, no pictures yet in the web version of this manual]

If you keep the Command-2 keys pressed, however, you will see that the tableaux change into something like the following in approximately 20 percent of the cases:

ranking valuedisharmony

***REPLACE (t, p)**112.000113.395

***REPLACE (n, m)**100.000103.324

***GESTURE**102.700101.722

[sorry, no pictures yet in the web version of this manual]

[sorry, no pictures yet in the web version of this manual]

We see that /at+ma/ always surfaces at [atma], because *REPLACE (t, p) is ranked much higher than the other two, and that the output of /an+pa/ is variable because of the close rankings of *GESTURE and *REPLACE (n, m).

Links to this page

- [OT learning](#)
- [OT learning 2. The grammar](#)
- [OT learning 2.9. Output distributions](#)
- [OT learning 5. Learning a stochastic grammar](#)

© *ppgb*, December 4, 2002

OT learning 2.7. Tableau pictures

To show a tableau in the Picture window instead of in the editor, you select an OTGrammar object and click **Draw tableau...** After you specify the input form, a tableau is drawn with the current font and size at the location of the current selection (*viewport*) in the Picture window. The top left corner of the tableau is aligned with the top left corner of the selection. You can draw more than one object into the Picture window, whose menus also allow you to add a lot of graphics of your own design.

Besides printing the entire picture (with Print...), you can save a part of it to an EPS file for inclusion into your favourite word processor (with Write to EPS file...). For the latter to succeed, make sure that the selection includes at least your tableau; otherwise, some part of your tableau may end up truncated.

Links to this page

- [OT learning](#)
- [OT learning 2. The grammar](#)

© *ppgb*, October 10, 2000

OT learning 2.8. Asking for one output

To ask the grammar to produce a single output for a specified input form, you can choose OTGrammar: Input to output.... The dialog will ask you to provide an input form and the strength of the noise (the standard value is 2.0 again). This will perform an evaluation and write the result into the Info window.

If you are viewing the grammar in the OTGrammarEditor, you will see the disharmonies change, and if the grammar allows variation, you will see that the winner in the tableau in the editor varies with the winner shown in the Info window.

Since the editor shows more information than the Info window, this command is not very useful except for purposes of scripting. See the following page for some related but more useful commands.

Links to this page

- [OT learning](#)
- [OT learning 2. The grammar](#)

© *ppgb*, September 16, 2003

OT learning 2.9. Output distributions

To ask the grammar to produce *many* outputs for a specified input form, and collect them in a Strings object, you select an OTGrammar and choose Input to outputs....

For example, select the object "OTGrammar assimilation" from our place assimilation example (\SS2.6), and click **Input to outputs....** In the resulting dialog, you specify 1000 trials, a noise strength of 2.0, and "an+pa" for the input form.

After you click OK, a Strings object will appear in the list. If you click Info, you will see that it contains 1000 strings. If you click Inspect, you will see that most of the strings are "ampa", but some of them are "anpa". These are the output forms computed from 1000 evaluations for the input /an+pa/.

To count how many instances of [ampa] and [anpa] were generated, you select the Strings object and click To Distributions. You will see a new Distributions object appear in the list. If you draw this to the Picture window (with **Draw as numbers...**), you will see something like:

ampa815

anpa185

which means that our grammar, when fed with 1000 /an+pa/ inputs, produced [ampa] 815 times, and [anpa] 185 times, which is consistent with our initial guess that a ranking difference of 2.7 would cause approximately an 80% - 20% distribution of [ampa] and [anpa].

Checking the distribution hypothesis

To see whether the guess of a 2.7 ranking difference is correct, we perform 1,000,000 trials instead of 1000. The output distribution (if you have enough memory in your computer) becomes something like (set the *Precision* to 7 in the drawing dialog):

ampa830080

anpa169920

The expected values under the 80% - 20% distribution hypothesis are:

ampa800000

anpa200000

We compute (e.g. with Calculator...) a χ^2 of $30080^2/800000 + 30080^2/200000 = 5655.04$, which, of course, is much too high for a distribution with a single degree of freedom. So the ranking difference must be smaller. If it is 2.4 (change the ranking of *GESTURE to 102.4), the numbers become something like

ampa801974

anpa198026

which gives a χ^2 of 24.35. By using the Calculator with the formula `chiSquareQ (24.35, 1)`, we find that values larger than this have a probability of $8 \cdot 10^{-7}$ under the 80% - 20% distribution hypothesis, which must therefore be rejected again.

Rather than continuing this iterative procedure to find the correct ranking values for an 80% - 20% grammar, we will use the Gradual Learning Algorithm (\SS5) to determine the rankings automatically, without any memory of past events other than the memory associated with maintaining the ranking values.

Measuring all inputs

To measure the outcomes of all the possible inputs at once, you select an OTGrammar and choose To output Distributions.... As an example, try this on our place assimilation grammar. You can supply 1000000 for the number of trials, and the usual 2.0 for the standard deviation of the noise. After you click OK, a Distributions object will appear in the list. If you draw this to the Picture window, the result will look like:

/an+pa/ -> anpa169855

/an+pa/ -> ampa830145

/at+ma/ -> atma999492

/at+ma/ -> apma508

We see that the number of [apma] outputs is not zero. This is due to the difference of 9.3 between the rankings of *REPLACE (t, p) and *GESTURE. If you rank *REPLACE (t, p) at 116.0, the number of produced [apma] reduces to about one in a million, as you can easily check with some patience.

Links to this page

- OT learning
- OT learning 2. The grammar

© ppgb, February 2, 2000

OT learning 3. Generating language data

A learner needs two things: a grammar that she can adjust (\backslash SS2), and language data.

- 3.1. Data from a pair distribution
- 3.2. Data from another grammar (tongue-root-harmony example)

Links to this page

- [OT learning](#)
-

© *ppgb*, December 4, 2002

OT learning 3.1. Data from a pair distribution

If the grammar contains faithfulness constraints, the learner needs pairs of underlying and adult surface forms. For our place assimilation example, she needs a lot of /at+ma/ - [atma] pairs, and four times as many /an+pa/ - [ampa] pairs as /an+pa/ - [anpa] pairs. We can specify this language-data distribution in a PairDistribution object, which we could simply write into a text file:

```
"ooTextFile"  
"PairDistribution"  
4 pairs  
"at+ma" "atma" 100  
"at+ma" "apma" 0  
"an+pa" "anpa" 20  
"an+pa" "ampa" 80
```

The values appear to represent percentages, but they could also have been 1.0, 0.0, 0.2, and 0.8, or any other values with the same proportions. We could also have left out the second pair and specified "3 pairs" instead of "4 pairs" in the third line.

We can create this pair distribution with **Create place assimilation distribution** from the Optimality Theory submenu of the New menu in the Objects window. To see that it really contains the above data, you can draw it to the Picture window. To change the values, use Inspect (in which case you should remember to click Change after any change).

To generate input-output pairs from the above distribution, select the PairDistribution and click To Stringses.... If you then just click OK, there will appear two Strings objects in the list, called "input" (underlying forms) and "output" (surface forms). Both contain 1000 strings. If you Inspect them both, you can see that e.g. the 377th string in "input" corresponds to the 377th string in "output", i.e., the two series of strings are aligned. See also the example at PairDistribution: To Stringses....

These two Strings objects are sufficient to help an OTGrammar grammar to change its constraint rankings in such a way that the output distributions generated by the grammar match the output distributions in the language data. See \SS5.

Links to this page

- OT learning
- OT learning 3. Generating language data

OT learning 3.2. Data from another grammar

Instead of generating input-output pairs directly from a PairDistribution object, you can also generate input forms and their winning outputs from an OTGrammar grammar. Of course, that's what the language data presented to real children comes from. Our example will be a tongue-root harmony grammar.

Choose Create tongue-root grammar... from the Optimality Theory submenu of the New menu. Set *Constraint set* to "Five", and *Ranking* to "Wolof". Click OK. An object called "OTGrammar Wolof" will appear in the list. Click **Edit**. You will see the following grammar appear in the OTGrammarEditor:

ranking valuedisharmony

***[rtr / hi]**100.000100.000

PARSE (rtr)50.00050.000

***GESTURE (contour)**30.00030.000

PARSE (atr)20.00020.000

***[atr / lo]**10.00010.000

This simplified Wolof grammar, with five constraints with clearly different rankings, is equivalent to the traditional OT ranking

***[rtr / hi] >> PARSE (rtr) >> *GESTURE (contour) >> PARSE (atr) >> *[atr / lo]**

These constraints are based on a description of Wolof by Archangeli & Pulleyblank (1994: 225--239). For the meaning of these constraints, see Boersma (1998: 295), or the Create tongue-root grammar... manual page.

For each input, there are four output candidates: the vowel heights will be the same as those in the input, but the tongue-root values of V_1 and V_2 are varied. For example, for the input [ita] we will have the four candidates [ita], [it_[IMAGE]], [_[IMAGE]ta], and [_[IMAGE]t_[IMAGE]].

With this way of generating candidates, we see that the five constraints are completely ranked. First, the absolute prohibition on surface [_[IMAGE]] shows that *[rtr / hi] outranks RTR faithfulness (otherwise, [_[IMAGE]t_[IMAGE]] would have been the winner):

[sorry, no pictures yet in the web version of this manual]

Second, the faithful surfacing of the disharmonic /itɛ/ shows that RTR faithfulness must outrank the harmony (anti-contour) constraint (otherwise, [ite] would have been the winner):

[sorry, no pictures yet in the web version of this manual]

Third, the RTR-dominant harmonicization of underlying disharmonic /etɛ/ shows that harmony must outrank ATR faithfulness (otherwise, [etɛ] would have won):

[sorry, no pictures yet in the web version of this manual]

Finally, the faithful surfacing of the low ATR vowel /_[IMAGE]/ even if not forced by harmony, shows that ATR faithfulness outranks *[atr / lo] (otherwise, [ata] would have been the winning candidate):

[sorry, no pictures yet in the web version of this manual]

These four ranking arguments clearly establish the crucial rankings of all five constraints.

Generating inputs from the grammar

According to Prince & Smolensky (1993), the input to an OT grammar can be *anything*. This is the idea of *richness of the base*. When doing a practical investigation, however, we are only interested in the inputs that will illustrate the properties of our partial grammars. In the case of simplified Wolof, this means the 36 possible V_1tV_2 sequences where V_1 and V_2 are any of the six front vowels i, _[IMAGE], e, ɛ, _[IMAGE], and a (see Create tongue-root grammar...).

A set of inputs can be generated from an OTGrammar object by inspecting the list of tableaux. So select the Wolof tongue-root grammar and choose Generate inputs.... Set *Number of trials* to 100, and click OK. A Strings object named "Wolof_in" will appear in the list. Click Inspect and examine the 100 input strings. You will see that they have been randomly chosen from the 36 possible V_1tV_2 sequences as described at Create tongue-root grammar...:

ɛ ta, etɛ, ɛ ti, itɛ, ɛ tɛ, iti, ɛ t_[IMAGE], it_[IMAGE], _[IMAGE] ti, etɛ, ...

Thus, when asked to generate a random input, these grammars produce any of the 36 possible V_1tV_2 sequences, all with equal probability.

Generating outputs from the grammar

To compute the outputs for the above set of input forms, select *both* the OTGrammar object *and* the input Strings object, and choose Inputs to outputs..., perhaps specifying zero evaluation noise. A new Strings object called "Wolof_out" will appear in the list. If you Inspect it, you will see that it contains a string sequence aligned with the original input strings:

ε ta, ε tε , ε ti, itε , ε tε , iti, ε ti, iti, iti, ε tε , ...

In this way, we have created two Strings objects, which together form a series of input-output pairs needed for learning a grammar that contains faithfulness constraints.

Links to this page

- OT learning
- OT learning 3. Generating language data
- OT learning 4. Learning an ordinal grammar

© ppgb, December 4, 2002

OT learning 4. Learning an ordinal grammar

With the data from a tongue-root-harmony language with five completely ranked constraints, we can have a throw at learning this language, starting with a grammar in which all the constraints are ranked at the same height, or randomly ranked, or with articulatory constraints outranking faithfulness constraints.

Let's try the third of these. Create an infant tongue-root grammar by choosing Create tongue-root grammar... and specifying "Five" for the constraint set and "Infant" for the ranking. The result after a single evaluation will be like:

ranking valuedisharmony

***GESTURE (contour)**100.000100.631

***[atr / lo]**100.000100.244

***[rtr / hi]**100.00097.086

PARSE (rtr)50.00051.736

PARSE (atr)50.00046.959

Such a grammar produces all kinds of non-adult results. For instance, the input /_[IMAGE] t_[IMAGE] / will surface as [at_[IMAGE]]:

[sorry, no pictures yet in the web version of this manual]

The adult form is very different: [_[IMAGE] ti]. The cause of the discrepancy is in the order of the constraints ***[atr / lo]** and ***[rtr / hi]**, which militate against [_[IMAGE]] and [_[IMAGE]], respectively. Simply reversing the rankings of these two constraints would solve the problem in this case. More generally, Tesar & Smolensky (1998) prove that demoting all the constraints that cause the adult form to lose into the stratum just below the highest-ranked constraint violated in the learner's form (here, moving ***[atr / lo]** just below ***[rtr / hi]** into the same stratum as **PARSE (rtr)**), will guarantee convergence to the target grammar,

if there is no variation in the data.

But Tesar & Smolensky's algorithm cannot be used for variable data, since all constraints would be tumbling down, exchanging places and producing wildly different grammars at each learning step. Since language data do tend to be variable, we need a gradual and balanced learning algorithm, and the following algorithm is guaranteed to converge to the target language, if that language can be described by a stochastic OT grammar.

The reaction of the learner to hearing the mismatch between the adult [ti] and her own [at], is simply:

1. to move the constraints violated in her own form, i.e. *[rtr / hi] and PARSE (atr), up by a small step along the ranking scale, thus decreasing the probability that her form will be the winner at the next evaluation of the same input;
2. and to move the constraints violated in the adult form, namely *[atr / lo] and PARSE (rtr), down along the ranking scale, thus increasing the probability that the adult form will be the learner's winner the next time.

If the small reranking step (the *plasticity*) is 0.1, the grammar will become:

ranking valuedisharmony

***GESTURE (contour)**100.000100.631

***[atr / lo]**99.900100.244

***[rtr / hi]**100.10097.086

PARSE (rtr)49.90051.736

PARSE (atr)50.10046.959

The disharmonies, of course, will be different at the next evaluation, with a probability slightly higher than 50% that *[rtr / hi] will outrank *[atr / lo]. Thus the relative rankings of these two grounding constraints have moved into the direction of the adult grammar, in which they are ranked at opposite ends of the

grammar.

Note that the relative rankings of PARSE (atr) and PARSE (rtr) are now moving in a direction opposite to where they will have to end up in this RTR-dominant language. This does not matter: the procedure will converge nevertheless.

We are now going to simulate the infant who learns simplified Wolof. Take an adult Wolof grammar and generate 1000 input strings and the corresponding 1000 output strings following the procedure described in §3.2. Now select the infant OTGrammar and both Strings objects, and choose Learn.... After you click OK, the learner processes each of the 1000 input-output pairs in succession, gradually changing the constraint ranking in case of a mismatch. The resulting grammar may look like:

ranking valuedisharmony

***[rtr / hi]**100.80098.644

***GESTURE (contour)**89.72894.774

***[atr / lo]**89.54486.442

PARSE (rtr)66.12365.010

PARSE (atr)63.55364.622

We already see some features of the target grammar, namely the top ranking of *[rtr / hi] and RTR dominance (the mutual ranking of the PARSE constraints). The steps have not been exactly 0.1, because we also specified a relative plasticity spreading of 0.1, thus giving steps typically in the range of 0.7 to 1.3.

After learning once more with the same data, the result is:

ranking valuedisharmony

***[rtr / hi]**100.800104.320

PARSE (rtr)81.42982.684

***[atr / lo]**79.96678.764

***GESTURE (contour)**81.31678.166

PARSE (atr)77.99177.875

This grammar now sometimes produces faithful disharmonic utterances, because the PARSE now often outrank the gestural constraints at evaluation time. But there is still a lot of variation produced. Learning once more with the same data gives:

ranking valuedisharmony

***[rtr / hi]**100.800100.835

PARSE (rtr)86.39282.937

***GESTURE (contour)**81.85581.018

***[atr / lo]**78.44778.457

PARSE (atr)79.40976.853

By inspecting the first column, you can see that the ranking values are already in the same order as in the target grammar, so that the learner will produce 100 percent correct adult utterances if her evaluation noise is zero. However, with a noise of 2.0, there will still be variation. For instance, the disharmonies above will produce [ata] instead of [IMAGE] t[IMAGE]] for underlying /IMAGE] t[IMAGE] /. Learning seven times more with the same data gives a reasonable proficiency:

*ranking value*disharmony

***[rtr / hi]**100.80099.167

PARSE (rtr)91.58093.388

***GESTURE (contour)**85.48786.925

PARSE (atr)80.36978.290

***[atr / lo]**75.40774.594

No input forms have error rates above 4 percent now, so the child has learned a lot with only 10,000 data, which may be on the order of the number of input data she receives every day.

We could have sped up the learning process appreciably by using a plasticity of 1.0 instead of 0.1. This would have given a comparable grammar after only 1000 data. After 10,000 data, we would have

*ranking value*disharmony

***[rtr / hi]**107.013104.362

PARSE (rtr)97.92499.984

***GESTURE (contour)**89.67989.473

PARSE (atr)81.47983.510

***[atr / lo]**73.06772.633

With this grammar, all the error rates are below 0.2 percent. We see that crucially ranked constraints will become separated after a while by a gap of about 10 along the ranking scale.

If we have three constraints obligatorily ranked as $A \gg B \gg C$ in the adult grammar, with ranking differences of 8 between A and B and between B and C in the learner's grammar (giving an error rate of 0.2%), the ranking $A \gg C$ has a chance of less than 1 in 100 million to be reversed at evaluation time. This relativity of error rates is an empirical prediction of our stochastic grammar model.

Links to this page

- [OT learning](#)
- [OTGrammar: Learn one...](#)

© *ppgb*, November 20, 2001

OT learning 5. Learning a stochastic grammar

Having shown that the algorithm can learn deep obligatory rankings, we will now see that it also performs well in replicating the variation in the language environment.

Create a place assimilation grammar as described in \SS2.6, and set all its rankings to 100.000:

ranking valuedisharmony

***GESTURE**100.000100.000

***REPLACE (t, p)**100.000100.000

***REPLACE (n, m)**100.000100.000

Create a place assimilation distribution and generate 1000 string pairs (\SS3.1). Select the grammar and the two Strings objects, and learn with a plasticity of 0.1:

ranking valuedisharmony

***REPLACE (t, p)**104.540103.140

***REPLACE (n, m)**96.21499.321

***GESTURE**99.24697.861

The output distributions are now (using OTGrammar: To output Distributions..., see \SS2.9):

/an+pa/ -> anpa14.3%

/an+pa/ -> ampa85.7%

/at+ma/ -> atma96.9%

/at+ma/ -> apma3.1%

After another 10,000 new string pairs, we have:

ranking valuedisharmony

***REPLACE (t, p)**106.764107.154

***GESTURE**97.89997.161

***REPLACE (n, m)**95.33796.848

With the following output distributions (measured with a million draws):

/an+pa/ -> anpa18.31%

/an+pa/ -> ampa81.69%

/at+ma/ -> atma99.91%

/at+ma/ -> apma0.09%

The error rate is acceptably low, but the accuracy in reproducing the 80% - 20% distribution could be better. This is because the relatively high plasticity of 0.1 can only give a coarse approximation. So we lower the plasticity to 0.001, and supply 100,000 new data:

ranking valuedisharmony

***REPLACE (t, p)**106.810107.184

***GESTURE**97.78299.682

***REPLACE (n, m)**95.40798.760

With the following output distributions:

/an+pa/ -> anpa20.08%

/an+pa/ -> ampa79.92%

/at+ma/ -> atma99.94%

/at+ma/ -> apma0.06%

So besides learning obligatory rankings like a child does, the algorithm can also replicate very well the probabilities of the environment. This means that a GLA learner can learn stochastic grammars.

Links to this page

- [OT learning](#)
 - [OTGrammar & 2 Strings: Learn...](#)
 - [OTGrammar: Learn one...](#)
-

© *ppgb*, November 20, 2001

OT learning 6. Shortcut to OT learning

Once you have mastered the tedious procedures of making Praat learn stochastic grammars, as described in the previous chapters of this tutorial, you can try a faster procedure, which simply involves selecting an OTGrammar object together with a PairDistribution object, and clicking **Learn...** Once you click OK, Praat will feed the selected grammar with input/output pairs drawn from the selected distribution, and the grammar will be modified every time its output is different from the given output. Here is the meaning of the arguments:

Evaluation noise (standard value: 2.0)

the standard deviation of the noise added to the ranking of each constraint at evaluation time.

Strategy (standard value: Symmetric all)

what to do when the learner's output is different from the given output. Possibilities:

Demotion only: lower the ranking of every constraint that is violated more in the correct output than in the learner's output. This algorithm crashes if there is variation in the data, i.e. if some inputs can have more than one possible adult outputs.

Symmetric one: lower the ranking of the highest-ranked constraint that is violated more in the adult output than in the learner's output, and raise the ranking of the highest-ranked constraint that is violated more in the learner's output than in the adult output. This is the "minimal" algorithm described and refuted in Boersma (1998), chapters 14-15.

Symmetric all: lower the ranking of all constraints that are violated more in the adult output than in the learner's output, and raise the ranking of all constraints that are violated more in the learner's output than in the adult output. This is the algorithm described in Boersma & Hayes (2001).

Weighted uncanceled: the same as "Symmetric all", but the size of the learning step is divided by the number of moving constraints. This makes sure that the average ranking of all the constraints is constant.

Weighted all: the "Symmetric all" strategy can be reworded as follows: "lower the ranking of all constraints that are violated in the adult output, and raise the ranking of all constraints that are violated in the learner's output". Do that, but divide the size of the learning step by the number of moving constraints.

EDCD: Error-Driven Constraint Demotion, the algorithm described by Tesar & Smolensky (1998). All constraints that prefer the adult form and are ranked above the highest-ranked constraint that prefers the learner's form, are demoted to the ranking of that last constraint minus 1.0.

Initial plasticity (standard value: 1.0)

Replications per plasticity (standard value: 100000)

Plasticity decrement (standard value: 0.1)

Number of plasticities (standard value: 4)

these four arguments determine the *learning scheme*, i.e. the number of times the grammar will receive data at a certain plasticity. With the standard values, there will be 100000 data while the plasticity is 1.0 (the initial plasticity), 100000 data while the plasticity is 0.1, 100000 data while the plasticity is 0.01, and 100000 data while the plasticity is 0.001. If you want learning at a constant plasticity, set the *number of plasticities* to 1.

Rel. plasticity spreading (standard value: 0.1)

if this is not 0, the size of the learning step will vary randomly. For instance, if the plasticity is set to 0.01, and the relative plasticity spreading is 0.1, you will get actual learning steps that could be

anywhere between 0.007 and 0.013, according to a Gaussian distribution with mean 0.01 and standard deviation 0.001.

Honour local rankings (standard value: on)

if this is on, the fixed rankings that you supplied in the grammar will be maintained during learning: if a constraint falls below a constraint that is supposed to be universally lower-ranked, this second constraint will be demoted as well.

Number of chews (standard value: 1)

the number of times that each input-output pair is fed to the grammar. Setting this number to 20 will give a slightly different (perhaps more accurate) result than simply raising the plasticity by a factor of 20.

Links to this page

- [OT learning](#)
-

© *ppgb*, September 16, 2003

OT learning 7. Learning from overt forms

In order to be able to learn phonological production, both EDCD and GLA require pairs of underlying form and surface form. However, the language-learning child hears neither of these forms: she only hears *overt forms*, with less structural information than the underlying and surface forms contain.

Interpretive parsing

The language-learning child has to construct both the surface form and the underlying form from the overt form that she hears. Tesar & Smolensky (1998) proposed that the child computes a surface form from the overt form by using the same constraint ranking as in production. For instance, the overt form [σσ [IMAGE] σ], which is a sequence of three syllables with stress on the second syllable, will be interpreted as the surface form / (σσ [IMAGE]) σ / in iambic left-aligning languages (IAMBIC >> TROCHAIC, and ALLFEETLEFT >> ALLFEETRIGHT), but as the surface form /σ (σ [IMAGE] σ) / in trochaic right-aligning languages. Tesar & Smolensky call this procedure Robust Interpretive Parsing, because it works even if the listener's grammar would never produce such a form. For instance, if IAMBIC >> ALLFEETRIGHT >> TROCHAIC >> ALLFEETLEFT, the listener herself would produce the iambic right-aligned /σ (σσ [IMAGE]) / for any trisyllabic underlying form, though she will still interpret [σσ [IMAGE] σ] as / (σσ [IMAGE]) σ /, which is illegal in her own grammar. Hearing forms that are illegal in one's own grammar is of course a common situation for language-learning children.

In Tesar & Smolensky's view, the underlying form can be trivially computed from the surface form, since the surface form *contains* enough information. For instance, the surface form / (σσ [IMAGE]) σ / must lead to the underlying form [IMAGE] σσσ [IMAGE] if all parentheses and stress marks are removed. Since McCarthy & Prince (1995), this *containment* view of surface representations has been abandoned. In PRAAT, therefore, the underlying form is not trivially computed from the surface form, but all the tableaux are scanned for the surface form that violates the least high-ranked constraints (in the usual OT sense), as long as it contains the given overt form. For instance, if IAMBIC >> ALLFEETRIGHT >> TROCHAIC >> ALLFEETLEFT, the overt form [σσ [IMAGE] σ] occurs in two candidates: the surface form / (σσ [IMAGE]) σ / in the tableau for the underlying form [IMAGE] σσσ [IMAGE], and the surface form /σ (σ [IMAGE] σ) / in the tableau for the underlying form [IMAGE] σσσ [IMAGE]. The best candidate is the surface form / (σσ [IMAGE]) σ / in the tableau for the underlying form [IMAGE] σσσ [IMAGE]. Hence, PRAAT's version of Robust Interpretive Parsing will map the overt form [σσ [IMAGE] σ] to the underlying form [IMAGE] σσσ [IMAGE] (the 'winning tableau') and to the surface form / (σσ [IMAGE]) σ / (to be sure, this is the same result as in Tesar & Smolensky's version, but crucial differences between the two versions will appear when faithfulness constraints are involved).

In PRAAT, you can do interpretive parsing. For example, create a grammar with **Create metrics grammar...** from the New menu. Then choose **Get interpretive parse...** from the **Query** submenu and supply "[L1 L L]" for the overt form, which means a sequence of three light syllables with a main stress on the first. The Info window will show you the optimal underlying and surface forms, given the current constraint ranking.

Learning from partial forms

Now that the learning child can convert an overt form to an underlying-surface pair, she can compare this surface form to the surface form that she herself would have derived from this underlying form. For instance, If IAMBIC >> ALLFEETRIGHT >> TROCHAIC >> ALLFEETLEFT, the winning tableau is , and the perceived adult surface form is $/(\sigma \sigma \sigma) \sigma /$. But from the underlying form , the learner will derive $/\sigma (\sigma \sigma) /$ as her own surface form. The two surface forms are different, so that the learner can take action by reranking one or more constraints, perhaps with EDCD or GLA.

In PRAAT, you can learn from partial forms. Select the metrics grammar and choose **Learn from one partial output...**, and supply "[L1 L L]". If you do this several times, you will see that the winner for the tableau "[L L L]" will become one of the two forms with overt part "[L1 L L]".

To run a whole simulation, you supply a Distributions object with one column, perhaps from a text file. The following text file shows the overt forms for Latin, with the bisyllabic forms occurring more often than the trisyllabic forms:

```
"ooTextFile"
"Distributions"
1 column with numeric data
  "Latin"
28 rows
"[L1 L]" 25
"[L1 H]" 25
"[H1 L]" 25
"[H1 H]" 25
"[L1 L L]" 5
"[H1 L L]" 5
"[L H1 L]" 5
"[H H1 L]" 5
"[L1 L H]" 5
"[H1 L H]" 5
"[L H1 H]" 5
"[H H1 H]" 5
"[L L1 L L]" 1
"[L H1 L L]" 1
"[L L H1 L]" 1
"[L H H1 L]" 1
"[L L1 L H]" 1
"[L H1 L H]" 1
"[L L H1 H]" 1
"[L H H1 H]" 1
"[H L1 L L]" 1
"[H H1 L L]" 1
"[H L H1 L]" 1
```

```
"[H H H1 L]" 1
"[H L1 L H]" 1
"[H H1 L H]" 1
"[H L H1 H]" 1
"[H H H1 H]" 1
```

Read this file into PRAAT with Read from file.... A Distributions object then appears in the object list. Click To Strings..., then OK. A Strings object containing 1000 strings, drawn randomly from the distribution, with relative frequencies as in the text file, will appear in the list. Click Inspect to check the contents.

You can now select the OTGrammar together with the Strings and choose **Learn from partial outputs...** A thousand times, PRAAT will construct a surface form from the overt form by interpretive parsing, and also construct the underlying form in the same way, from which it will construct another surface form by evaluating the tableau. Whenever the two surface forms are not identical, some constraints will be reranked. In the current implementation, the disharmonies for interpretive parsing and for production are the same, i.e., if the evaluation noise is not zero, the disharmonies are randomly renewed before each interpretive parsing but stay the same for the subsequent virtual production.

Links to this page

- [OT learning](#)
-

© *ppgb*, December 20, 2003

Constraints

In Optimality Theory, the ‘rules’ that an output form has to satisfy. Since there can be many constraints and these constraints can conflict with each other, the constraints are *violable* and the highest-ranked constraints have the largest say in determining the optimal output.

See the OT learning tutorial for many examples.

Links to this page

- [OT learning 1. Kinds of OT grammars](#)
-

© *ppgb*, November 5, 2002

Create tongue-root grammar...

A command in the New menu for creating an OTGrammar object with a tongue-root-harmony grammar.

These OTGrammar grammars only accept inputs of the form $V_1 tV_2$, where V_1 and V_2 are chosen from the six front vowels i , $[i]$, e , $[e]$, and a . In a text field, these vowels should be typed as \mathbf{i} , $\backslash\mathbf{ic}$, \mathbf{e} , $\backslash\mathbf{ep}$, $\backslash\mathbf{sw}$, and \mathbf{a} , respectively (see Special symbols).

The following phonological features are relevant:

ATRRTR

high $[i]$

mid e

low $[a]$

Constraints

The resulting OTGrammar will usually contain at least the following five constraints:

*[rtr / hi]

"do not implement [retracted tongue root] if the vowel is high."

*[atr / lo]

"do not implement [advanced tongue root] if the vowel is low."

PARSE (rtr)

"make an underlying [retracted tongue root] specification surface."

PARSE (atr)

"make an underlying [advanced tongue root] specification surface."

*GESTURE (contour)

"do not go from advanced to retracted tongue root, nor the other way around, within a word."

This set of constraints thus comprises:

- two *grounding conditions* (Archangeli & Pulleyblank (1994)), which we can see as gestural constraints;
- two *faithfulness constraints*, which favour the similarity between input and output, and can be seen as implementing the principle of maximization of perceptual contrast;
- a *harmony constraint*, which, if crucially ranked higher than at least one faithfulness constraint, forces *tongue-root harmony*.

In addition, there may be the following four constraints:

*[rtr / mid]

"do not implement [retracted tongue root] if the vowel is mid; universally ranked lower than *[rtr / hi]."

*[rtr / lo]

"do not implement [retracted tongue root] if the vowel is low; universally ranked lower than *[rtr / mid]."

*[atr / mid]

"do not implement [advanced tongue root] if the vowel is mid; universally ranked lower than *[atr / lo]."

*[atr / hi]

"do not implement [advanced tongue root] if the vowel is high; universally ranked lower than *[atr / mid]."

The universal rankings referred to are due to the *local-ranking principle* (Boersma (1997a)). A learning algorithm may enforce this principle, e.g., if *[rtr / hi] falls down the ranking scale, *[rtr / mid] may be pushed along.

For information on learning these tongue-root grammars, see OT learning and Boersma (2000).

Links to this page

- OT learning 3.2. Data from another grammar
- OT learning 4. Learning an ordinal grammar

© ppgb, December 4, 2002

Optimality Theory

A framework for transferring one linguistic representation into another, e.g. transferring an underlying form into a surface form. Before Prince & Smolensky (1993), phonologists tended to this with a sequentially ordered set of rules, each of which transferred a representation into another. With OT (that's the abbreviation), there are no intermediate steps in the derivation, but a set of ranked constraints chooses the optimal output form from a set of candidates.

In PRAAT, you can draw Optimality-Theoretic tableaux and simulate Optimality-Theoretic learning. See the OT learning tutorial.

Links to this page

- [OT learning 1. Kinds of OT grammars](#)
-

© *ppgb*, November 5, 2002

OTAnyGrammar

One of the types of objects in PRAAT. Virtual... See OT learning.

Class description

Every descendant class (*subclass*) of **OTAnyGrammar** contains:

- A list of constraint names.
- A method for evaluating each constraint on any input (EVAL).
- A method for generating output candidates (GEN).
- A method for generating input data relevant to the grammar (richness of the base).

An **OTAnyGrammar** object contains an ordered collection of **OTConstraint** objects. The order may change during learning.

Each **OTConstraint** object contains the following attributes:

integer *which*

a pointer into the fixed list of constraints; will never change.

real *ranking*

the continuous ranking value; will change during learning.

real *disharmony*

the effective ranking value during evaluation; with stochastic evaluation, this will be different from *ranking*.

OTAnyGrammar creation

You can easily create some examples of OTGrammar_tongueRoot.

OTAnyGrammar actions

You can perform the following actions on every object of class **OTAnyGrammar** or a subclass thereof (like OTGrammar_tongueRoot).

- OTAnyGrammar: Generate one input
- OTAnyGrammar: Generate inputs...
- OTAnyGrammar: Sort...
- OTAnyGrammar: Input to output...
- OTAnyGrammar & Strings: Inputs to outputs...
- OTAnyGrammar: Learn one (T&S)...
- OTAnyGrammar: Learn one (GLA)...
- OTAnyGrammar & 2 Strings: Learn (T&S)
- OTAnyGrammar & 2 Strings: Learn (GLA)...

You can view an **OTAnyGrammar** in an OTAnyGrammarEditor.

Links to this page

- [OT learning 1. Kinds of OT grammars](#)
- [OTAnyGrammar examples](#)
- [OTGrammar](#)

© *ppgb*, March 16, 2003

OTAnyGrammar & 2 Strings: Learn (GLA)...

Causes the selected OTAnyGrammar object to process a number of input/output pairs according to the Minimal Gradual Learning Algorithm by Boersma (2000). See OT learning.

Links to this page

- [OTAnyGrammar examples](#)
 - [OTAnyGrammar: Learn one \(GLA\)...](#)
-

© *ppgb*, October 27, 2000

OTAnyGrammar & 2 Strings: Learn (T&S)

Causes the selected OTAnyGrammar object to process a number of input/output pairs according to the Error-Driven Constraint Demotion algorithm by Tesar & Smolensky (1998). See OT learning.

Links to this page

- [OTAnyGrammar examples](#)
 - [OTAnyGrammar: Learn one \(T&S\)...](#)
-

© *ppgb*, December 19, 1998

OTAnyGrammar examples

This page describes the use of OTAnyGrammar objects. For most problems, you would use OTGrammar instead, as explained in the OT learning tutorial.

Example 1: Tongue-root grammars

Choose Create tongue-root method grammar... from the *OT* submenu of the New menu. Set *Constraint set* to "Five", and *Ranking* to "Wolof". Click *OK*. An object called "OTGrammar_tongueRoot Wolof" will appear in the list. Click *Edit*. You will see the following grammar appear in the OTAnyGrammarEditor (constraint name & ranking & disharmony):

```
*[rtr / hi] 100.000 100.000
PARSE (rtr) 50.000 50.000
*GESTURE (contour) 30.000 30.000
PARSE (atr) 20.000 20.000
*[atr / lo] 10.000 10.000
```

This grammar, with five constraints with clearly different rankings, is equivalent to the traditional OT ranking

```
*[rtr / hi] >> PARSE (rtr) >> *GESTURE (contour) >> PARSE (atr) >> *[atr / lo]
```

See OTGrammar_tongueRoot for the meanings of these constraints.

Evaluation

When given an input string, an OT grammar can compute the output.

Example. Select your Wolof grammar and click *To Tableau....* For *Input string*, type "ati". An object called "OTAnyTableau Wolof_ati" will appear in the list. Click *Draw...* to draw the evaluation tableau to the Picture window, in a standard OT format with asterisks for violations, exclamation marks for crucial violations, and grey shading for irrelevant cells. You see that the tableau evaluates the four candidates [ita], [it_[IMAGE]], [_[IMAGE] ta], and [_[IMAGE] t_[IMAGE]], and that [ati] is the winner (the resulting output). For an input /at_[IMAGE] / (typed as "at\sw", see Special symbols), the winner is [ata].

See OTGrammar_tongueRoot for the possible inputs and for the generation of output candidates.

Generating inputs from the grammar

According to Prince & Smolensky (1993), the input to an OT grammar can be *anything*. This is the idea of *richness of the base*. When doing a practical investigation, however, we are only interested in the inputs that will illustrate the properties of our partial grammars.

In Praat, therefore, a set of inputs can be generated from an OTAnyGrammar.

Example. Select the Wolof tongue-root grammar and click **Generate inputs...** Set *Number of inputs* to 100, and click **OK**. An object named "Strings tongueRoot_inputs" will appear in the list. Click the **Inspect** button and examine the 100 input strings. You will see that they have been randomly chosen from the 36 possible $V_1 tV_2$ sequences where V_1 and V_2 are any of the six front vowels i, [IMAGE], e, ε, [IMAGE], and a, as described at OTGrammar_tongueRoot.

See OTAnyGrammar: Generate one input and OTAnyGrammar: Generate inputs....

Generating inputs from a distribution

You can also generate a Strings object from a Distributions object (see Distributions: To Strings...).

Stochastic evaluation

In the above example, the disharmonies equalled the ranking values. The three commands discussed here let you specify a *ranking spreading*.

You can see random evaluation even if the ranking spreading is zero: view an **OTAnyGrammar** in an OTAnyGrammarEditor and choose **Sort...** several times with a *Noise* of 0.0. If the grammar contains constraints with equal rankings, their order will randomly vary, as described at OTAnyGrammar: Sort....

To compute directly an output string from a given input string, use OTAnyGrammar: Input to output...; because this command lets you specify a ranking spreading, the disharmonies in the OTAnyGrammar will change, and so may, therefore, the order of the constraints.

To compute a *list* of output strings from a list of input strings, use OTAnyGrammar & Strings: Inputs to outputs....

Learning from an input-output pair

For learning in general, you need an adult model form. If there are faithfulness constraints involved, you also need an underlying form.

Tesar & Smolensky's learning algorithm

We will show how the Error-Driven Constraint Demotion (EDCD) algorithm of Tesar & Smolensky (1998) would allow a learner to acquire the tongue-root-harmony system of Wolof.

1. Create a five-constraint tongue-root grammar with "equal" initial rankings (with Create tongue-root method grammar...), and click **Edit**. All the constraints have a ranking value of 100 (Tesar & Smolensky use no ranking values, but this is the way we can simulate their algorithm).
2. Check that the possible outputs of the input /ita/ are [ita], [IMAGE ta], and [it[IMAGE]], by repeatedly performing the command OTAnyGrammar: Input to output... with an *Input string* of "ita" and a ranking spreading of 0. You could also create a Strings object that contains the string "ita" a thousand times, then

select the grammar and this **Strings**, then click the command OTAnyGrammar & Strings: Inputs to outputs... with a *Noise* of 0, and Inspect the resulting output **Strings**.

3. The true Wolof output is [ita]. You can teach the grammar by choosing OTAnyGrammar: Learn one (T&S).... Set both the *Input string* and the *Output string* to "ita".

4. Because of the random evaluation of tied constraints, you may have to do this several times, but eventually, the ranking of the constraint *GESTURE (contour) will fall to 99, given the fixed demotion step of 1 with which we simulate Tesar & Smolensky's stratification algorithm (the grammar started with one stratum; it now has two).

5. Once *GESTURE (contour) has fallen to 99, no amount of teaching the /ita/ - [ita] pair will change the grammar any further. This is the 'tit-for-tat' learning strategy discussed below.

6. Proceed with other relevant pairs: /etɛ/ - [ɛ tɛ], to teach the grammar that the harmony constraint dominates ATR faithfulness; and /_[IMAGE] t_[IMAGE] / - [_[IMAGE] t_[IMAGE]], to teach the grammar that ATR faithfulness outranks the grounding constraint *[atr / lo]. You may have to re-teach earlier input-output pairs after a grammar change.

7. The grammar will always end up in the following state:

```
*[rtr / hi] 100.000 100.000
PARSE (rtr) 99.000 99.000
*GESTURE (contour) 98.000 98.000
PARSE (atr) 97.000 97.000
*[atr / lo] 96.000 96.000
```

Apparently, this is a grammar with five strata: all the constraints are crucially ranked with respect to one another. It took 10 learning steps to arrive at the adult ranking. The number of input-output pairs is generally much higher than 10. For instance, the relative ranking of PARSE (atr) and *[atr / lo] can only be learned from the output [_[IMAGE] t_[IMAGE]] (which has a probability of occurrence of 1/36), because the outputs [_[IMAGE] ti] and [_[IMAGE] te] could have been forced by the ranking of *GESTURE (contour) above *[atr / lo].

Robustness

Tesar & Smolensky's algorithm is not very resistant against errors in the data. For instance, start from the above final state of the grammar, and feed it the incorrect pair /ɛ te/ - [ete]. The result will be:

```
*[rtr / hi] 100.000 100.000
*GESTURE (contour) 98.000 98.000
PARSE (atr) 97.000 97.000
PARSE (rtr) 96.000 96.000
*[atr / lo] 96.000 96.000
```

PARSE (rtr) has fallen three strata down the hierarchy. This grammar would now regard a new /ε te/ - [ete] datum as correct (the ‘tit-for-tat’ strategy: imitate the most recently perceived action). But it will take the learner nine steps to climb out of this miserable situation, i.e., she will have to make nine errors before she has restored a correct grammar. Regarding the relative stability of a language system, however, we would rather expect the ratio of errors with the learner and in the language environment to be about one, not nine.

Simulating the learning process

Instead of feeding the grammar one input-output pair at a time, we can feed it a thousand pairs:

1. Create a five-constraint Wolof grammar (Create tongue-root method grammar...).
2. Generate 1000 input strings (OTAnyGrammar: Generate inputs...).
3. Compute the 1000 output strings (OTAnyGrammar & Strings: Inputs to outputs...): select the grammar and the input Strings, choose **Inputs to outputs...**, and specify 0.0 for the *Noise*.
4. Create a five-constraint "equal" grammar (Create tongue-root method grammar...).
5. Select this monostratal grammar, the input strings, and the output strings, and click **Learn (T&S)**. See OTAnyGrammar & 2 Strings: Learn (T&S).

After this procedure, the grammar will very probably have the correct final state.

The Gradual Learning Algorithm

The learning strategy discussed above (OTAnyGrammar: Learn one (T&S)..., OTAnyGrammar & 2 Strings: Learn (T&S)) can (almost) be seen as a special case of a more general algorithm with small demotion steps along a continuous scale. If you use OTAnyGrammar: Learn one (GLA)..., you can see the grammar slowly changing. With OTAnyGrammar & 2 Strings: Learn (GLA)..., you will see the grammar reaching the correct final state after a few thousand data (with the standard settings).

The algorithm is described in Boersma (2000), which proves its convergence, and in Boersma (1997b), which illustrates its relation to variation and optionality.

Safety margin and stochastic evaluation

What is a *small* demotion step? This must be taken relative to another quantity. This quantity is the evaluation noise.

If the *ranking spreading* were zero, the demotions in the GLA would immediately stop once that a constraint has fallen below its competitors. If the data contain an error, the grammar will change to an incorrect state, and the learner has to make an error to correct it. Though the error ratio is now one (because of the constant demotion step), the learner’s behaviour can still be described as a ‘tit-for-tat’ strategy, which is an unknown phenomenon in human speech variation.

The solution is to have a finite ranking spreading: in this way, the learner will continue making errors (though less than 50%) after the constraint has fallen below its competitors. For instance, with a ranking spreading of 2.0, the distance between the constraints will be about 10 after a few thousand relevant data: the *safety margin*. If the mean demotion step is 0.1, there is no chance that a modest amount of erroneous data will reverse the ranking. It is true, however, that one erroneous datum will decrease the constraint

distance by 0.1, so that the learner will have to make one mistake herself to restore the original distance. But this is only natural, as she may well wait a long time before doing this: on the average, the same number of thousands of data. This is the *patient error-matching learner*.

Learning from surface data alone

Many a young learner will take the adult surface forms, as perceived by herself, as her underlying forms. In other words, the input to her grammar will equal the output of the adult grammar.

We can simulate what happens here by taking the adult output as the input to the learning algorithms:

1. Create a five-constraint Wolof grammar.
2. Generate 1000 input strings.
3. Compute the 1000 output strings.
4. Create a five-constraint "equal" grammar.
5. Select this monostratal grammar and the output strings only, and click **Learn output (T&S)**. See OTAnyGrammar & Strings: Learn output (T&S).

The result will be a grammar where the faithfulness constraints outrank all the gestural constraints that can be violated in any output string:

```
*[rtr / hi] 100.000 100.000
PARSE (atr) 100.000 100.000
PARSE (rtr) 100.000 100.000
*GESTURE (contour) 99.000 99.000
*[atr / lo] 99.000 99.000
```

You will get a comparable result with OTAnyGrammar & Strings: Learn output (GLA)...

The resulting grammar represents the learner's state after the acquisition of all the relevant gestures. The learner will now faithfully reproduce /etɛ/ if that were in her lexicon. Before being able to render such an underlying form as [ɛ tɛ], she must learn that faithfulness can be violated.

Example 2: When underlying forms are irrelevant

Underlying forms are relevant only if faithfulness constraints are involved. If a grammar only contains constraints that evaluate the output, we need no input strings for our simulations. However, if the relevant constraint had fixed rankings, there would only be a single possible output, which seems uninteresting. An interesting output-only grammar, therefore, necessarily features stochastic evaluation, and at least some of the constraints will have rankings that are close to each other.

Example. Hayes & MacEachern (1998) identify 11 output-oriented constraints for the forms of quatrains in English folk verse.

1. Create a folk-verse grammar with equal constraint rankings (all 100). You may find it in the file **folkVerse.OTGrammar** in the **demo** directory of your **Praat** distribution, or get it from <http://www.fon.hum.uva.nl/praat/folkVerse.OTGrammar>.
2. Generate 1000 input strings. They will all be empty strings.

3. Read the file that contains the surface distribution of the possible outputs. It is in the **demo** folder or at <http://www.fon.hum.uva.nl/praat/folkVerse.Distributions>. A Distributions object will appear in the list. Column "Actual" is the last column of table (10) in Hayes & MacEachern (1998).
4. From this surface distribution, create a list of 1000 output strings, using Distributions: To Strings... (set *column* to 1).
5. Select the grammar, the "input strings", and the output strings, and learn in the usual way. After learning, you can see that some constraints have risen above 100, and some have fallen below 100.

With each of the 1000 outputs, the learner can be regarded as having generated a quatrain herself and compared it to a quatrain in her folk-verse environment. If these quatrains are equal (a 10% chance or so), nothing happens. Otherwise, the learner will demote the highest violated constraint (i.e., the one that is most disharmonic during her stochastic evaluation) in the heard quatrain that is not (or less) violated in the winner (the quatrain that she generated herself). She will also promote the highest violated constraint in the winner that is not (or less) violated in the heard quatrain.

We are next going to generate a set of 589 quatrains, in order to be able to compare the behaviours of our folk-verse grammar and the English folk-verse writers:

1. Select the learned grammar and generate 589 (empty) input strings.
2. Select the learned grammar and the so-called input strings, and generate the output strings.
3. To see the distribution of the output strings, choose Strings: To Distributions, and draw the resulting Distributions object to your Picture window.
4. You can now compare the two distributions.

Instead of generating the data from a Distributions, you could have generated them from the target grammar in table (9) of Hayes & MacEachern (1998). Such a grammar is in your **demo** folder (**folkVerse59.OTGrammar**) or at <http://www.fon.hum.uva.nl/praat/folkVerse59.OTGrammar>. Because of the loosening of the tie between two of the constraints (see H & McE, fn. 43), this grammar will give different distributions from the "actual" values, but our algorithm will learn them correctly, provided you choose **Symmetric all** or **weighted uncanceled** for the learning strategy.

OTAnyGrammar: Generate inputs...

A command to create a Strings object from a selected OTAnyGrammar.

A practical grammar-specific implementation of the *richness of the base*. See OT learning.

Argument

Number of strings

the number of times a string will be drawn from the possible inputs to the grammar.

Links to this page

- [OTAnyGrammar examples](#)
- [OTAnyGrammar: Generate one input](#)

© ppgb, October 22, 1997

OTAnyGrammar: Learn one (GLA)...

Causes every selected OTAnyGrammar object to process one input/output pair according to the Minimal Gradual Learning Algorithm by Boersma (2000). See OT learning.

This is a one-string version of OTAnyGrammar & 2 Strings: Learn (GLA)...

Links to this page

- [OTAnyGrammar examples](#)

© *ppgb*, October 27, 2000

OTAnyGrammar: Learn one (T&S)...

Causes every selected OTAnyGrammar object to process one input/output pair according to the Error-Driven Constraint Demotion algorithm by Tesar & Smolensky (1998). See OT learning.

This is a one-string version of OTAnyGrammar & 2 Strings: Learn (T&S).

Links to this page

- [OTAnyGrammar examples](#)

© *ppgb*, December 19, 1998

OTAnyGrammarEditor

An Editor for viewing an OTAnyGrammar object. To raise an **OTAnyGrammarEditor**, select an **OTAnyGrammar** object and click *Edit*.

You can see the grammar change as you teach it. See OT learning for more information.

Links to this page

- [OTAnyGrammar examples](#)
- [Types of objects](#)

© *ppgb*, October 21, 1997

OTGrammar_tongueRoot

One of the types of objects in PRAAT. A subclass of OTAnyGrammar.

Creation

You can create an **OTGrammar_tongueRoot** object with Create tongue-root method grammar..., or by reading one from a file (which may be an edited version of a saved **OTGrammar_tongueRoot** that you created earlier).

Inputs

These grammars only accept inputs of the form $V_1 tV_2$, where V_1 and V_2 are chosen from the six front vowels **i**, , **e**, , and **a**. In a text field, these vowels should be typed as **i**, **\ic**, **e**, **\ep**, **\sw**, and **a**, respectively (see Special symbols).

When asked to generate a random input, these grammars produce any of the 36 possible $V_1 tV_2$ sequences, all with equal probability.

Output generation

The following phonological features are relevant:

ATTRRTR

high 

mid 

low  a

For a given input, GEN (the generator of this grammar) generates four output candidates: the vowel heights will be same as those in the input, but the tongue-root values of V_1 and V_2 are varied. For example, for the input [ita] GEN will generate the four candidates [ita], [it ], [ ta], and [ t ].

Constraints

An **OTGrammar_tongueRoot** will usually contain at least the following five constraints:

*[rtr / hi]

"do not implement [retracted tongue root] if the vowel is high."

*[atr / lo]

"do not implement [advanced tongue root] if the vowel is low."

PARSE (rtr)

"make an underlying [retracted tongue root] specification surface."

PARSE (atr)

"make an underlying [advanced tongue root] specification surface."

*GESTURE (contour)

"do not go from advanced to retracted tongue root, nor the other way around, within a word."

This set of constraints thus comprises:

- two *grounding conditions* (Archangeli & Pulleyblank (1994)), which we can see as gestural constraints;
- two *faithfulness constraints*, which favour the similarity between input and output, and can be seen as implementing the principle of maximization of perceptual contrast;
- a *harmony constraint*, which, if crucially ranked higher than at least one faithfulness constraint, forces *tongue-root harmony*.

In addition, there may be the following four constraints:

*[rtr / mid]

"do not implement [retracted tongue root] if the vowel is mid; universally ranked lower than *[rtr / hi]."

*[rtr / lo]

"do not implement [retracted tongue root] if the vowel is low; universally ranked lower than *[rtr / mid]."

*[atr / mid]

"do not implement [advanced tongue root] if the vowel is mid; universally ranked lower than *[atr / lo]."

*[atr / hi]

"do not implement [advanced tongue root] if the vowel is high; universally ranked lower than *[atr / mid]."

The universal rankings referred to are due to the *local-ranking principle* (Boersma (1997a)). A learning algorithm may enforce this principle, e.g., if *[rtr / hi] falls down the ranking scale, *[rtr / mid] may be pushed along.

For information on learning these tongue-root grammars, see OT learning and Boersma (2000).

Links to this page

- OTAnyGrammar examples

© ppgb, March 16, 2003

SSCP

One of the types of objects in PRAAT.

An object of type SSCP represents the sums of squares and cross products of a multivariate data set.

Besides the matrix part, an object of type SSCP also contains a vector with centroids.

Inside a SSCP

With Inspect you will see that this type contains the same attributes as a TableOfReal with the following extras:

numberOfObservations

centroid

Links to this page

- [Covariance](#)
- [Discriminant & SSCP: Project](#)
- [Discriminant: Extract pooled within-groups SSCP](#)
- [Discriminant: Extract within-group SSCP...](#)
- [Eigen & SSCP: Project](#)
- [PCA & SSCP: Project](#)
- [Principal component analysis](#)
- [SSCP & TableOfReal: Extract quantile range...](#)
- [SSCP: Draw sigma ellipse...](#)
- [SSCP: Get confidence ellipse area...](#)
- [SSCP: Get diagonality \(bartlett\)...](#)
- [SSCP: Get fraction variation...](#)
- [SSCP: Get sigma ellipse area...](#)
- [SSCP: To CCA...](#)
- [SSCP: To Covariance...](#)
- [TableOfReal: To SSCP...](#)

© djmw, November 3, 1998

Covariance

One of the types of objects in PRAAT.

An object of type Covariance represents the sums of squares and cross products of a multivariate data set divided by the number of observations.

An object of type Covariance contains the same attributes as an object of type SSCP.

Since an object of type Covariance contains the mean values (the centroids), the covariances as well as the number of observations it has all the information necessary to be the subject of all kinds of statistical tests on means and variances.

Links to this page

- [Covariance & TableOfReal: Extract quantile range...](#)
- [Covariance: Difference](#)
- [Covariance: Get fraction variance...](#)
- [Covariance: Get significance of means difference...](#)
- [Covariance: Get significance of one mean...](#)
- [Covariance: Get significance of one variance...](#)
- [Covariance: Get significance of variance ratio...](#)
- [Covariance: To TableOfReal \(random sampling\)...](#)
- [PCA & Covariance: Project](#)
- [Principal component analysis](#)
- [SSCP: To Covariance...](#)
- [T-test](#)
- [TableOfReal: To Covariance](#)

© *djmw*, January 5, 1999

Correlation

One of the types of objects in PRAAT.

An object of type Correlation represents the correlation coefficients of a multivariate data set.

Links to this page

- [Canonical correlation analysis](#)
- [CCA & Correlation: To TableOfReal \(loadings\)](#)
- [Correlation: Confidence intervals...](#)
- [Principal component analysis](#)
- [TableOfReal: To Correlation](#)
- [TableOfReal: To Correlation \(rank\)](#)

© *djmw*, January 5, 1999

PCA

One of the types of objects in PRAAT. See the Principal component analysis tutorial.

An object of type PCA represents the principal components analysis of a multivariate dataset.

Commands

Creation:

- Principal component analysis tutorial
- TableOfReal: To PCA

Inside a PCA

With Inspect you will see that this type contains the same attributes as an Eigen with the following extras:

numberOfObservations

the number of observations in the multivariate dataset that originated the PCA, usually equal to the dataset's number of rows.

labels[1..dimension]

the label that corresponds to each dimension.

centroid

the centroids of the originating multivariate data set.

Links to this page

- Eigen: Draw eigenvector...
- PCA & Configuration: To TableOfReal (reconstruct)
- PCA & Covariance: Project
- PCA & SSCP: Project
- PCA & TableOfReal: Get fraction variance...
- PCA & TableOfReal: To Configuration...
- PCA: Get eigenvalue...
- PCA: Get eigenvector element...
- PCA: Get equality of eigenvalues...
- PCA: Get fraction variance accounted for...
- PCA: Get number of components (VAF)...

TableOfReal

One of the types of objects in PRAAT.

A TableOfReal object contains a number of *cells*. Each cell belongs to a *row* and a *column*. For instance, a TableOfReal with 10 rows and 3 columns has 30 cells.

Each row and each column may be labeled with a *title*.

Creating a TableOfReal from data in a text file

Suppose you have F1 and F2 data for vowels. You can create a simple text file like the following:

```
"ooTextFile" ! The line by which Praat can recognize your file
"TableOfReal" ! The line that tells Praat about the contents
2 "F1" "F2" ! Number of columns, and column labels
3 ! Number of rows
"a" 800 1100 ! Row label (vowel), F1 value, F2 value
"i" 280 2800 ! Row label (vowel), F1 value, F2 value
"u" 260 560 ! Row label (vowel), F1 value, F2 value
```

Praat is rather forgiving about the use of spaces, tabs, and newlines. See Write to text file... for general information.

You will often have your data in a file with a self-describing format, i.e. in which the number of values on a line equals the number of columns of the table:

```
800 1100
280 2800
260 560
```

Such a file can be read with Read Matrix from raw text file.... This creates a Matrix object, which can be cast to a TableOfReal object by Matrix: To TableOfReal. The resulting TableOfReal does not have any row or column labels yet. You could add column labels with:

```
Set column label (index)... 1 F1
Set column label (index)... 2 F2
```

Of course, if the row labels contain crucial information, and the number of rows is large, this is not a feasible method.

Links to this page

- Canonical correlation analysis
- CCA & TableOfReal: To TableOfReal (loadings)
- CCA & TableOfReal: To TableOfReal (scores)...
- Confusion
- Confusion: To TableOfReal (marginals)
- Correlation: Confidence intervals...
- Covariance & TableOfReal: Extract quantile range...
- Covariance: To TableOfReal (random sampling)...
- Create TableOfReal (Pols 1973)...
- Discriminant & TableOfReal: To ClassificationTable...
- Discriminant & TableOfReal: To Configuration...
- Dissimilarity
- Distributions
- Eigen & TableOfReal: Project...
- FFNet: Extract weights...
- Formulas 7. Attributes of objects
- PCA & Configuration: To TableOfReal (reconstruct)
- PCA & TableOfReal: Get fraction variance...
- PCA & TableOfReal: To Configuration...
- PCA: To TableOfReal (reconstruct 1)...
- Principal component analysis
- SSCP
- SSCP & TableOfReal: Extract quantile range...
- T-test
- TableOfReal: Centre columns
- TableOfReal: Centre rows
- TableOfReal: Change column labels...
- TableOfReal: Change row labels...
- TableOfReal: Draw biplot...
- TableOfReal: Draw box plots...
- TableOfReal: Draw rows as histogram...
- TableOfReal: Get table norm
- TableOfReal: Normalize columns...
- TableOfReal: Normalize rows...
- TableOfReal: Normalize table...
- TableOfReal: Select columns where row...
- TableOfReal: Set value...
- TableOfReal: Standardize columns
- TableOfReal: To CCA...
- TableOfReal: To Configuration (lda)...
- TableOfReal: To Configuration (pca)...

- TableOfReal: To Correlation
 - TableOfReal: To Correlation (rank)
 - TableOfReal: To Covariance
 - TableOfReal: To Discriminant
 - TableOfReal: To Pattern and Categories...
 - TableOfReal: To PCA
 - TableOfReal: To SSCP...
 - What's new?
-

© *ppgb*, March 16, 2003

TableOfReal: Standardize columns

Standardizes each column of the selected TableOfReal.

The entries x_{ij} in the TableOfReal will change to:

$$(x_{ij} - \mu_j) / \sigma_j,$$

where μ_j and σ_j are the mean and the standard deviation as calculated from the j^{th} column, respectively. After standardization all column means will equal zero and all column standard deviations will equal one.

Links to this page

- [Principal component analysis](#)
-

© *djmw*, April 28, 1999

TableOfReal: To PCA

A command that creates a PCA object from every selected TableOfReal object.

Links to this page

- [Principal component analysis](#)
-

© *djmw*, January 6, 1998

Scree plot

A scree plot shows the sorted eigenvalues, from large to small, as a function of the eigenvalue index.

Links to this page

- [Eigen: Draw eigenvalues...](#)
 - [Principal component analysis](#)
-

© *djmw*, March 31, 2004

Eigen: Draw eigenvalues...

A command to draw the eigenvalues of the selected Eigen object(s).

Arguments

Fraction of eigenvalues summed

defines whether or not fractions are plotted. Fractions f_i will be calculated for each number e_i by dividing this number by the sum of all numbers e_j : $f_i = e_i / \sum_{j=1..numberOfEigenvalues} e_j$.

Cumulative

defines whether or not cumulative values are plotted. Cumulative values c_i will be calculated for each number e_i by summing the first i numbers e_j : $c_i = \sum_{j=1..i} e_j$.

A scree plot can be obtained if both *Fraction of eigenvalues summed* and *Cumulative* are unchecked.

Links to this page

- [Principal component analysis](#)

© djmw, April 7, 2004

PCA: Get fraction variance accounted for...

A command to query the selected PCA for the fraction *variance accounted for* by the selected components.

Arguments

Principal component range

defines the range of the principal components. If you choose both numbers equal, you get the fraction of the "variance" explained by that one component.

Details

The contribution is defined as:

$$\sum_{i=from..to} eigenvalue[i] / \sum_{i=1..numberOfEigenvalues} eigenvalue[i]$$

Links to this page

- [Principal component analysis](#)

© djmw, January 6, 1999

PCA: Get equality of eigenvalues...

A command to get the probability that some of the eigenvalues of the selected PCA object are equal. A low probability means that it is not very likely that these numbers are equal.

We test the hypothesis $H_0: \lambda_{from} = \dots = \lambda_{to}$ that $r (= to-from+1)$ of the eigenvalues λ of the covariance matrix are equal. The remaining eigenvalues are unrestricted as to their values and multiplicities. The alternative hypothesis to H_0 is that some of the eigenvalues in the set are distinct.

Arguments

Eigenvalue range

define the range of eigenvalues to be tested for equality.

Conservative test

when on, a more conservative estimate for n is chosen (see below).

Details

The test statistic is:

$$\chi^2 = -n \sum_{j=from..to} \ln eigenvalue[j] + n r \ln (\sum_{j=from..to} eigenvalue[j] / r),$$

with $r(r+1)/2 - 1$ degrees of freedom. Here $n = totalNumberOfCases - 1$.

A special case occurs when the variation in the last r dimensions is spherical. In a slightly more conservative test we may replace n by $n-from-(2r^2+r+2)/6r$.

Also see Morrison (1990), page 336.

Links to this page

- [Principal component analysis](#)

© djmw, November 2, 1998

PCA: Get number of components (VAF)...

A command to ask the selected PCA for the minimum number of components that are necessary to explain the given fraction *variance accounted for*.

Arguments

Variance accounted for (fraction)
the fraction variance accounted for that must be explained.

Links to this page

- [Principal component analysis](#)
-

© *djmw*, January 11, 1999

TableOfReal: To Configuration (pca)...

Calculates a Configuration based on the principal components from the selected TableOfReal.

Parameters

Number of dimensions

determines the number of dimensions of the resulting Configuration.

Algorithm

We form principal components without explicitly calculating the covariance matrix $\mathbf{C} = \mathbf{M}' \cdot \mathbf{M}$, where \mathbf{M} is the matrix part of the TableOfReal.

1. Make the singular value decomposition of \mathbf{M} . This results in $\mathbf{M} = \mathbf{U} \cdot \mathbf{d} \cdot \mathbf{V}'$.
2. Sort singular values \mathbf{d} and corresponding row vectors in \mathbf{V} (descending).
3. The principalComponent $ij = \sum_{k=1..numberOfColumns} M_{ik} \cdot V_{jk}$.

Remark

The resulting configuration is unique up to reflections along the new principal directions.

Links to this page

- [Principal component analysis](#)

© djmw, September 9, 1998

PCA & TableOfReal: To Configuration...

A command to construct a Configuration from the selected TableOfReal and PCA.

Arguments

Number of dimensions

determines the dimension of the resulting Configuration.

Algorithm

The TableOfReal is projected on the eigenspace of the PCA, i.e., each row of the TableOfReal is treated as a vector, and the inner product with the eigenvectors of the PCA determine its coordinates in the Configuration.

Because the algorithm performs a projection, the resulting Configuration will **only be centered**, i.e., its centroid will be at **0**, if the data in the TableOfReal object are centered too.

See also Eigen & TableOfReal: Project....

Links to this page

- [Principal component analysis](#)

© djmw, January 11, 1999

singular value decomposition

The *singular value decomposition* (svd) of a real $m \times n$ matrix \mathbf{A} is the factorization

$$\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}',$$

The matrices in this factorization have the following properties:

\mathbf{U} [$m \times m$] and \mathbf{V} [$n \times n$]

are orthogonal matrices. The columns \mathbf{u}_i of $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_m]$ are the *left singular vectors*, and the columns \mathbf{v}_i of $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]$ are the *right singular vectors*.

$\mathbf{\Sigma}$ [$m \times n$] = $\text{diag}(\sigma_1, \dots, \sigma_n)$

is a real, nonnegative, and diagonal matrix. Its diagonal contains the so called *singular values* σ_i , where $\sigma_1 \geq \dots \geq \sigma_n \geq 0$.

Links to this page

- [Principal component analysis](#)
- [TableOfReal: Draw biplot...](#)
- [TableOfReal: To CCA...](#)
- [TableOfReal: To Covariance](#)

© djmw, October 7, 1998

Configuration

One of the types of objects in PRAAT.

An object of type Configuration represents the positions of a number of labelled points in a multidimensional space.

How to create a Configuration

From the New menu:

- o Create Configuration...

By multidimensional scaling:

- o Dissimilarity: To Configuration (monotone mds)...
- o Dissimilarity: To Configuration (i-spline mds)...
- o Dissimilarity: To Configuration (interval mds)...
- o Dissimilarity: To Configuration (ratio mds)...
- o Dissimilarity: To Configuration (absolute mds)...

By multidimensional scaling with weights (Dissimilarity & Weight: To Configuration...):

- o **Dissimilarity & Weight: To Configuration (monotone mds)...**
- o **Dissimilarity & Weight: To Configuration (i-spline mds)...**
- o **Dissimilarity & Weight: To Configuration (interval mds)...**
- o **Dissimilarity & Weight: To Configuration (ratio mds)...**
- o **Dissimilarity & Weight: To Configuration (absolute mds)...**

By multidimensional scaling with a start Configuration:

- o Dissimilarity & Configuration: To Configuration (monotone mds)...
- o Dissimilarity & Configuration: To Configuration (i-spline mds)...
- o Dissimilarity & Configuration: To Configuration (interval mds)...
- o Dissimilarity & Configuration: To Configuration (ratio mds)...
- o Dissimilarity & Configuration: To Configuration (absolute mds)...

By transforming an existing Configuration:

- o Configuration: To Configuration (varimax)...
- o Configuration & AffineTransform: To Configuration
- o Configuration & Procrustus: To Configuration

From Principal component analysis:

- o TableOfReal: To Configuration (pca)...
- o PCA & TableOfReal: To Configuration...

From Discriminant analysis:

- TableOfReal: To Configuration (lda)...
- o Discriminant & TableOfReal: To Configuration...

How to draw a Configuration

- Configuration: Draw...
- **Configuration: Draw as numbers...**
- **Configuration: Draw as squares...**

How to modify a Configuration

- Configuration: Randomize
- Configuration: Rotate (pc) (to principal directions)
- Configuration: Rotate... (in a plane around the origin)
- Configuration: Invert dimension...
- Configuration: Normalize...

Inside a Configuration

With Inspect you will see the following attributes:

numberOfRows

the number of points (*numberOfPoints* \geq 1).

numberOfColumns

the dimension of the space (*numberOfDimensions* \geq 1).

rowLabels

the names associated with the points.

columnLabels

the names for the dimensions.

data [1..*numberOfPoints*][1..*numberOfDimensions*]

the coordinates of the points.

metric

determines the way distances between points are measured. In general the distance between points \mathbf{x}_i and \mathbf{x}_j is:

$$d_{ij} = (\sum_{k=1..numberOfDimensions} w_k |x_{ik} - x_{jk}|^{metric})^{1/metric}$$

For Euclidean distances *metric* is 2.

w [*1..numberOfDimensions*]

weight given to each dimension in the distance calculation.

Links to this page

- Configuration: Centralize
- Configuration: To Configuration (procrustus)
- Configuration: To Distance
- Configuration: To Similarity (cc)
- Configurations: To AffineTransform (congruence)...
- Configurations: To Procrustus
- congruence coefficient
- ContingencyTable: To Configuration (ca)...
- Correspondence analysis
- Create INDSCAL Carroll & Wish example...
- Dissimilarity & Configuration & Weight: Get stress...
- Dissimilarity & Configuration & Weight: To Configuration...
- Dissimilarity & Configuration: Draw regression (absolute mds)...
- Dissimilarity & Configuration: Draw regression (i-spline mds)...
- Dissimilarity & Configuration: Draw regression (interval mds)...
- Dissimilarity & Configuration: Draw regression (monotone mds)...
- Dissimilarity & Configuration: Draw regression (ratio mds)...
- Dissimilarity & Configuration: Draw Shepard diagram...
- Dissimilarity & Configuration: Get stress (absolute mds)...
- Dissimilarity & Configuration: Get stress (i-spline mds)...
- Dissimilarity & Configuration: Get stress (interval mds)...
- Dissimilarity & Configuration: Get stress (monotone mds)...
- Dissimilarity & Configuration: Get stress (ratio mds)...
- Dissimilarity & Configuration: To Configuration (kruskal)...
- Dissimilarity & Weight: To Configuration...
- Dissimilarity: To Configuration (kruskal)...
- Distance & Configuration & Salienc: Get VAF...
- Distance & Configuration & Salienc: To Configuration (indscal)...
- Distance & Configuration: Get VAF...
- Distance & Configuration: To Configuration (indscal)...
- Distance: To Configuration (indscal)...
- Distance: To Configuration (ytl)...
- INDSCAL analysis
- Kruskal analysis
- MDS models
- Multidimensional scaling
- PCA & Configuration: To TableOfReal (reconstruct)
- Salienc

© *djmw*, September 9, 1998

Dissimilarity

One of the types of objects in PRAAT.

It represents a one-way table with dissimilarities between "objects".

Creating a Dissimilarity from data in a text file

Suppose you have three objects A, B and C. In one way or another, you have acquired the following (symmetric) dissimilarities: $\delta_{AB} = 2$ ($= \delta_{BA}$), $\delta_{AC} = 1$ ($= \delta_{CA}$), and $\delta_{BC} = 1.4$ ($= \delta_{CB}$), where δ_{AB} represents the dissimilarity between object A and object B.

You can create a simple text file like the following:

```
"ooTextFile" ! The line by which Praat can recognize your file
"Dissimilarity" ! The line that tells Praat about the contents
3 "A" "B" "C" ! Number of columns, and column labels
3 ! Number of rows
"A" 0 2 1 ! Row label (A), A-B value, A-C value
"B" 2 0 1.4 ! Row label (B), B-A value, B-C value
"C" 1 1.4 0 ! Row label (C), C-A value, C-B value
```

Notice that:

- the row and column labels are identical.
- the matrix elements on the diagonal are zero.
- the matrix is symmetrical.

This text file can be read with the Read from file... command. Since a Dissimilarity object has the data structure of a square symmetrical TableOfReal, you could also start from an appropriate TableOfReal object and cast it to a Dissimilarity object.

Commands

Creation

- Confusion: To Dissimilarity...

Drawing

- Draw as numbers...
- Draw as squares...

Query

- **Get column mean (index)...**
- **Get column mean (label)...**
- **Get column stdev (index)...**
- **Get column stdev (label)...**
- Dissimilarity: Get additive constant

Modification

- Formula...
- **Set value...**
- **Remove column (index)...**
- **Insert column (index)...**
- **Set row label (index)...**
- **Set row label (label)...**
- **Set column label (index)...**
- **Set column label (label)...**

Multidimensional scaling analysis

- Dissimilarity: To Configuration (monotone mds)...
- Dissimilarity: To Configuration (i-spline mds)...
- Dissimilarity: To Configuration (interval mds)...
- Dissimilarity: To Configuration (ratio mds)...
- Dissimilarity: To Configuration (absolute mds)...
- Dissimilarity: To Configuration (kruskal)...
- Transformations
- Dissimilarity: To Distance...
- Dissimilarity: To Weight

Links to this page

- Confusion: To Dissimilarity (pdf)...
- Create INDSCAL Carroll & Wish example...
- Create letter R example...
- Dissimilarity & Configuration & Weight: Get stress...
- Dissimilarity & Configuration & Weight: To Configuration...
- Dissimilarity & Configuration: Draw regression (absolute mds)...
- Dissimilarity & Configuration: Draw regression (i-spline mds)...
- Dissimilarity & Configuration: Draw regression (interval mds)...
- Dissimilarity & Configuration: Draw regression (monotone mds)...
- Dissimilarity & Configuration: Draw regression (ratio mds)...
- Dissimilarity & Configuration: Draw Shepard diagram...
- Dissimilarity & Configuration: Get stress (absolute mds)...

- Dissimilarity & Configuration: Get stress (i-spline mds)...
 - Dissimilarity & Configuration: Get stress (interval mds)...
 - Dissimilarity & Configuration: Get stress (monotone mds)...
 - Dissimilarity & Configuration: Get stress (ratio mds)...
 - Dissimilarity & Configuration: To Configuration (absolute mds)...
 - Dissimilarity & Configuration: To Configuration (i-spline mds)...
 - Dissimilarity & Configuration: To Configuration (interval mds)...
 - Dissimilarity & Configuration: To Configuration (kruskal)...
 - Dissimilarity & Configuration: To Configuration (monotone mds)...
 - Dissimilarity & Configuration: To Configuration (ratio mds)...
 - Dissimilarity & Weight: To Configuration...
 - Dissimilarity & Weight: To Configuration...
 - INDSCAL analysis
 - Kruskal analysis
 - MDS models
 - Multidimensional scaling
 - Similarity: To Dissimilarity...
 - Weight
-

© *djmw*, March 27, 2001

Distance

One of the types of objects in PRAAT.

An object of type Distance represents distances between objects in a metrical space.

Creation

- Confusion: To Dissimilarity (pdf)...
- Dissimilarity: To Distance...

Links to this page

- Configuration: To Distance
- Distance & Configuration & Saliency: Get VAF...
- Distance & Configuration & Saliency: To Configuration (indscal)...
- Distance & Configuration: Get VAF...
- Distance & Configuration: To Configuration (indscal)...
- Distance: To Configuration (indscal)...
- Distance: To Configuration (ytl)...
- Distance: To ScalarProduct...
- INDSCAL analysis
- Multidimensional scaling

© *djmw*, November 24, 1997

MDS models

Multidimensional scaling (MDS) models are defined by specifying how given Dissimilarity data, δ_{ij} , are mapped into distances of an m -dimensional MDS Configuration \mathbf{X} . The mapping is specified by a *representation function*, $f: \delta_{ij} \rightarrow d_{ij}(\mathbf{X})$, which specifies how dissimilarities should be related to the distances. The MDS analysis tries to find the configuration (in a given dimensionality) whose distances satisfy f as closely as possible. This closeness is quantified by a badness-of-fit measure which is often called stress.

Representation functions

In the application of MDS we try to find a configuration \mathbf{X} such that the following relations are satisfied as well as possible:

$$f(\delta_{ij}) \approx d_{ij}(\mathbf{X})$$

The numbers that result from applying f on δ_{ij} are sometimes called disparities d_{ij} . In most applications of MDS, besides the configuration \mathbf{X} , also the function f is not completely specified, i.e., the exact parameters of f are unknown and must also be estimated during the analysis. If no particular f can be derived from a theoretical model, one often restricts f to a particular class of functions on the basis of the scale level of the dissimilarity data. If the disparities are related to the proximities by a specific parametric function we speak of *metric* MDS otherwise we speak of *ordinal* or *non-metric* MDS.

- *absolute* mds: $d'_{ij} = \delta_{ij}$
- No parameters need to be estimated.
- *ratio* mds: $d'_{ij} = b \cdot \delta_{ij}$,
- where the value of b can be estimated by a linear regression of d_{ij} on δ_{ij} .
- *interval* mds: $d'_{ij} = a + b \cdot \delta_{ij}$,
- where the values of a and b can be estimated by a linear regression of d_{ij} on δ_{ij} .
- *i-spline* mds: $d'_{ij} = i\text{-spline}(\delta_{ij})$,
- where $i\text{-spline}(\cdot)$ is a smooth monotonically increasing spline curve. The conceptual idea is that it is not possible to map all dissimilarities into disparities by one simple function.
- *monotone* mds: $d'_{ij} = \text{monotone}(\delta_{ij})$,
- where $\text{monotone}(\cdot)$ is restricted to be a monotonic function that preserves the order of the dissimilarities:

if $\delta_{ij} < \delta_{kl}$, then $d_{ij}(\mathbf{X}) < d_{kl}(\mathbf{X})$

If $\delta_{ij} = \delta_{kl}$ and no particular constraint is involved for $d_{ij}(\mathbf{X})$ and $d_{kl}(\mathbf{X})$ this is referred to as the *primary approach* to ties. The *secondary approach* to ties requires that if $\delta_{ij} = \delta_{kl}$, then also $d_{ij}(\mathbf{X}) =$

$d_{kl}(\mathbf{X})$.

More information on all aspects of multidimensional scaling can be found in: Borg & Groenen (1997) and Ramsay (1988).

The most important types and the conversions between them are shown in the following figure.

[sorry, no pictures yet in the web version of this manual]

- Kruskal analysis
- Multidimensional scaling

© *djmw*, January 8, 1998

Create letter R example...

Creates a Dissimilarity object that bears the name *R*. The dissimilarities in this object were chosen to be a monotone transformation of the distances between the 32 two-dimensional points that make up the capital letter **R**.

[sorry, no pictures yet in the web version of this manual]

All $32 \cdot (32-1)/2$ interpoint distances were subjected to the transformation:

$$dissimilarity_{ij} = distance_{ij}^2 + 5 + noiseRange \cdot \mathbf{u},$$

where \mathbf{u} is a uniform random variable between 0 and 1.

This example was chosen from Green, Carmone & Smith (1989).

Links to this page

- [Kruskal analysis](#)
- [Multidimensional scaling](#)

© djmw, December 1, 1997

Dissimilarity: To Configuration (monotone mds)...

A command that creates a Configuration object from a Dissimilarity object.

Dissimilarities δ_{ij} and disparities d'_{ij} are related by:

$$d'_{ij} \leq d'_{kl} \text{ if } \delta_{ij} \leq \delta_{kl}$$

Settings

Number of dimensions

determines the dimension of the configuration.

Primary or secondary approach to ties

When dissimilarities are equal, i.e., $\delta_{ij} = \delta_{kl}$, the primary approach imposes no conditions on the corresponding disparities d'_{ij} and d'_{kl} , while the *secondary* approach demands that also $d'_{ij} = d'_{kl}$.

Finding the optimal Configuration involves a minimization process:

Tolerance

When successive values for the stress differ less than *Tolerance* the minimization process stops.

Maximum number of iterations

Minimization stops after this number of iterations has been reached.

Number of repetitions

When chosen larger than 1, the minimalization process will be repeated, each time with another random start configuration. The configuration that results in minimum stress will be saved.

Links to this page

- [Dissimilarity & Configuration & Weight: To Configuration...](#)
- [Dissimilarity & Weight: To Configuration...](#)
- [Dissimilarity & Weight: To Configuration...](#)
- [Kruskal analysis](#)
- [Multidimensional scaling](#)

© djmw, April 7, 2004

Kruskal analysis

One of the MDS models in PRAAT.

You can perform a Kruskal-type multidimensional scaling only on objects of type Dissimilarity. Objects of other types first have to be converted to objects of Dissimilarity type.

Example

Convert a Dissimilarity object into a Configuration object.

- Dissimilarity: To Configuration (monotone mds)...
- choose appropriate parameters
- Dissimilarity & Configuration: Get stress (monotone mds)...
- choose stress-1 to obtain the value for the stress according to Kruskal.

How to get started

You can create an example Dissimilarity object with the Create letter R example... button which you can find under the **Multidimensional scaling** option in the **New** menu.

Links to this page

- [Multidimensional scaling](#)
- [Types of objects](#)

© djmw, December 1, 1997

Configuration: Draw...

Draws a projection of the selected Configuration on a coordinate plane.

Settings

X-coordinate, Y-coordinate

control the dimensions that will show in the plot.

xmin, xmax; ymin, ymax

range for horizontal and vertical axes, respectively.

garnish

when on, draws a bounding box with decoration.

Links to this page

- [Multidimensional scaling](#)

© *djmw*, April 7, 2004

stress

A badness-of-fit measure for the entire MDS representation.

Several measures exist.

Raw stress

$$\begin{aligned}\sigma_r(\mathbf{d}', \mathbf{X}) &= \sum_{i < j} w_{ij} (d'_{ij} - d_{ij}(\mathbf{X}))^2 \\ &= \sum_{i < j} w_{ij} d'^2_{ij} + \sum_{i < j} w_{ij} d_{ij}^2(\mathbf{X}) - 2 \sum_{i < j} w_{ij} d'_{ij} d_{ij}(\mathbf{X}) \\ &= \eta_{d'}^2 + \eta^2(\mathbf{X}) - 2\rho(\mathbf{d}', \mathbf{X})\end{aligned}$$

where the d'_{ij} are the disparities that are the result from the transformation of the dissimilarities, i.e., $f(\delta_{ij})$. Raw stress can be misleading because it is dependent on the normalization of the disparities. The following measure tries to circumvent this inconvenience.

Normalized stress

$$\sigma_n = \sigma_r / \eta_{d'}^2$$

This is the stress function that we minimize by iterative majorization. It goes back to De Leeuw (1977).

Kruskal's stress-1

$$\sigma_1 = \sqrt{(\sum_{i < j} w_{ij} (d'_{ij} - d_{ij}(\mathbf{X}))^2 / \sum_{i < j} w_{ij} d_{ij}^2(\mathbf{X}))^{1/2}}$$

In this measure, which is due to Kruskal (1964), stress is expressed in relation to the size of \mathbf{X} .

Kruskal's stress-2

$$\sigma_2 = \sqrt{(\sum_{i < j} w_{ij} (d'_{ij} - d_{ij}(\mathbf{X}))^2 / \sum_{i < j} w_{ij} (d_{ij}(\mathbf{X}) - \text{averageDistance})^2)^{1/2}}.$$

In general, this measure results in a stress value that is approximately twice the value for stress-1.

Relation between σ_1 and σ_n

When we have calculated σ_n for Configuration \mathbf{X} , disparities \mathbf{d}' and Weight \mathbf{W} we cannot directly use \mathbf{X} , \mathbf{d}' and \mathbf{W} to calculate σ_1 because the scale of \mathbf{X} is not necessarily optimal for σ_1 . We allow therefore a scale factor $b > 0$ and try to calculate $\sigma_1(\mathbf{d}', b \mathbf{X})$. We minimize the resulting expression for b and substitute the result back into the formula for stress, i.e.,

$$\sigma_1^2(\mathbf{d}', b \mathbf{X}) = (\eta_{d'}^2 + b^2 \eta^2(\mathbf{X}) - 2 b \rho(\mathbf{d}', \mathbf{X})) / b^2 \eta^2(\mathbf{X})$$

$$d\sigma_1^2(b) / db == 0, \text{ gives}$$

$$b = \eta_{d'}^2 / \rho$$

$$\sigma_1^2 = (1 - \rho^2 / (\eta_{d'}^2 \cdot \eta^2(\mathbf{X})))$$

This means that $\sigma_1 = \sqrt{\sigma_n}$.

Relation between σ_2 and σ_n

We can do the same trick as before for σ_2 :

$$\sigma_2^2(\mathbf{d}', b \mathbf{X}) = (\eta_{d'}^2 + b^2 \eta^2(\mathbf{X}) - 2 b \rho(\mathbf{d}', \mathbf{X})) / (b^2 \sum_{i < j} w_{ij} (d_{ij}(\mathbf{X}) - averageDistance)^2)$$

From which we derive:

$$\sigma_2 = \sqrt{((\eta_{d'}^2 \cdot \eta^2(\mathbf{X}) - \rho^2(\mathbf{d}', \mathbf{X})) / (\eta_{d'}^2 \cdot \sum_{i < j} w_{ij} (d_{ij}(\mathbf{X}) - averageDistance)^2))}$$

Links to this page

- Dissimilarity & Configuration & Weight: Get stress...
- Dissimilarity & Configuration & Weight: To Configuration...
- Dissimilarity & Configuration: Get stress (absolute mds)...
- Dissimilarity & Configuration: Get stress (i-spline mds)...
- Dissimilarity & Configuration: Get stress (interval mds)...
- Dissimilarity & Configuration: Get stress (monotone mds)...
- Dissimilarity & Configuration: Get stress (ratio mds)...

- Dissimilarity & Weight: To Configuration...
 - Dissimilarity & Weight: To Configuration...
 - Kruskal analysis
 - MDS models
 - Multidimensional scaling
 - Weight
-

© *djmw*, January 8, 1998

Dissimilarity & Configuration: Get stress (monotone mds)...

A command to obtain the stress value for the selected Dissimilarity and Configuration object.

Behaviour

We use stress formula's that are independent of the scale of the Configuration: you would have got the same stress value if you had pre-multiplied the selected Configuration with any number greater than zero.

Links to this page

- [Kruskal analysis](#)
- [Multidimensional scaling](#)

© *djmw*, January 19, 1998

Dissimilarity & Configuration: Draw Shepard diagram...

Draws the Shepard diagram. This is a scatterplot of the dissimilarities from the Dissimilarity object versus distances (as calculated from the Configuration).

Settings

Minimum proximity, Maximum proximity

minimum and maximum values for the proximities (horizontal axis).

Minimum distance, Maximum distance

minimum and maximum values for the distances (vertical axis).

Mark size (mm), Mark string

size and kind of the marks in the plot.

garnish

when on, draws a bounding box with decoration.

Links to this page

- [Multidimensional scaling](#)

© djmw, April 7, 2004

Dissimilarity & Configuration: Draw regression (monotone mds)...

Draws a scatterplot of the dissimilarities δ_{ij} from the selected Dissimilarity versus disparities d'_{ij} obtained from the monotone regression of distances d_{ij} from Configuration on the dissimilarities δ_{ij} .

Settings

Primary or *secondary* approach to ties

When dissimilarities are equal, i.e., $\delta_{ij} = \delta_{kl}$ the *primary* approach imposes no conditions on the corresponding distances d_{ij} and d_{kl} , while the *secondary* approach demands that also $d_{ij} = d_{kl}$.

Minimum proximity, *Maximum proximity*

minimum and maximum values for the proximities (horizontal axis).

Minimum distance, *Maximum distance*

minimum and maximum values for the distances (vertical axis).

Mark size (mm), *Mark string*

size and kind of the marks in the plot.

garnish

when on, draws a bounding box with decoration.

Links to this page

- [Multidimensional scaling](#)

© djmw, April 7, 2004

Weight

One of the types of objects in PRAAT.

An object of type Weight represents a matrix with weights w_{ij} .

An object of type Weight selected together with an object of type Dissimilarity can be used to make distinctions in the importance of the contribution of each individual dissimilarity δ_{ij} to stress and therefore to the final configuration. Weights can be used for instance to code for missing values, i.e., take $w_{ij} = 0$ if dissimilarity δ_{ij} is missing and $w_{ij} = 1$ if δ_{ij} is known.

Commands

Creation

- Dissimilarity: To Weight

Analysis

See Dissimilarity & Weight: To Configuration... for help on the following analysis items:

- **Dissimilarity & Weight: To Configuration (monotone mds)...**
- **Dissimilarity & Weight: To Configuration (i-spline mds)...**
- **Dissimilarity & Weight: To Configuration (interval mds)...**
- **Dissimilarity & Weight: To Configuration (ratio mds)...**
- **Dissimilarity & Weight: To Configuration (absolute mds)...**

Query

See Dissimilarity & Configuration & Weight: Get stress... for help on the following query items:

- **Dissimilarity & Configuration & Weight: Get stress (monotone mds)...**
- **Dissimilarity & Configuration & Weight: Get stress (i-spline mds)...**
- **Dissimilarity & Configuration & Weight: Get stress (interval mds)...**
- **Dissimilarity & Configuration & Weight: Get stress (ratio mds)...**
- **Dissimilarity & Configuration & Weight: Get stress (absolute mds)...**

Links to this page

- Dissimilarity & Configuration & Weight: To Configuration...
- Dissimilarity & Weight: To Configuration...
- Multidimensional scaling

© djmw, January 8, 1998

Dissimilarity: To Weight

Creates an object of type Weight for each selected Dissimilarity object.

The values in the weight matrix will be:

$$w_{ii} = 0$$
$$w_{ij} = 1 \text{ if } \delta_{ij} > 0$$

Links to this page

- [Multidimensional scaling](#)
-

© djmw, January 8, 1998

TableOfReal: Set value...

A command to change the value of one table cell in each selected TableOfReal object.

Arguments

Row number

the number of the row of the cell whose value you want to change.

Column number

the number of the column of the cell whose value you want to change.

New value

the value that you want the specified cell to have.

Links to this page

- [Multidimensional scaling](#)
-

© ppgb, January 5, 1998

individual difference scaling

The purpose of individual difference scaling is to represent objects, whose dissimilarities are given, as points in a metrical space. The distances in the space should be in accordance with the dissimilarities as well as is possible. Besides the configuration a Saliency matrix is calculated.

The basic Euclidean model is:

$$\delta_{ijk} \approx \left(\sum_{s=1..r} w_{ks} (x_{is} - x_{js})^2 \right)^{1/2}$$

Here δ_{ijk} is the (known) dissimilarity between *objects* i and j , as measured on *data source* k . The x 's are the *coordinates* of the objects in an r -dimensional space and the w 's are weights or saliencies. Because straight minimization of the expression above is difficult, one applies transformations on this expression. Squaring both sides gives the model:

$$\delta_{ijk}^2 \approx \sum_{s=1..r} w_{ks} (x_{is} - x_{js})^2$$

and the corresponding least squares loss function:

$$\sum_{k=1..numberOfSources} \sum_{i=1..numberOfPoints} \sum_{j=1..numberOfPoints} (\delta_{ijk}^2 - d_{ijk}^2)^2$$

This loss function is minimized in the (ratio scale option of the) **ALSCAL** program of Takane, Young & de Leeuw (1976).

The transformation used by Carroll & Chang (1970) in the INDSCAL model, transforms the data from each source into scalar products of vectors. For the dissimilarities:

$$\beta_{ijk} = -\{ \delta_{ijk}^2 - \delta_{i.k}^2 - \delta_{.jk}^2 + \delta_{..k}^2 \} / 2,$$

where dots replacing indices indicate averaging over the range of that index. In the same way for the distances:

$$z_{ijk} = -\{ d_{ijk}^2 - d_{i.k}^2 - d_{.jk}^2 + d_{..k}^2 \} / 2.$$

$$\beta_{ijk} \approx z_{ijk} = \sum_{s=1..numberOfDimensions} w_{ks} x_{is} x_{js}$$

Translated into matrix algebra, the equation above translates to:

$$B_k \approx Z_k = X W_k X',$$

where X is a $numberOfPoints \times numberOfDimensions$ configuration matrix, W_k , a non-negative $numberOfDimensions \times numberOfDimensions$ matrix with weights, and B_k the k^{th} slab of β_{ijk} .

This translates to the following INDSCAL loss function:

$$f(X, W_1, \dots, W_{\text{numberOfSources}}) = \sum_{k=1..numberOfSources} |B_k - XW_kX'|^2$$

- Distance: To Configuration (ytl)...
 - INDSCAL analysis
 - Multidimensional scaling
-

© *djmw*, May 2, 1997

INDSCAL analysis

A method for individual difference scaling analysis in PRAAT.

An INDSCAL analysis can be performed on objects of type Distance.

If you start with Dissimilarity objects you first have to transform them to Distance objects.

- Dissimilarity: To Distance...

If you start with a Confusion you can use:

- Confusion: To Dissimilarity (pdf)...

Examples

- Distance: To Configuration (indscal)...
- Perform an INDSCAL analysis on one or more objects of type Distance to calculate a Configuration.
- Distance & Configuration: To Configuration (indscal)...
- Perform an INDSCAL analysis on one or more objects of type Distance to calculate a Configuration. Use the selected Configuration object as the initial Configuration in the iteration process.

Algorithm

The function to be minimized in INDSCAL is the following:

$$f(X, W_1, \dots, W_{\text{numberOfSources}}) = \sum_{i=1..numberOfSources} |S_i - XW_iX'|^2$$

where X an unknown $\text{numberOfPoints} \times \text{numberOfDimensions}$ configuration matrix, the W_i are numberOfSources unknown diagonal $\text{numberOfDimensions} \times \text{numberOfDimensions}$ matrices with weights, often called saliences, and the S_i are known symmetric matrices with scalar products of dimension $\text{numberOfPoints} \times \text{numberOfPoints}$.

In the absence of an algorithm that minimizes f , Carroll & Chang (1970) resorted to the CANDECOMP algorithm, which instead of the function given above minimizes the following function:

$$g(X, Y, W_1, \dots, W_{\text{numberOfSources}}) = \sum_{i=1..numberOfSources} |S_i - XW_iY'|^2.$$

Carroll & Chang claimed that for most practical circumstances X and Y converge to matrices that will be columnwise proportional. However, INDSCAL does not only require symmetry of the solution, but also non-negativity of the weights. Both these aspects can not be guaranteed with the CANDECOMP algorithm.

Ten Berge, Kiers & Krijnen (1993) describe an algorithm that automatically satisfies symmetry because it solves f directly, and, also, can guarantee non-negativity of the weights. This algorithm proceeds as follows:

Let \mathbf{x}_h be the h -th column of X . We then write the function f above as:

$$f(\mathbf{x}_h, w_{1h}, \dots, w_{\text{numberOfSources } h}) = \sum_{i=1..numberOfSources} |S_{ih} - \mathbf{x}_h w_{ih} \mathbf{x}'_h|^2,$$

with S_{ih} defined as:

$$S_{ih} = (S_i - \sum_{j \neq h, j=1..numberOfDimensions} \mathbf{x}_j w_{ij} \mathbf{x}'_j).$$

Without loss of generality we may require that

$$\mathbf{x}'_h \mathbf{x}_h = 1$$

Minimizing f over \mathbf{x}_h is equivalent to minimizing

$$\sum_{i=1..numberOfSources} |S_{ih}|^2 - 2 \text{tr} \sum S_{ih} \mathbf{x}_h w_{ih} \mathbf{x}'_h + \sum w_{ih}^2$$

This amounts to maximizing

$$g(\mathbf{x}_h) = \mathbf{x}'_h (\sum w_{ih} S_{ih}) \mathbf{x}_h$$

subject to $\mathbf{x}'_h \mathbf{x}_h = 1$. The solution for \mathbf{x}_h is the dominant eigenvector of $(\sum w_{ih} S_{ih})$, which can be determined with the power method (see Golub & van Loan (1996)). The optimal value for the w_{ih} , given that all other parameters are fixed:

$$w_{ih} = \mathbf{x}'_h S_{ih} \mathbf{x}_h$$

In an alternating least squares procedure we may update columns of X and the diagonals of the W matrices in any sensible order.

Links to this page

- [Create INDSCAL Carroll & Wish example...](#)
- [Distance & Configuration & Saliience: To Configuration \(indscal\)...](#)
- [Distance: To Configuration \(ytl\)...](#)
- [Multidimensional scaling](#)
- [Types of objects](#)

© *djmw*, April 24, 1997

Create INDSCAL Carroll & Wish example...

Creates eight Dissimilarity objects that bear names "1" ... "8".

These objects contain the interpoint distances for a twodimensional 3×3 Configuration of points, labelled A, B, C, ... I. All Dissimilarity objects are based on the following underlying configuration.

[sorry, no pictures yet in the web version of this manual]

The eight sources weigh this configuration in the following manner:

[sorry, no pictures yet in the web version of this manual]

For each source, the distances were subjected to the transformation:

$$dissimilarity_{ij} = distance_{ij} + noiseRange \cdot \mathbf{u},$$

where \mathbf{u} is a uniform random variable between 0 and 1.

Now you can do the following for example:

Select all the Dissimilarity objects and choose To Distance....

Uncheck scale (add "additive constant").

Select all the Distance objects and choose To Configuration (indscale)....

and an INDSCAL analysis will be performed. In order to reproduce the saliences, you have to uncheck the "Normalize scalar products" option.

This example was adapted from Carroll & Wish (1974).

Links to this page

- [Multidimensional scaling](#)

© djmw, December 1, 1997

Dissimilarity: To Distance...

A command that creates a Distance object from a selected Dissimilarity object.

Settings

Scale

when on, the `additiveConstant` is determined, when off the `additiveConstant = 0`.

dissimilarities are transformed to distances according to:

$$distance_{ij} = dissimilarity_{ij} + additiveConstant.$$

Links to this page

- [Create INDSCAL Carroll & Wish example...](#)
- [Dissimilarity: To Configuration \(kruskal\)...](#)
- [INDSCAL analysis](#)
- [Multidimensional scaling](#)

© djmw, April 7, 2004

Salience

One of the types of objects in PRAAT.

Elements s_{ij} in the Salience matrix represent the importance of dimension j (in the Configuration) for data source i .

Commands

Creation, as a by-product of:

- Distance: To Configuration (indscal)...
- Distance: To Configuration (ytl)...

Links to this page

- Distance & Configuration: Get VAF...
 - individual difference scaling
 - Multidimensional scaling
-

© djmw, January 12, 1998

Pols et al. (1973)

L.C.W. Pols, H.R.C. Tromp & R. Plomp (1973), "Frequency analysis of Dutch vowels from 50 male speakers", *J.Acoust.Soc.Am.* **53**, 1093-1101.

Links to this page

- [Canonical correlation analysis](#)
- [Create formant table \(Pols & Van Nierop 1973\)](#)
- [Create TableOfReal \(Pols 1973\)...](#)
- [Discriminant analysis](#)

© *djmw*, April 26, 1999

Create TableOfReal (Pols 1973)...

A command to create a TableOfReal filled with the first three formant frequency values and (optionally) the levels from the 12 Dutchmonophthongal vowels as spoken in /h_t/ context by 50 male speakers.

The first three columns will contain the frequencies in Hz, the next three columns the levels in decibels below the overall SPL of the measured vowel segment. Each row will be labelled with its corresponding vowel symbol.

More details about these data and how they were measured can be found in the paper of Pols et al. (1973).

Links to this page

- Discriminant analysis
-

© djmw, April 26, 1999

TableOfReal: To Discriminant

A command that creates a Discriminant object from every selected TableOfReal object. Row labels in the table indicate group membership.

Algorithm

We solve for directions \mathbf{x} that are eigenvectors of the generalized eigenvalue equation:

$$\mathbf{B} \mathbf{x} - \lambda \mathbf{W} \mathbf{x} = 0,$$

where \mathbf{B} and \mathbf{W} are the between-groups and the within-groups sums of squares and cross-products matrices, respectively. Both \mathbf{B} and \mathbf{W} are symmetric matrices. Standard formula show that both matrices can also be written as a matrix product. The formula above then transforms to:

$$\mathbf{B}_1' \mathbf{B}_1 \mathbf{x} - \lambda \mathbf{W}_1' \mathbf{W}_1 \mathbf{x} = 0$$

The equation can be solved with the generalized singular value decomposition. This procedure is numerically very stable and can even cope with cases when both matrices are singular.

The a priori probabilities in the Discriminant will be calculated from the number of *training* vectors n_i in each group:

$$aprioriProbability_i = n_i / \sum_{k=1..numberOfGroups} n_k$$

Links to this page

- Discriminant analysis
- TableOfReal: To Configuration (lda)...

© djmw, January 4, 1999

Discriminant & TableOfReal: To Configuration...

A command to project each row in the selected TableOfReal onto a space spanned by the eigenvectors of the selected Discriminant.

Arguments

Number of dimensions

specifies the number of eigenvectors taken into account, i.e., determines the dimension of the resulting Configuration. When the default value (0) is given the resulting Configuration will have the maximum dimension as allowed by the number of eigenvectors in the selected Discriminant.

Precondition

The number of columns in the TableOfReal must equal the dimension of the eigenvectors in the Discriminant.

See also Eigen & TableOfReal: Project....

Links to this page

- [Discriminant analysis](#)

© djmw, April 7, 2004

TableOfReal: To Configuration (lda)...

Calculates a Configuration based on the Discriminant scores obtained from the selected TableOfReal. Row labels in the table indicate group membership.

Parameters

Number of dimensions

determines the number of dimensions of the resulting Configuration.

Algorithm

First we calculate the Discriminant from the data in the TableOfReal. See TableOfReal: To Discriminant for details.

The eigenvectors of the Discriminant determine the directions that the data in the TableOfReal will be projected unto.

Links to this page

- [Discriminant analysis](#)

© djmw, November 3, 1998

Discriminant: Draw sigma ellipses...

A command to draw for each group from the selected Discriminant an ellipse that covers part of the multivariate data.

Arguments

Number of sigmas

determines the data coverage.

Discriminant plane

When on, the selected *X* and *Y*-dimension will refer to the eigenvectors of the discriminant space, and, consequently, the projection of the hyper ellipsoid onto the space spanned by these eigenvectors will be drawn. When off, the selected *X* and *Y*-dimension will refer to the original dimensions.

Xmin, Xmax, Ymin, Ymax

determine the limits of the drawing area.

Label size

determines the size of the labels at the centre of the ellipse. No labels will be drawn when a value less than or equal to zero is chosen.

Links to this page

- [Discriminant analysis](#)
-

© djmw, April 7, 2004

Discriminant & TableOfReal: To ClassificationTable...

A command to use the selected Discriminant to classify each row from the selected TableOfReal. The newly created ClassificationTable will then contain the posterior probabilities of group membership.

Arguments

Pool covariance matrices

when on, all group covariance matrices are pooled and distances will be determined on the basis of only this pooled covariance matrix (see below).

Details

The posterior probabilities of group membership p_j for a vector \mathbf{x} are defined as:

$$p_j = p(j|\mathbf{x}) = \exp(-d_j^2(\mathbf{x})/2) / \sum_{k=1..numberOfGroups} \exp(-d_k^2(\mathbf{x})/2),$$

where d_i^2 is the generalized squared distance function:

$$d_i^2(\mathbf{x}) = ((\mathbf{x}-\mu_i)' \Sigma_i^{-1} (\mathbf{x}-\mu_i) + \ln \text{determinant}(\Sigma_i)) / 2 - \ln \text{aprioriProbability}_i$$

that depends on the individual covariance matrix Σ_i and the mean μ_i for group i .

When the covariances matrices are *pooled*, the squared distance function can be reduced to:

$$d_i^2(\mathbf{x}) = ((\mathbf{x}-\mu_i)' \Sigma^{-1} (\mathbf{x}-\mu_i) - \ln \text{aprioriProbability}_i,$$

and Σ is now the pooled covariance matrix.

The a priori probabilities normally will have values that are related to the number of *training* vectors n_i in each group:

$$\text{aprioriProbability}_i = n_i / \sum_{k=1..numberOfGroups} n_k$$

Links to this page

- Discriminant & Pattern: To Categories...
- Discriminant analysis

© *djmw*, April 7, 2004

Confusion

One of the types of objects in PRAAT.

An object of type Confusions represents a confusion matrix, with stimuli as row indices and responses as column indices. The entry at position $[i][j]$ represents the number of times that response j was given to the stimulus i .

Creating a Confusion from data in a text file

Suppose you have two objects A and B. In one way or another, you have acquired the following confusions: $\delta_{AA} = 6$, $\delta_{AB} = 2$, $\delta_{BA} = 1$, and $\delta_{BB} = 7$, where δ_{AB} represents the number of confusions between object A and object B.

You can create a simple text file like the following:

```
"ooTextFile" ! The line by which Praat can recognize your file
"Confusion" ! The line that tells Praat about the contents
2 "A" "B" ! Number of columns, and column labels
2 ! Number of rows
"A" 6 2 ! Row label (A), A-A value, A-B value
"B" 1 7 ! Row label (B), B-A value, B-B value
```

This text file can be read with the Read from file... command. Since a Confusion object has the data structure of a TableOfReal, you could also start from an appropriate TableOfReal object and cast it to a Confusion object.

Commands

Creation:

- Categories: To Confusion

Drawing

- Draw as numbers...
- Draw as squares...

Query

- Get column mean (index)...
- Get column mean (label)...
- Get column stdev (index)...
- Get column stdev (label)...
- Get fraction correct

Modification

- **Formula...**
- **Remove column (index)...**
- **Insert column (index)...**
- **Set row label (index)...**
- **Set row label (label)...**
- **Set column label (index)...**
- **Set column label (label)...**

Analysis:

- Confusion: To Similarity...
- Confusion: To Dissimilarity (pdf)...

Inside a Confusion

With Inspect you will see the following attributes:

numberOfRows

the number of stimuli.

numberOfColumns

the number of responses.

rowLabels

the names of the stimuli.

columnLabels

the names of the responses.

Links to this page

- Confusion: Condense...
- Confusion: To Dissimilarity...
- Confusion: To TableOfReal (marginals)
- Discriminant analysis
- Feedforward neural networks 2. Quick start
- Feedforward neural networks 3. FFNet versus discriminant classifier
- INDSCAL analysis

© djmw, May 1, 2001

Confusion: Get fraction correct

A query to ask the selected Confusion matrix for the fraction of correct classifications.

The "fraction correct" is defined as the quotient of the number of correct classifications and the total number of entries in the matrix.

Cells with correct classifications have equal row and column labels.

Links to this page

- [Discriminant analysis](#)

© *djmw*, February 25, 2000

Jackknife

A technique for estimating the bias and standard deviation of an estimate.

Suppose we have a sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and wish to estimate the bias and standard error of an estimator $\hat{\theta}$. The jackknife focuses on the samples that leave out one observation at a time: the i -th jackknife sample consists of the data with the i -th observation removed.

Links to this page

- [Discriminant analysis](#)

© *djmw*, November 3, 2003

Bootstrap

The bootstrap data points are a random sample of size n drawn *with* replacement from the sample (x_1, \dots, x_n) . This means that the bootstrap data set consists of members of the original data set, some appearing zero times, some appearing once, some appearing twice, etc.

More information can be found in Efron & Tibshirani (1993).

Links to this page

- [Discriminant analysis](#)

© djmw, November 3, 2003

Canonical correlation analysis

This tutorial will show you how to perform canonical correlation analysis with PRAAT.

1. Objective of canonical correlation analysis

In canonical correlation analysis we try to find the correlations between two data sets. One data set is called the *dependent* set, the other the *independent* set. In PRAAT these two sets must reside into one TableOfReal object. The lower numbered columns of this table will then be interpreted as the dependent part, the rest of the columns as the independent part. The dimension of, i.e., the number of columns in, the dependent part may not exceed the dimension of the independent part.

As an example, we will use the dataset from Pols et al. (1973) with the frequencies and levels of the first three formants from the 12 Dutch monophthongal vowels as spoken in /h_t/ context by 50 male speakers. We will try to find the canonical correlation between formant frequencies (the *dependent* part) and levels (the *independent* part). The dimension of both groups of variates is 3. In the introduction of the discriminant analysis tutorial you can find how to get these data, how to take the logarithm of the formant frequency values and how to standardize them. The following script summarizes:

```
Create TableOfReal (Pols 1973)... yes
Formula... if col < 4 then log10 (self) else self endif
Standardize columns
```

Before we start with the *canonical* correlation analysis we will first have a look at the *Pearson* correlations of this table and calculate the Correlation matrix. It is given by:

```
      F1 F2 F3 L1 L2 L3
F1 1 -0.338 0.191 0.384 -0.505 -0.014
F2 -0.338 1 0.190 -0.106 0.526 -0.568
F3 0.191 0.190 1 0.113 -0.038 0.019
L1 0.384 -0.106 0.113 1 -0.038 0.085
L2 -0.505 0.526 -0.038 -0.038 1 0.128
L3 -0.014 -0.568 0.019 0.085 0.128 1
```

The following script summarizes:

```
select TableOfReal pols_50males
To Correlation
Draw as numbers... 1 0 decimal 3
```

The correlation matrix shows that high correlations exist between some formant frequencies and some levels. For example, the correlation coefficient between F2 and L2 equals 0.526.

In a canonical correlation analysis of the dataset above, we try to find the linear combination u_1 of F_1 , F_2 and F_3 that correlates maximally with the linear combination v_1 of L_1 , L_2 and L_3 . When we have found

these u_1 and v_1 we next try to find a new combination u_2 of the formant frequencies and a new combination v_2 of the levels that have maximum correlation. These u_2 and v_2 must be uncorrelated with u_1 and v_1 . When we express the above with formulas we have:

$$u_1 = y_{11}F_1 + y_{12}F_2 + y_{13}F_3$$

$$v_1 = x_{11}L_1 + x_{12}L_2 + x_{13}L_3$$

$$\rho(u_1, v_1) = \text{maximum}, \rho(u_2, v_2) = \text{submaximum},$$

$$\rho(u_2, u_1) = \rho(u_2, v_1) = \rho(v_2, v_1) = \rho(v_2, u_1) = 0,$$

where the $\rho(u_i, v_i)$ are the correlations between the **canonical variates** u_i and v_i and the y_{ij} 's and x_{ij} 's are the **canonical coefficients** for the dependent and the independent variates, respectively.

2. How to perform a canonical correlation analysis

Select the TableOfReal and choose from the dynamic menu the option To CCA.... This command is available in the "Multivariate statistics" action button. We fill out the form and supply 3 for *Dimension of dependent variate*. The resulting CCA object will bear the same name as the TableOfReal object. The following script summarizes:

```
select TableOfReal pols_50males
To CCA... 3
```

3. How to get the canonical correlation coefficients

You can get the canonical correlation coefficients by queries of the CCA object. You will find that the three canonical correlation coefficients, $\rho(u_1, v_1)$, $\rho(u_2, v_2)$ and $\rho(u_3, v_3)$ are approximately 0.86, 0.53 and 0.07, respectively. The following script summarizes:

```
cc1 = Get correlation... 1
cc2 = Get correlation... 2
cc3 = Get correlation... 3
println cc1 = 'cc1', cc2 = 'cc2', cc3 = 'cc3'
```

4. How to obtain canonical scores

Canonical **scores**, also named canonical variates, are the linear combinations:

$$u_i = y_{i1}F_1 + y_{i2}F_2 + y_{i3}F_3, \text{ and,}$$

$$v_i = x_{i1}L_1 + x_{i2}L_2 + x_{i3}L_3,$$

where the index i runs from 1 to the number of correlation coefficients.

You can get the canonical scores by selecting a CCA object together with the TableOfReal object and choose To TableOfReal (scores)...

When we now calculate the **Correlation** matrix of these canonical variates we get the following table:

	u1	u2	u3	v1	v2	v3
u1	1	.	.	0.860	.	.
u2	.	1	.	0.531	.	.
u3	.	.	1	.	0.070	.
v1	0.860	.	.	1	.	.
v2	.	0.1	.	.	1	.
v3	.	.	0.070	.	.	1

The scores with a dot are zero to numerical precision. In this table the only correlations that differ from zero are the canonical correlations. The following script summarizes:

```
select CCA pols_50males
plus TableOfReal pols_50males
To TableOfReal (scores)... 3
To Correlation
Draw as numbers if... 1 0 decimal 2 abs(self) > 1e-14
```

5. How to predict one dataset from the other

CCA & TableOfReal: Predict...

Additional information can be found in Weenink (2003).

Links to this page

- CCA

Discriminant

One of the types of objects in PRAAT.

An object of type Discriminant represents the discriminant structure of a multivariate data set with several groups. This discriminant structure consists of a number of orthogonal directions in space, along which maximum separability of the groups can occur.

Commands

Creation:

- Discriminant analysis tutorial
- TableOfReal: To Discriminant

Drawing

- Draw eigenvalues...
- Draw eigenvector...
- Draw sigma ellipses...

Links to this page

- Discriminant & Pattern: To Categories...
- Discriminant & SSCP: Project
- Discriminant & TableOfReal: To ClassificationTable...
- Discriminant & TableOfReal: To Configuration...
- Discriminant: Extract pooled within-groups SSCP
- Discriminant: Extract within-group SSCP...
- Discriminant: Get concentration ellipse area...
- Discriminant: Get confidence ellipse area...
- Discriminant: Get contribution of component...
- Discriminant: Get partial discrimination probability...
- Discriminant: Get Wilks' lambda...
- epoch
- Feedforward neural networks 3. FFNet versus discriminant classifier
- TableOfReal: To Configuration (lda)...

© djmw, November 3, 1998

Feedforward neural networks 3. FFNet versus discriminant classifier

You might want to compare the FFNet classifier with a discriminant classifier. Unlike the FFNet, a discriminant classifier does not need any iterative procedure in the learning phase and can be used immediately after creation for classification. The following three simple steps will give you the confusion matrix based on discriminant analysis:

1. Select the Pattern and the Categories together and choose **To Discriminant**. A newly created Discriminant will appear.
2. Select the Discriminant and the Pattern together and choose **To Categories...**. A newly created Categories will appear.
3. Select the two appropriate Categories and choose To Confusion. A newly created Confusion will appear. After pushing the Info button, the info window will show you the fraction correct.

See also the Discriminant analysis tutorial for more information.

Links to this page

- [Feedforward neural networks](#)
- [Feedforward neural networks 2. Quick start](#)

© *djmw*, April 26, 2004

PostScript settings...

One of the commands in the File menus of many windows. The PostScript settings influence Printing and writing to Encapsulated PostScript files.

Arguments

Allow direct PostScript printing (Windows and Macintosh only)

this determines whether Praat prints explicit PostScript commands to your printer if it is a PostScript printer. This is what you will usually want. However, if you find that some of the options that you choose in the printing dialog seem not to be supported (e.g. scaling, printing two-up...), you may switch this off; Praat will then send native Windows or Macintosh drawing commands, which the printer driver will try to translate to PostScript. If your printer does not support PostScript, this switch is ignored. On Unix, this switch is superfluous, since all printing is done directly in PostScript.

Grey resolution

you can choose from two image qualities:

- o the *finest* quality for grey plots (106 spots per inch), which gives the best results directly from the printer;
- o a *photocopyable* quality, which has fewer spots per inch (85) and gives the best results after photocopying.

Your choice of the grey resolution influences direct PostScript printing and writing to Encapsulated PostScript files.

Paper size (Unix only)

you can choose from A4 (210 × 297 mm), A3 (297 × 420 mm) or US Letter (8.5 × 11"). This choice applies to Unix only; on Windows, you choose the paper size in the **Print...** dialog; on Macintosh, you choose the paper size in the **Page setup...** dialog.

Orientation (Unix only)

you can choose between *portrait* (e.g., 297 mm high and 210 mm wide) and *landscape* (e.g., 210 mm high and 297 mm wide). This choice applies to Unix only; on Windows, you choose the orientation in the **Print...** dialog; on Macintosh, you choose the orientation in the **Page setup...** dialog.

Magnification (Unix only)

the relative size with which your picture will be printed; normally 1.0. This choice applies to Unix only; on Windows, you choose the scaling in the **Print...** dialog; on Macintosh, you choose the scaling in the **Page setup...** dialog.

Print command (Unix only)

When printing on Unix, a temporary PostScript[®] file is created in the "/usr/tmp" directory; it will have a name like "picXXXXXX", and is automatically removed after printing. This file is sent to the printer with the print command, which will often look like `lp -c %s`, where %s stands for the file name.

Links to this page

- [Picture window](#)
 - [Write to EPS file...](#)
-

© *ppgb*, October 10, 2000

Write to EPS file...

One of the commands in the File menu of the Picture window.

It saves the picture to an Encapsulated PostScript (EPS) file, so that you can use high-quality graphics in your word-processor documents.

The EPS picture is saved in the grey resolution that you specified with PostScript settings.... On Unix and Windows, the file will contain PostScript text only; on the Macintosh, the file will consist of PostScript text (for the printer) plus a bitmapped screen preview (in the resource fork).

You can insert an EPS file into a Microsoft® Word™ document by choosing **File...** or **Picture...** from the **Insert** menu (in Word 5.1). Word will create a picture with the same size as your original selection. If you created this file on a Macintosh, you will see the screen preview; if you created it elsewhere, you will only see the file name and the date and time of creation. In either case, you will see the high-quality PostScript version when you print. Though all the picture data was written to the file, only the selected part will be visible and printed.

Previewing

On SGI or Sun, you can view the EPS file by double-clicking on it.

Links to this page

- [Copy to clipboard](#)
- [OT learning 2.7. Tableau pictures](#)
- [Printing](#)
- [Write to Mac PICT file...](#)
- [Write to Windows metafile...](#)

© ppgb, October 10, 2000

Copy to clipboard

A command in the File menu of the Picture window on Windows[®] and Macintosh[®].

It copies the selected part of the picture to the clipboard. You can then 'Paste' it into any Windows or Macintosh program that knows pictures. On Windows, most programs know how to deal with this high-resolution clipboard. On Macintosh, drawing programs like MacDraw[™] can handle it, but Microsoft[®] Word[™] unfortunately rounds down the high resolution to screen pixels.

Behaviour

Though all the picture data will be written to the clipboard, only the selected part will be visible.

Usage

If you have a PostScript printer, you will want to use Write to EPS file... instead. If the clipboard is too large, use Write to Mac PICT file... or Write to Windows metafile... instead.

Links to this page

- [Printing](#)
-

© ppgb, October 16, 2000

Write to Mac PICT file...

A command in the File menu of the Picture window on Macintosh[®].

It saves the selected part of the picture in a Macintosh[®] extended PICT2 format. An extended PICT2 file can be read by many Mac drawing programs, like MacDraw[™], although Microsoft[®] Word[™] unfortunately rounds the high resolution down to screen pixels.

Behaviour

Though all the picture data will be written to the file, only the selected part will be visible later.

Usage

You will not use this command very often, because it is usually easier to copy the selection to the clipboard with the Copy to clipboard command, and 'Paste' it into your drawing program. If the clipboard is too large for the drawing program to read, you may use a PICT file.

If you have a PostScript printer, you would use Write to EPS file... instead for best printing results.

Links to this page

- [Printing](#)

© ppgb, October 16, 2000

Write to Windows metafile...

A command in the File menu of the Picture window on Microsoft® Windows™.

It saves the selected part of the picture in an "enhanced metafile" (.EMF) format. An enhanced metafile can be read by many Windows programs, like Adobe® Illustrator™ or Microsoft® Word™.

Behaviour

Though all the picture data will be written to the file, only the selected part will be visible later.

Usage

You will not use this command very often, because it is usually easier to copy the selection to the clipboard with the Copy to clipboard command, and 'Paste' it into Microsoft Word or Adobe Illustrator.

If you have a PostScript printer, you would use Write to EPS file... instead for best printing results.

Links to this page

- [Printing](#)

© ppgb, October 16, 2000

Print...

One of the commands in the File menu of the Picture window.

With this command, you send your entire picture immediately to the printer. See the Printing tutorial for details.

Links to this page

- [OT learning 2.7. Tableau pictures](#)
-

© *ppgb*, October 10, 2000

ScriptEditor

An aid to scripting.

The **ScriptEditor** is a text editor that allows you to edit, save, and run any Praat script. You could type such a script from scratch, but it is easier to use the History mechanism, which automatically records all your commands and mouse clicks, and which can paste these directly into a **ScriptEditor**.

To add a script as a button to a fixed or dynamic menu, use Add to fixed menu... or Add to dynamic menu... from the File menu.

Example 1

In this example, we create a fixed button that will play a 1-second sine wave with a specified frequency.

First, we create a ScriptEditor by choosing New Praat script from the Control menu. Then, we choose Clear history from the Edit menu in the ScriptEditor. We then perform some actions that will create a sine wave, play it, and remove it:

1. Choose **Create Sound...** from the New menu and type the formula of a sine wave (i.e. remove the "randomGauss" term).
2. Click **Play** in the dynamic menu.
3. Click the fixed **Remove** button.

We then choose Paste history from the Edit menu in the ScriptEditor (or type Command-H). The text will now contain at least the following lines (delete any other lines):

```
Create Sound... sine 0 1 22050 1/2*sin(2*pi*377*x)
Play
Remove
```

We can run this script again by choosing **Run** from the **Run** menu (or typing Command-R). However, this always plays a sine with a frequency of 377 Hz, so we will add the variable "Frequency" to the script, which then looks like:

```
form Play a sine wave
  positive Frequency
endform
Create Sound... sine'frequency' 0 1 22050 1/2*sin(2*pi*frequency*x)
Play
Remove
```

When we choose **Run**, the ScriptEditor will ask us to supply a value for the "Frequency" variable. We can now play 1-second sine waves with any frequency. Note that the name of the temporary Sound is now "sine356" if *Frequency* is "356": any occurrence of the string "'Frequency'" is replaced with the supplied argument.

It is advisable to supply a standard value for each argument in your script. If the duration should be variable, too, the final script could look like:

```
form Play a sine wave
  positive Frequency 440
  positive Duration 1
endform
Create Sound... sine'frequency' 0 'Duration' 22050
0.9*sin(2*pi*frequency*x)
Play
Remove
```

When you run this script, the ScriptEditor will ask you to supply values for the two variables, but the values "440" and "1" are already visible in the form window, so that you will get a sensible result if you just click **OK**.

If this script is useful to you, you may want to put a button for it in the New menu, in the **Create Sound** submenu:

1. Save the script to a file, with **Save** from the **File** menu. The file name that you supply, will be shown in the title bar of the ScriptEditor window.
2. Choose Add to fixed menu... from the **File** menu. Supply **Objects** for the *window*, **New** for the *menu*, "Play sine wave..." for the *command*, **Create Sound...** for *after command*, and "1" for the depth (because it is supposed to be in a submenu); the *script* argument has already been set to the file name that you supplied in step 1.
3. Click **OK** and ensure that the button has been added in the New menu. This button will still be there after you leave the program and enter it again; to remove it from the menu, use the ButtonEditor.

Example 2

In this example, we will create a shortcut for the usual complex pitch-analysis command.

First, we perform the required actions:

1. Select a Sound object.
2. Click **To Pitch...** and set the arguments to your personal standard values.
3. Click **OK**. A new **Pitch** object will appear.

We then paste the history into the ScriptEditor, after which this will contain at least a line like (delete all the other lines):

```
To Pitch... 0.01 150 900
```

You can run this script only after selecting one or more Sound objects.

If this script is useful to you, you may want to put a button for it in the dynamic menu:

1. Save the script to a file, with **Save** from the **File** menu.
2. Choose Add to dynamic menu... from the **File** menu. Supply "Sound" for *class1* (because the button is supposed to be available only if a Sound is selected), "0" for *number1* (because the command is supposed to work for any number of selected Sound objects), "To Pitch (child)" for the *command*, "To Spectrum" for *after command*, and "0" for the depth (because it is not supposed to be in a submenu); the *script* argument has already been set to the file name that you supplied in step 1.
3. Click **OK** and ensure that the button is clickable if you select one or more Sound objects. This button will still be available after you leave the program and enter it again; to remove it from the dynamic menus, use the ButtonEditor.

Links to this page

- [Add action command...](#)
- [Add menu command...](#)
- [Buttons file](#)
- [Hidden commands](#)
- [Object window](#)
- [Open Praat script...](#)
- [Scripting 1. My first script](#)
- [Scripting 2. Arguments to commands](#)
- [Scripting 7.2. Scripting an editor from within](#)
- [What's new?](#)

© ppgb, April 14, 2004

Scripting 1. My first script

Suppose that you want to create a script that allows you to play a selected Sound object twice. You first create an empty script, by choosing New Praat script from the Control menu. A ScriptEditor will appear on your screen. In this editor, you type

```
Play  
Play
```

Now select a **Sound** in the object menu. As you expect from selecting a Sound, a **Play** button will appear in the dynamic menu. If you now choose **Run** in the ScriptEditor, the sound will be played twice.

What commands can I put into my script?

In the above example, you could use the "Play" command because that was the text on a button currently available button in the dynamic menu. Apart from these selection-dependent (dynamic) commands, you can also use all fixed commands in the menus of the Object window and the Picture window. For how to proceed with commands that need *arguments* (i.e. the commands that end in "..." and present a settings window), see \SS2.

Faster ways of getting a script text

Instead of manually typing the command lines, as described above, it may be easier to create a script text from a *macro recording*, with the History mechanism. This allows you to create script texts without typing. For instance, if you choose Clear history in your ScriptEditor, then click the **Play** button twice (after selecting a Sound), and then choose Paste history, your new script text contains exactly two lines that read "Play".

To edit a script that is already contained in a file on disk, use Open Praat script....

How to run a script

You can run scripts from the ScriptEditor. If you will have to use the script very often, it is advisable to create a button for it in the fixed menu or in a dynamic menu. See the ScriptEditor manual page.

On Unix and Windows, you can also run scripts from the command line. See Scripting 6.9. Calling from the command line.

Links to this page

- Scripting
-

© *ppgb*, April 14, 2004

Scripting 2. Arguments to commands

This chapter describes how your script should represent commands that need *arguments*, i.e. commands whose button has a title that ends in "...". Clicking such a button normally presents a *settings window*, which asks the user to supply *arguments* (settings) and then press the OK button. In a script, all of these arguments should be supplied on the same line as the command, in the same order as in the settings window, counted from top to bottom.

Numeric arguments

The command **Draw line...** from the World menu in the Picture window, normally presents a settings window that asks the user to supply values for the numeric parameters *From x*, *From y*, *To x*, and *To y* (from top to bottom). To draw a line from the point (0, 0.5) to the point (1, 1), the values of the arguments would be 0, 0.5, 1, and 1, respectively. In a script, the command would read

```
Draw line... 0 0.5 1 1
```

Check buttons

In a script, you represent a *check button* (yes/no choice) by "true" or "false", which may be abbreviated to "t" etc., or written with a capital; "yes" or "no" can also be used, as well as "1" or "0":

```
Marks left every... 1.0 100 yes yes no
```

Radio boxes

You represent a *radio box* (multiple choice) by the text on the button that you want to select:

```
Print picture to PostScript printer... Finest A4 Portrait 1.0
```

If the choice (in a non-final argument) contains spaces, it should be enclosed within double quotes:

```
Print picture to PostScript printer... Finest "US Letter" Portrait 1.0
```

Using these double quotes is necessary because spaces are normally used for separating the arguments. Putting quotes around "US Letter" ensures that the second argument consists of both words. If you forget the quotes, the interpreter will think that the second argument is "US", the third argument is "Letter", and the fourth argument is the rest of the line, namely "Portrait 1.0", and the command will fail with the message

```
`Paper size' cannot have the value "US".
```

Text arguments

A text argument at the end should be typed without any quotes:

```
Text... 400 Centre 1.5 Bottom This is the summit
```

Though spaces separate the arguments, the last argument is always written without surrounding quotes, even if it contains spaces. This is possible because the interpreter knows that the **Text...** command takes five arguments. The fifth argument is simply everything that follows the first four arguments. With this strategy, you will not suffer from *quote paranoia*, which would occur if your text itself contains quotes:

```
Text... 0.5 Centre 0.5 Half "hello world"
```

In this case, the quotes around "hello world" will be written to the Picture window as well.

For these reasons, nearly all texts in settings windows appear in the last (bottom) field, so that it can be scripted without using quotes. The exceptions are those few commands that have more than a single text parameter. A text argument that is not at the end should then be enclosed in double quotes if it is empty or contains spaces (otherwise, the quotes are optional):

```
Add menu command... Objects New "Create a Bicycle..." "" 0  
/u/miep/createBicycle.script
```

If you want to include a double quote in such a text, use *two* double quotes:

```
Add menu command... Objects New "Create a ""Bicycle""..." (etc.)
```

This doubling of quotes is what you want to avoid in your scripts, especially if your texts come from the expansion of string variables.

File arguments

The commands from the Read and Write menus, and several other commands whose names start with Read, Open, or Write, present a *file selector window* instead of a typical Praat settings window. File selector windows ask the user to supply a single argument: the file name.

In a script, you can supply the complete *path*, including the directory (folder) hierarchy and the name of the file. On Unix, it goes like this (if you are user "miep"):

```
Read from file... /home/miep/sounds/animals/miauw.aifc
```

On Macintosh, it goes like this (supposing your hard drive is called "Macintosh HD"):

```
Read from file... Macintosh HD:Sounds:Animals:miauw.aifc
```

If your Sounds folder is on the desktop, it would be:

Read from file... Macintosh HD:Desktop Folder:Sounds:Animals:miauw.aifc

If your Sounds folder is on a Zip disk called Miep, it would be:

Read from file... Miep:Sounds:Animals:miauw.aifc

In Windows, you may type:

Read from file... C:\Sounds\Animals\miauw.aifc

Instead of these complete path names, you can use *relative* path names. These are taken as relative to the directory in which your script resides.

On Unix, a relative path name starts without a "/". So if your script is */home/miep/sounds/analysis.praat*, the above line could be

Read from file... animals/miauw.aifc

On Macintosh, a relative (*partial*) path name starts with a colon. So if your script is Macintosh HD:Sounds:Analysis.praat, the sound file is read by

Read from file... :Animals:miauw.aifc

In Windows, a relative path name starts without a backslash. So if your script is C:\Sounds\Analysis.praat, the sound file is read by

Read from file... Animals\miauw.aifc

Finally, your script may not be in a directory *above* the directory from which you like to read, but in a directory on the side, like */home/miep/scripts*, Macintosh HD:Scripts, or C:\Scripts. The commands would then read

Read from file... ../animals/miauw.aifc

Read from file... ::Animals:miauw.aifc

Read from file... ..\Animals\miauw.aifc

How to supply arguments automatically

If you dislike manually copying arguments from settings windows into your script, you can use the history mechanism to automate this process: choose Clear history from the Edit menu in your ScriptEditor, click your command button, edit the arguments, and click OK. The command will be executed. Then choose Paste history, and the command line, including the arguments (with correct quotes), appears in a the ScriptEditor. You can build a new script on the basis of this line.

Links to this page

- [Scripting](#)
 - [Scripting 1. My first script](#)
-

© *ppgb*, April 14, 2004

Scripting 3. Layout

This chapter handles the way you use white space, comments, and continuation lines in a Praat script.

White space

All white space (spaces and tabs) at the beginning of lines is ignored. This means that you can use indenting to make your script readable. You are advised to use three spaces for each level of indenting:

```
sum = 0
for i to 10
  for j to 10
    sum += i^j
  endfor
endfor
echo The sum of the products is 'sum'
```

Lines that are empty or consist solely of white space, are also ignored.

Comments

Comments are lines that start with `!`, `#`, or `;`. These lines are ignored when your script is running:

```
# Create 1 second of a sine wave with a frequency of 100 Hertz,
# sampled at 22050 Hz:
Create Sound... sine 0 1 22050 sin (2*pi*100*x)
```

Continuation lines

There is normally one line per command, and one command per line.

But you can chop up long lines by using continuation lines that start with three dots ("`...`"). You will normally want to follow this *ellipsis* with a space, unless you want to concatenate the parts of a long word:

```
Viewport... 0 10 0 4
Text top... yes It's a long way to Llanfairpwllgwyngyll
  ...gogerychwyndrobwlllantysiliogogoch,
  ... unless you start from Tyddyn-y-felin.
```

Links to this page

- [Scripting](#)

© *ppgb*, June 24, 2002

Scripting 4. Object selection

This chapter is about how to select objects from your script, and how to find out what objects are currently selected.

Selecting objects

To simulate the mouse-clicked and dragged selection in the list of objects, you have the following commands:

select *object*

selects one object, and deselects all others. If there are more objects with the same name, the most recently created one (i.e., the one nearest to the bottom of the list of objects) is selected:

```
select Sound hallo  
Play
```

plus *object*

adds one object to the current selection.

minus *object*

removes one object from the current selection.

select all

selects all objects:

```
select all  
Remove
```

In the Praat shell, newly created objects are automatically selected. This is also true in scripts:

```
! Generate a sine wave, play it, and draw its spectrum.  
Create Sound... sine377 0 1 10000 0.9 * sin (2*pi*377*x)  
Play  
To Spectrum  
! Draw the Spectrum:  
Draw... 0 5000 20 80 yes  
! Remove the created Spectrum and Sound:  
plus Sound sine377  
Remove
```

Instead of by name, you can also select objects by their sequential ID:

```
select 43
```

This selects the 43rd object that you created since you started the program (see below).

Querying selected objects

You can get the name of a selected object into a string variable. For instance, the following reads the name of the second selected Sound (as counted from the top of the list of objects) into the variable *name\$*:

```
name$ = selected$ ("Sound", 2)
```

If the Sound was called "Sound hallo", the variable *name\$* will contain the string "hallo". To get the name of the topmost selected Sound object, you can leave out the number:

```
name$ = selected$ ("Sound")
```

Negative numbers count from the bottom. Thus, to get the name of the bottom-most selected Sound object, you say

```
name$ = selected$ ("Sound", -1)
```

You would use **selected\$** for drawing the object name in a picture:

```
Draw... 0 0 0 0 yes  
name$ = selected$ ("Sound")  
Text top... no This is sound 'name$'
```

For identifying previously selected objects, this method is not very suitable, since there may be multiple objects with the same name:

```
# The following two lines are OK:  
soundName$ = selected$ ("Sound", -1)  
pitchName$ = selected$ ("Pitch")  
# But the following line is questionable, since it doesn't  
# necessarily select the previously selected Sound again:  
select Sound 'soundName$'
```

Instead of this error-prone approach, you should get the object's unique ID. The correct version of our example becomes:

```
sound = selected ("Sound", -1)  
pitch = selected ("Pitch")  
# Correct:  
select sound
```

To get the number of selected objects into a variable, use

```
numberOfSelectedSounds = numberOfSelected ("Sound")
```

Links to this page

- [Scripting](#)
-

© *ppgb*, February 22, 2004

Scripting 5. Language elements

In a Praat script, you can use variables, expressions, and functions, of numeric as well as string type, and most of the control structures known from other procedural computer languages. The way the distinction between numbers and strings is made, may remind you of the programming language Basic.

- Scripting 5.1. Variables (numeric, string, copy, expansion)
- Scripting 5.2. Formulas (numeric, string)
- Scripting 5.3. Jumps (if, then, elsif, else, endif)
- Scripting 5.4. Loops (for/endfor, while/endwhile, repeat/until)
- Scripting 5.5. Procedures (call, procedure)
- Scripting 5.6. Arrays
- Scripting 5.7. Including other scripts
- Scripting 5.8. Quitting (exit)

Links to this page

- [Scripting](#)
-

© ppgb, December 18, 2002

Scripting 5.1. Variables

In a Praat script, you can use numeric variables as well as string variables.

Numeric variables

Numeric variables contain integer numbers between -1,000,000,000,000,000 and +1,000,000,000,000,000 or real numbers between -10^{308} and $+10^{308}$. The smallest numbers lie near -10^{-308} and $+10^{-308}$.

You can use *numeric variables* in your script:

```
variable = formula
```

evaluates a numeric formula and assign the result to a variable.

Example:

```
length = 10  
Draw line... 0 length 1 1
```

Names of numeric variables must start with a lower-case letter, optionally followed by a sequence of letters, digits, and underscores.

String variables

You can also use *string variables*, which contain text:

```
title$ = "Dutch nasal place assimilation"
```

As in the programming language Basic, the names of string variables end in a dollar sign.

Variable substitution

Existing variables are substituted when put between quotes:

```
x = 99  
x2 = x * x  
echo The square of 'x' is 'x2'.
```

This will write the following text to the Info window:

```
The square of 99 is 9801.
```

You can reduce the number of digits after the decimal point by use of the colon:

```
root = sqrt (2)  
echo The square root of 2 is approximately 'root:3'.
```

This will write the following text to the Info window:

```
The square root of 2 is approximately 1.414.
```

By using ":0", you round to whole values:

```
root = sqrt (2)
echo The square root of 2 is very approximately 'root:0'.
```

This will write the following text to the Info window:

```
The square root of 2 is very approximately 1.
```

By using ":3%", you give the result in a percent format:

```
jitter = 0.0156789
echo The jitter is 'jitter:3%'.
```

This will write the following text to the Info window:

```
The jitter is 1.568%.
```

The number 0, however, will always be written as 0, and for small numbers the number of significant digits will never be less than 1:

```
jitter = 0.000000156789
echo The jitter is 'jitter:3%'.
```

This will write the following text to the Info window:

```
The jitter is 0.00002%.
```

Predefined string variables are `newline$`, `tab$`, and `shellDirectory$`. The last one specifies the directory that was the default directory when Praat started up; you can use it in scripts that run from the Unix or DOS command line.

Links to this page

- [Formulas 1.9. Formulas in scripts](#)
- [Scripting](#)
- [Scripting 5. Language elements](#)

© *ppgb*, April 14, 2004

Scripting 5.2. Formulas

In a Praat script, you can use numeric expressions as well as string expressions.

Numeric expressions

You can use a large variety of Formulas in your script:

```
length = 10
height = length/2
area = length * height
```

You can use numeric variables and formulas in arguments to commands:

```
Draw line... 0 length 0 length/2
```

Of course, all arguments except the last should either not contain spaces, or be enclosed in double quotes. So you would write either

```
Draw line... 0 height*2 0 height
```

or

```
Draw line... 0 "height * 2" 0 height
```

You can use numeric expressions in assignments (as above), or after **if**, **elsif**, **while**, **until**, and twice after **for**.

On how to get information from commands that normally write to the Info window, see Scripting 6.4. Query commands.

Links to this page

- [Scripting](#)
- [Scripting 5. Language elements](#)

© ppgb, December 18, 2002

Scripting 5.3. Jumps

You can use conditional jumps in your script:

if *expression*

elsif *expression*

if the expression evaluates to zero or *false*, the execution of the script jumps to the next **elsif** or after the next **else** or **endif** at the same depth..

The following script computes the preferred length of a bed for a person 'age' years of age:

```
if age <= 3
    length = 1.20
elsif age <= 8
    length = 1.60
else
    length = 2.00
endif
```

A variant spelling for **elsif** is **elif**.

Links to this page

- [Scripting](#)
- [Scripting 5. Language elements](#)

© ppgb, November 12, 1999

Scripting 5.4. Loops

"For" loops

for *variable* **from** *expression*₁ **to** *expression*₂

for *variable* **to** *expression*

the statements between the **for** line and the matching **endfor** will be executed while a variable takes on values between two expressions, with an increment of 1 on each turn of the loop. The default starting value of the loop variable is 1.

The following script plays nine sine waves, with frequencies of 200, 300, ..., 1000 Hz:

```
for i from 2 to 10
  frequency = i * 100
  Create Sound... tone 0 0.3 22050 0.9*sin(2*pi*frequency*x)
  Play
  Remove
endfor
```

The stop value of the **for** loop is evaluated on each turn. If the second expression is already less than the first expression to begin with, the statements between **for** and **endfor** are not executed even once.

"Repeat" loops

until *expression*

the statements between the matching preceding **repeat** and the **until** line will be executed again if the expression evaluates to zero or *false*.

The following script measures the number of trials it takes me to throw 12 with two dice:

```
throws = 0
repeat
  eyes = randomInteger (1, 6) + randomInteger (1, 6)
  throws = throws + 1
until eyes = 12
echo It took me 'throws' trials to throw 12 with two dice.
```

The statements in the **repeat/until** loop are executed at least once.

"While" loops

while *expression*

if the expression evaluates to zero or *false*, the execution of the script jumps after the matching **endwhile**.

endwhile

execution jumps back to the matching preceding **while** line, which is then evaluated again.

The following script forces the number x into the range $[0; 2\pi)$:

```
while x < 0
  x = x + 2 * pi
endwhile
while x >= 2 * pi
  x = x - 2 * pi
endwhile
```

If the expression evaluates to zero or *false* to begin with, the statements between **while** and **endwhile** are not executed even once.

Links to this page

- [Scripting](#)
 - [Scripting 5. Language elements](#)
-

© *ppgb*, April 14, 2004

Scripting 5.5. Procedures

In a Praat script, you can define and call *procedures* (subroutines).

call *procedureName* [*argument1* [*argument2* [...]]]

the execution of the script jumps to the line after the matching **procedure** line, which can be anywhere in the script.

procedure *procedureName* [*parameter1* [*parameter2* [...]]]

introduces a procedure definition (when the execution of the script happens to arrive here, it jumps after the matching **endproc**, i.e., the statements in the procedure are ignored).

endproc

the execution of the script jumps to the line after the **call** line that invoked this procedure.

The following script plays three notes:

```
call play_note 440
call play_note 400
procedure play_note frequency
  Create Sound... note 0 0.3 22050 0.9 * sin (2*pi*'frequency'*x)
  Play
  Remove
endproc
call play_note 500
```

The variable *frequency* is a normal variable, global to the script. The procedure uses the same name space as the rest of the script, so beware of possible conflicts.

For arguments that contain spaces, you use double quotes, except for the last argument, which is the rest of the line:

```
call Conjugate be "I am" "you are" she is
procedure Conjugate verb$ first$ second$ third$
  echo Conjugation of 'to 'verb$':
  printline 1sg 'first$'
  printline 2sg 'second$'
  printline 3sg 'third$'
endproc
```

Arguments (except for the last) that contain double quotes should also be put between double quotes, and the double quotes should be doubled:

```
procedure texts top$ bottom$
  Text top... yes 'top$'
  Text bottom... yes 'bottom$'
endproc
call texts ""hello"" at the top" "goodbye" at the bottom
```

Links to this page

- [Scripting](#)
 - [Scripting 5. Language elements](#)
-

© *ppgb*, December 1, 2002

Scripting 5.6. Arrays

Quote substitution allows you to simulate arrays of variables:

```
for i from 1 to 5
  square'i' = i * i
endfor
```

After this, the variables *square1*, *square2*, *square3*, *square4*, and *square5* contain the values 1, 4, 9, 16, and 25, respectively.

You can use any number of variables in a script, but you can also use Matrix or Sound objects for arrays.

You can substitute variables with the usual single quotes, as in 'square3'. If the index is also a variable, however, you may need a dummy variable:

```
echo Some squares:
for i from 1 to 5
  hop = square'i'
  printline The square of 'i' is 'hop'
endfor
```

The reason for this is that the following line would not work, because of the required double substitution:

```
print The square of 'i' is 'square'i''
```

Links to this page

- [Scripting](#)
- [Scripting 5. Language elements](#)

© ppgb, June 24, 2002

Scripting 5.7. Including other scripts

You can include other scripts within your script:

```
a = 5
include square.praat
echo 'a'
```

The Info window will show the result 25 if the file `square.praat` is as follows:

```
a = a * a
```

The inclusion is done before any other part of the script is considered, so you can use the **form** statement and all variables in it. Usually, however, you will put some procedure definitions in the include file, that is what it seems to be most useful for. Watch out, however, for using variable names in the include file: the example above shows that there is no such thing as a separate name space.

Since including other scripts is the first thing Praat will do when considering a script, you cannot use variable substitution. For instance, the following will not work:

```
scriptName$ = "myscript.praat"
#This will *not* work:
include 'scriptName$'
#That did *not* work!!!
```

You can use full or relative file names. For instance, the file `square.praat` is expected to be in the same directory as the script that says *include square.praat*. If you use the ScriptEditor, you will first have to save the script that you are editing before any relative file names become meaningful (this is the same as with other uses of relative file names in scripts).

You can *nest* include files, i.e., included scripts can include other scripts. However, relative file names are always evaluated relative to the directory of the outermost script.

Links to this page

- [Scripting](#)
- [Scripting 5. Language elements](#)
- [What's new?](#)

Scripting 5.8. Quitting

Usually, the execution of your script ends when the interpreter has executed the last line that is not within a procedure definition. However, you can also explicitly stop the script:

exit

stops the execution of the script in the normal way, i.e. without any messages to the user. Any settings window is removed from the screen.

exit *error-message*

stops the execution of the script while sending an error message to the user. Any settings window will stay on the screen.

For an example, see Scripting 6.8. Error message to the user.

Links to this page

- [Scripting](#)
- [Scripting 5. Language elements](#)

© *ppgb*, April 14, 2004

Scripting 6. Communication outside the script

- Scripting 6.1. Arguments to the script (form/endform, execute)
- Scripting 6.2. Calling system commands (system, system_nocheck)
- Scripting 6.3. Writing to the Info window (echo, print, printtab, printline)
- Scripting 6.4. Query commands (Get, Count)
- Scripting 6.5. Files (fileReadable, <, >, >>, filedelete, fileappend)
- Scripting 6.6. Controlling the user (pause)
- Scripting 6.7. Sending a message to another program (sendsocket)
- Scripting 6.8. Error message to the user (exit, assert)
- Scripting 6.9. Calling from the command line

Links to this page

- [Scripting](#)
-

© *ppgb*, July 26, 2003

Scripting 6.1. Arguments to the script

You can cause a Praat script to prompt for arguments. The file `playSine.praat` may contain the following:

```
form Play a sine wave
  positive Sine_frequency_(Hz) 377
  positive Gain_(0..1) 0.3 (= not too loud)
endform
Create Sound... sine'sine_frequency' 0 1 10000 'gain' * sin
(2*pi*'sine_frequency'*x)
Play
Remove
```

When running this script, the interpreter puts a settings window (*form*) on your screen, entitled "Play a sine wave", with two fields, titled "Sine frequency (Hz)" and "Gain", that have been provided with the standard values "377" and "0.3 (= not too loud)", which you can change before clicking **OK**.

As you see, the underscores have been replaced with spaces: that looks better in the form. Inside the script, the field names can be accessed as variables: these do contain the underscores, since they must not contain spaces, but the parentheses (Hz) have been chopped off. Note that the first letter of these variables is converted to lower case, so that you can assign to them in your script.

Inside the script, the value "0.3 (= not too loud)" will be known as "0.3", because this is a numeric field.

You can use the following field types in your forms:

real *variable initialValue*

for real numbers.

positive *variable initialValue*

for positive real numbers: the form issues an error message if the number that you enter is negative or zero; further on in the script, the number may take on any value.

integer *variable initialValue*

for whole numbers: the form reads the number as an integer; further on in the script, the number may take on any real value.

natural *variable initialValue*

for positive whole numbers: the form issues an error message if the number that you enter is negative or zero; further on in the script, the number may take on any real value.

word *variable initialValue*

for a string without spaces: the form only reads up to the first space ("oh yes" becomes "oh"); further on in the script, the string may contain spaces.

sentence *variable initialValue*

for any short string.

text *variable initialValue*

for any possibly long string (the variable name will not be shown in the form).

boolean *variable initialValue*

a check box will be shown; the value is 0 if off, 1 if on.

choice *variable initialValue*

a radio box will be shown; the value is 1 or higher. This is followed by a series of:

button *text*

a button in a radio box.

comment *text*

a line with any text.

Inside the script, strings are known as string variables, numbers as numeric variables:

```
form Sink it
  sentence Name_of_the_ship Titanic
  real Distance_to_the_iceberg_(m) 500.0
  natural Number_of_people 1800
  natural Number_of_boats 10
endform
```

In this script, the variables are known as *name_of_the_ship*\$, *distance_to_the_iceberg*, *number_of_people*, and *number_of_boats*.

The variable associated with a radio box will get a numeric as well as a string value:

```
form Fill attributes
  comment Choose any colour and texture for your paintings
  choice Colour: 5
    button Dark red
    button Sea green
    button Navy blue
    button Canary yellow
    button Black
    button White
  choice Texture: 1
    button Smooth
    button Rough
    button With holes
endform
echo You chose the colour 'colour$' and texture 'texture$'.
```

This shows two radio boxes. In the Colour box, the fifth button (Black) is the standard value here. If you click on "Navy blue" and then **OK**, the variable *colour* will have the value "3", and the variable *colour*\$ will have the value "Navy blue". Note that the trailing colon is chopped off, and that the button and comment texts may contain spaces. So you can test the value of the Colour box in either of the following ways:

```
if colour = 4
```

or

```
if colour$ = "Canary yellow"
```

The field types **optionmenu** and **option** are completely analogous to **choice** and **button**, but use up much less space on the screen:

form Fill attributes

```
  comment Choose any colour and texture for your paintings
```

```
  optionmenu Colour: 5
```

```
    option Dark red
```

```
    option Sea green
```

```
    option Navy blue
```

```
    option Canary yellow
```

```
    option Black
```

```
    option White
```

```
  optionmenu Texture: 1
```

```
    option Smooth
```

```
    option Rough
```

```
    option With holes
```

endform

```
echo You chose the colour 'colour$' and texture 'texture$'.
```

You can combine two short fields into one by using *left* and *right*:

form Get duration

```
  natural left_Year_range 1940
```

```
  natural right_Year_range 1945
```

endform

```
duration = right_Year_range - left_Year_range
```

```
echo The duration is 'duration' years.
```

The interpreter will only show the single text "Year range", followed by two small text fields.

Calling a script from another script

Scripts can be nested: the file *doremi.praat* may contain the following:

```
execute playSine.praat 550 0.9
```

```
execute playSine.praat 615 0.9
```

```
execute playSine.praat 687 0.9
```

With the **execute** command, Praat will not display a form window, but simply execute the script with the two arguments that you supply on the same line (e.g. 550 and 0.9).

Arguments (except for the last) that contain spaces must be put between double quotes, and values for **choice** must be passed as strings:

execute "fill attributes.praat" "Navy blue" With holes

You can pass values for **boolean** either as "yes" and "no" or 1 and 0.

Links to this page

- [Scripting](#)
 - [Scripting 6. Communication outside the script](#)
 - [Scripting 6.9. Calling from the command line](#)
-

© *ppgb*, April 14, 2004

Scripting 6.2. Calling system commands

You can call system commands from a Praat script on Unix, Windows, and MacOS X computers.

system *command*

executes a Unix or Windows shell command, interpreting non-zero return values as errors:

```
system cd /u/miep/sounds; sfplay hallo.aifc
```

system_nocheck *command*

executes a Unix or Windows shell command, ignoring return values:

```
system_nocheck rm dummy.aifc
```

In the last example, using **system** `rm dummy.aifc` would cause the script to stop if the file `dummy.aifc` does not exist.

environment\$ (*symbol-string*)

returns the value of an environment variable under Unix, e.g.

```
homeDirectory$ = environment$ ( "HOME" )
```

Links to this page

- [Scripting](#)
- [Scripting 6. Communication outside the script](#)

© ppgb, March 17, 2002

Scripting 6.3. Writing to the Info window

With the Info button and several commands in the **Query** submenu, you write to the Info window. If your program is run from batch (on Unix or Windows), the text goes to *stdout*.

The following commands allow you to write to the Info window from a script only:

echo *text*

clears the Info window and writes some text to it:

```
echo Starting simulation...
```

clearinfo

clears the Info window.

print *text*

appends some text to the Info window, without clearing it and without going to a new line.

printtab

appends a *tab* character to the Info window. This allows you to create table files that can be read by some spreadsheet programs.

println [*text*]

causes the following text in the Info window to begin at a new line. You can add text, just like with **print**.

The following script builds a table with statistics about a pitch contour:

```
clearinfo
println Minimum Maximum
Create Sound... sin 0 0.1 10000 sin(2*pi*377*x)
To Pitch... 0.01 75 600
minimum = Get minimum... 0 0 Hertz Parabolic
print 'minimum'
printtab
maximum = Get maximum... Hertz
print 'maximum'
println
```

You could combine the last four print statements into:

```
println 'minimum' 'tab$' 'maximum'
```

or:

```
print 'minimum' 'tab$' 'maximum' 'newline$'
```

Links to this page

- [Scripting](#)
 - [Scripting 6. Communication outside the script](#)
-

© *ppgb*, March 2, 2000

Scripting 6.4. Query commands

If you click the "Get mean..." command for a Pitch object, the Info window will contain a text like "150 Hz" as a result. In a script, you would rather have this result in a variable instead of in the Info window. The solution is simple:

```
mean = Get mean... 0 0 Hertz Parabolic
```

The numeric variable "mean" now contains the number 150. When assigning to a numeric variable, the interpreter converts the part of the text before the first space into a number.

You can also assign to string variables:

```
mean$ = Get mean... 0 0 Hertz Parabolic
```

The string variable "mean\$" now contains the entire string "150 Hz".

This works for every command that would otherwise write into the Info window.

Links to this page

- [Query](#)
- [Scripting](#)
- [Scripting 5.2. Formulas](#)
- [Scripting 6. Communication outside the script](#)

© ppgb, January 8, 1999

Scripting 6.5. Files

You can read from and write to text files from a Praat script.

Reading a file

You can check the availability of a file for reading with the function

```
fileReadable (fileName$)
```

which returns **true** if the file exists and can be read, and **false** otherwise.

To read the contents of an existing text file into a string variable, you use

```
text$ < fileName
```

where *text\$* is any string variable and *fileName* is an unquoted string. If the file does not exist, the script terminates with an error message.

Example: reading a settings file

Suppose that the file **height.inf** may contain an appropriate value for a numeric variable called *height*, which we need to use in our script. We would like to read it with

```
height$ < height.inf  
height = 'height$'
```

However, this script will fail if the file **height.inf** does not exist. To guard against this situation, we could check the existence of the file, and supply a default value in case the file does not exist:

```
fileName$ = "height.inf"  
if fileReadable (fileName$)  
    height$ < 'fileName$'  
    height = 'height$'  
else  
    height = 180  
endif
```

Writing a file

To write the contents of an existing string into a new text file, you use

```
text$ > fileName
```

where `text$` is any string variable and `fileName` is an unquoted string. If the file cannot be created, the script terminates with an error message.

To append the contents of an existing string at the end of an existing text file, you use

```
text$ >> fileName
```

If the file does not yet exist, it is created first.

You can delete an existing file with

```
filedelete fileName
```

If the file does not exist, **filedelete** does nothing.

The simplest way to append text to a file is by using **fileappend**:

```
fileappend out.txt Hello world!
```

Example: writing a table of squares

Suppose that we want to create a file with the following text:

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
...
The square of 100 is 10000
```

We can do this by collecting each line in a variable:

```
filedelete squares.txt
for i to 100
  square = i * i
  fileappend squares.txt The square of 'i' is 'square' 'newline$'
endfor
```

Note that we delete the file before appending to it, in order that we do not append to an already existing file.

If you put the name of the file into a variable, make sure to surround it with double quotes when using **fileappend**, since the file name may contain spaces and is not at the end of the line:

```
name$ = "Hard disk:Desktop Folder:squares.text"
filedelete 'name$'
for i to 100
  square = i * i
  fileappend "'name$'" The square of 'i' is 'square' 'newline$'
endfor
```

Finally, you can append the contents of the Info window to a file with

```
fappendinfo fileName
```

Directory listings

To get the names of the files of a certain type in a certain directory, use Create Strings as file list...

Links to this page

- [Scripting](#)
 - [Scripting 6. Communication outside the script](#)
-

© *ppgb*, August 21, 2001

Scripting 6.6. Controlling the user

You can temporarily halt a Praat script:

pause *text*

suspends execution of the script, and allows the user to interrupt it. A message window will appear with the *text* and the buttons Continue and Stop:

```
pause The next file will be beerbeet.TextGrid
```

Links to this page

- [Scripting](#)
 - [Scripting 6. Communication outside the script](#)
-

© *ppgb*, April 14, 2004

Scripting 6.7. Sending a message to another program

To send messages to running programs that use the Praat shell, use `sendpraat` (see Scripting 8. Controlling Praat from another program).

To send a message to another running program that listens to a socket, you can use the `sendsocket` directive. This works on Unix and Windows only.

Example

Suppose we are in the Praat-shell program **Praat**, which is a system for doing phonetics by computer. From this program, we can send a message to the *non*-Praat-shell program **MovieEdit**, which does know how to display a sound file:

```
Write to file... hallo.wav
sendsocket fonsg19.hum.uva.nl:6667 display hallo.wav
```

In this example, `fonsg19.hum.uva.nl` is the computer on which MovieEdit is running; you can specify any valid Internet address instead, as long as that computer allows you to send messages to it. If MovieEdit is running on the same computer as Praat, you can specify `localhost` instead of the full Internet address.

The number 6667 is the port number on which MovieEdit is listening. Other programs will use different port numbers.

Links to this page

- [Scripting](#)
- [Scripting 6. Communication outside the script](#)
- [What's new?](#)

© *ppgb*, December 18, 2002

Scripting 6.8. Error message to the user

If the user makes a mistake (e.g. types conflicting settings into your form window), you can use the **exit** directive (\SS5.7) to stop the execution of the script with an error message:

```
form My analysis
  real Starting_time_(s) 0.0
  real Finishing_time_(s) 1.0
endform
if finishing_time <= starting_time
  exit The finishing time should exceed 'starting_time' seconds.
endif
# Proceed with the analysis...
```

For things that should not normally go wrong, you can use the **assert** directive:

```
power = Get power
assert power > 0
```

This is the same as:

```
if (power > 0) = undefined
  exit Assertion failed in line xx (undefined): power > 0
elsif not (power > 0)
  exit Assertion failed in line xx (false): power > 0
endif
```

Links to this page

- [Scripting](#)
- [Scripting 6. Communication outside the script](#)

© *ppgb*, April 14, 2004

Scripting 6.9. Calling from the command line

On most computers, you can call a Praat script from the command line.

Command lines on Unix and MacOS X

On Unix or MacOS X, you call Praat scripts from the command line like this:

```
> /people/mietta/praat doit.praat 50 hallo
```

or

```
> /Applications/Praat.app/Contents/MacOS/Praat doit.praat 50 hallo
```

This opens PRAAT, runs the script **doit.praat** with arguments "50" and "hallo", and closes PRAAT.

You also have the possibility of running the program interactively from the command line:

```
> /people/mietta/praat -
```

You can then type in any of the fixed and dynamic commands, and commands that handle object selection, like **select**. This method also works in pipes:

```
> echo "Statistics..." | /people/mietta/praat -
```

Command lines on Windows

On Windows, you call Praat scripts from the command line like this:

```
e:\praatcon.exe e:\doit.praat 50 hallo
```

Note that you use **praatcon.exe** instead of **praat.exe**. The script will write to the console output, and its output can be used in pipes.

How to get arguments into the script

In the above example, the script **doit.praat** requires two arguments. In the script **doit.praat**, you use **form** and **endform** to receive these arguments. See Scripting 6.1. Arguments to the script. As with the **execute** command, Praat will not present a form window, but simply execute the script with the arguments given on the command line. The example given in Scripting 6.1. Arguments to the script will be called in the following way:

```
> /people/mietta/praat playSine.praat 550 0.9
```

or

```
e:\praatcon.exe playSine.praat 550 0.9
```

Links to this page

- [Scripting](#)
 - [Scripting 1. My first script](#)
 - [Scripting 6. Communication outside the script](#)
 - [Scripting 8.1. The sendpraat subroutine](#)
-

© *ppgb*, April 14, 2004

Scripting 7. Scripting the editors

With a Praat script, you can automatize your work in the editors.

Warning: if the purpose of your script is to get information about analyses (pitch, formants, intensity, spectrogram) from the Sound, we do *not* advise to script the Sound editor window. It is much simpler, faster, and more reproducible to create the analyses with the commands of the dynamic menu, then use the Query commands of the dynamic menu to extract information from the analyses. This also applies if you want to use a TextGrid to determine the times at which you want to query the analyses. See Scripting examples.

Scripting 7.1. Scripting an editor from a shell script (editor/endeditor)

Scripting 7.2. Scripting an editor from within

Links to this page

- [Scripting](#)

© ppgb, February 22, 2004

Scripting 7.1. Scripting an editor from a shell script

From a Praat shell script, you can switch to an editor and back again:

```
sound$ = "hallo"  
start = 0.3  
finish = 0.7  
Read from file... 'sound$.aifc  
Edit  
editor Sound 'sound$'  
    Zoom... start finish  
endeditor  
Play
```

This script reads a sound file from disk, pops up an editor for the resulting object, makes this editor zoom in on the part between 0.3 and 0.7 seconds, and returns to the Praat shell to play the entire sound.

Links to this page

- [FAQ: Scripts](#)
- [Scripting](#)
- [Scripting 7. Scripting the editors](#)

© *ppgb*, June 6, 2001

Scripting 7.2. Scripting an editor from within

This section will show how you can permanently extend the functionality of an editor.

As an example, consider the following problem: you want to see a graphic representation of the spectrum of the sound around the cursor position in the SoundEditor. To achieve this, follow these steps:

1. Create a Sound.
2. View it in a SoundEditor by clicking Edit.
3. Choose **New editor script** from the File menu in the SoundEditor. The resulting ScriptEditor will have a name like "untitled script [Sound hallo]".
4. Type the following lines into the ScriptEditor:

```
cursor = Get cursor
Select... cursor-0.02 cursor+0.02
Extract windowed selection... slice Kaiser2 2 no
endeditor
To Spectrum (fft)
Edit
```

If you choose Run from the Run menu in the ScriptEditor, a region of 40 milliseconds around the current cursor position in the SoundEditor will become selected. This piece will be copied to the list of objects, after applying a double Kaiser window (total length 80 ms). Thus, a Sound named "slice" will appear in the list. Subsequently, a Spectrum object also called "slice" will appear in the list, and a SpectrumEditor titled "Spectrum slice" will finally appear on your screen.

5. Save the script to disk, e.g. as /us/miep/spectrum.praat. The title of the ScriptEditor will change accordingly.
6. Since you will want this script to be available in all future SoundEditors, you choose **Add to menu...** from the File menu. For the *Window*, you specify "SoundEditor" (this is preset). For the *Menu*, you may want to choose "Spec." instead of the preset value ("File"). For the name of the *Command*, you type something like "Show spectrum at cursor" (instead of "Do it..."). Then you click OK.

The command will be visible in every SoundEditor that you create from now on. To see this, close the one visible SoundEditor, select the original Sound, choose Edit again, and inspect the "Spec." menu. You can now view the spectrum around the cursor just by choosing this menu command.

After you leave Praat and start it again, the command will continue to appear in the SoundEditor. If you don't like the command any longer, you can remove it with the ButtonEditor, which you can start by choosing **Buttons** from the Preferences submenu of the Control menu in the Objects window.

Improving your script

The above spectrum-viewing example has a number of disadvantages. It clutters the object list with a number of indiscriminable Sounds and Spectra called "slice", and the spectrum is shown up to the Nyquist frequency while we may just be interested in the lower 5000 Hz. Furthermore, the original selection in the SoundEditor is lost.

To improve the script, we open it again with **Open editor script...** from the File menu in the SoundEditor. After every change, we can run it with Run from the Run menu again; alternatively, we could save it (with Save from the File menu) and choose our new "Show spectrum at cursor" button (this button will always run the version on disk, never the one viewed in a ScriptEditor).

To zoom in on the first 5000 Hz, we add the following code at the end of our script:

```
editor Spectrum slice  
  Zoom... 0 5000
```

To get rid of the "Sound slice", we can add:

```
endeditor  
select Sound slice  
Remove
```

Note that **endeditor** is needed to change from the environment of a SpectrumEditor to the environment of the object & picture windows.

If you now choose the "Show spectrum at cursor" button for several cursor positions, you will notice that all those editors have the same name. To remedy the ambiguity of the line **editor** Spectrum slice, we give each slice a better name. For example, if the cursor was at 635 milliseconds, the slice could be named "635ms". We can achieve this by changing the extraction in the following way:

```
milliseconds = round (cursor*1000)  
Extract windowed selection... 'milliseconds'ms Kaiser2 2 no
```

The names of the Sound and Spectrum objects will now have more chance of being unique. Two lines will have to be edited trivially.

Finally, we will reset the selection to the original. At the top of the script, we add two lines to remember the positions of the selection markers:

```
begin = Get begin of selection  
end = Get end of selection
```

At the bottom, we reset the selection:

```
editor  
  Select... begin end
```

Note that the **editor** directive if not followed by the name of an editor, returns the script to the original environment.

The complete script is:

```
begin = Get begin of selection
end = Get end of selection
cursor = Get cursor
Select... cursor-0.02 cursor+0.02
# Create a name. E.g. "670ms" means at 670 milliseconds.
milliseconds = round (cursor*1000)
Extract windowed selection... 'milliseconds'ms Kaiser2 2 no
endeditor
To Spectrum (fft)
Edit
editor Spectrum 'milliseconds'ms
Zoom... 0 5000
endeditor
select Sound 'milliseconds'ms
Remove
editor
Select... begin end
```

This script is useful as it stands. It is good enough for safe use. For instance, if the created Sound object has the same name as an already existing Sound object, it will be the newly created Sound object that will be removed in the Remove line, because **select** always selects the most recently created object in case of ambiguity.

Links to this page

- Scripting
- Scripting 7. Scripting the editors

© ppgb, December 4, 2002

Scripting 8. Controlling Praat from another program

Scripting 8.1. The sendpraat subroutine
Scripting 8.2. The sendpraat program
Scripting 8.3. The sendpraat directive

Links to this page

- [Scripting](#)
 - [Scripting 6.7. Sending a message to another program](#)
 - [sendpraat](#)
-

© *ppgb*, December 18, 2002

Scripting 8.1. The sendpraat subroutine

A subroutine for sending messages to a *running* PRAAT. Also a Unix, MacOS, or DOS console program with the same purpose.

Syntax

```
sendpraat (void *display, const char *program, long timeOut, char *text);
```

Arguments

display

the display pointer if the subroutine is called from a running X program; if NULL, sendpraat will open the display by itself. On Windows and Macintosh, this argument is ignored.

program

the name of a running program that uses the Praat shell, e.g. "Praat" or "ALS". The first letter may be specified as lower or upper case; it will be converted to lower case for Unix and to upper case for Macintosh and Windows.

message

a sequence of Praat shell lines (commands and directives).

timeOut (Unix and Macintosh only)

the number of seconds that sendpraat will wait for an answer before writing an error message. A *timeOut* of 0 means that the message will be sent asynchronously, i.e., that sendpraat will return immediately without issuing any error message.

text

the script text to be sent. Sendpraat may alter this text!

Example 1: killing a program

```
char message [100], *errorMessage;
strcpy (message, "Quit");
errorMessage = sendpraat (NULL, "praat", 0, message);
if (errorMessage != NULL) fprintf (stderr, "%s", errorMessage);
```

This causes the program **Praat** to quit (gracefully), because **Quit** is a fixed command in the Control menu of that program. On Unix and Macintosh, sendpraat returns immediately; on Windows, you the *timeOut* argument is ignored. The return value *errorMessage* is a statically allocated string internal to sendpraat, and is overwritten by the next call to sendpraat.

Example 2: playing a sound file in reverse

Suppose you have a sound file whose name is in the variable `fileName`, and you want the program **Praat**, which can play sounds, to play this sound backwards.

```
char message [1000], *errorMessage;
sprintf (message, "Read from file... %s\nPlay reverse\nRemove",
fileName);
errorMessage = sendpraat (NULL, "praat", 0, message);
```

This will work because **Play reverse** is an action command that becomes available in the dynamic menu when a Sound is selected. On Unix, `sendpraat` will allow **Praat** at most 1000 seconds to perform this.

Example 3: executing a large script file

Sometimes, it may be unpractical to send a large script directly to `sendpraat`. Fortunately, the receiving program knows the `execute` directive:

```
char message [100], *errorMessage;
strcpy (message, "doAll.praat 20");
errorMessage = sendpraat (NULL, "praat", 0, message);
```

This causes the program **Praat** to execute the script `doAll.praat` with an argument of "20".

How to download

You can download the source code of the `sendpraat` subroutine and program via www.praat.org or from <http://www.fon.hum.uva.nl/praat/sendpraat.html>.

See also

To start a program from the command line instead and sending it a message, you would not use `sendpraat`, but instead run the program with a script file as an argument. See Scripting 6.9. Calling from the command line.

Links to this page

- Scripting
- Scripting 8. Controlling Praat from another program
- Scripting 8.2. The `sendpraat` program
- Scripting 8.3. The `sendpraat` directive

© *ppgb*, May 28, 2003

Scripting 8.2. The sendpraat program

A Unix or DOS console program for sending messages to a *running* PRAAT program.

Syntax

```
sendpraat [timeOut] program message...
```

For the meaning of the arguments, see the sendpraat subroutine.

Example 1: killing a program

```
sendpraat 0 praat Quit
```

Causes the program **Praat** to quit (gracefully), because **Quit** is a fixed command in the Control menu. On Unix, **sendpraat** returns immediately; on Windows, you leave out the *timeOut* argument.

Example 2: playing a sound file in reverse

```
sendpraat 1000 praat "Read from file... hello.wav" "Play reverse"  
"Remove"
```

This works because **Play reverse** is an action command that becomes available in the dynamic menu of the **Praat** program when a Sound is selected. On Unix, sendpraat will allow **Praat** at most 1000 seconds to perform this.

Each line is a separate argument. Lines that contain spaces should be put inside double quotes.

Example 3: drawing

```
sendpraat als "for i from 1 to 5" "Draw circle... 0.5 0.5 i" "endfor"
```

This causes the program **Als** to draw five concentric circles into the Picture window.

Example 4: executing a large script

```
sendpraat praat "execute doAll.praat 20"
```

This causes the program **Praat** to execute the script **doAll.praat** with an argument of "20".

Links to this page

- [Scripting](#)
- [Scripting 8. Controlling Praat from another program](#)
- [Scripting 8.3. The sendpraat directive](#)

© *ppgb*, May 28, 2003

Scripting 8.3. The sendpraat directive

Besides being a subroutine (Scripting 8.1. The sendpraat subroutine) and a program (Scripting 8.2. The sendpraat program), sendpraat can also be called from within a Praat script.

Example 1: killing a program

Suppose we are in the Praat-shell program **Als**, which is a browser for dictionaries, and we want to kill the Praat-shell program **Praat**, which is a program for phonetics research:

```
sendpraat Praat Quit
```

Example 2: playing a sound

Suppose we are in the Praat-shell program **Als**, which is a browser for dictionaries, and has no idea of what a *sound* is. From this program, we can play a sound file by sending a message to the Praat-shell program **Praat**, which does know about sounds:

```
fileName$ = "hallo.wav"
sendpraat Praat
... 'newline$' Read from file... 'fileName$'
... 'newline$' Play
... 'newline$' Remove
```

The first `newline$` is superfluous, but this format seems to read nicely.

Links to this page

- [Scripting](#)
- [Scripting 8. Controlling Praat from another program](#)

© *ppgb*, December 18, 2002

Scripting examples

Here is a number of examples of how to use scripting in the Praat program. Refer to the scripting tutorial when necessary.

- Script for listing time\F0 pairs
- Script for listing time\F0\--intensity
- Script for listing F0 statistics
- Script for creating a frequency sweep
- Script for onset detection
- Script for TextGrid boundary drawing
- Script for analysing pitch with a TextGrid

Links to this page

- [Scripting 7. Scripting the editors](#)

© *ppgb*, February 22, 2004

Feedforward neural networks 1.1. The learning phase

During the learning phase the weights in the FFNet will be modified. All weights are modified in such a way that when a pattern is presented, the output unit with the correct category, hopefully, will have the largest output value.

How does learning take place?

The FFNet uses a *supervised* learning algorithm: besides the input pattern, the neural net also needs to know to what category the pattern belongs. Learning proceeds as follows: a pattern is presented at the inputs. The pattern will be transformed in its passage through the layers of the network until it reaches the output layer. The units in the output layer all belong to a different category. The outputs of the network as they are now are compared with the outputs as they ideally would have been if this pattern were correctly classified: in the latter case the unit with the correct category would have had the largest output value and the output values of the other output units would have been very small. On the basis of this comparison all the connection weights are modified a little bit to guarantee that, the next time this same pattern is presented at the inputs, the value of the output unit that corresponds with the correct category is a little bit higher than it is now and that, at the same time, the output values of all the other incorrect outputs are a little bit lower than they are now. (The differences between the actual outputs and the idealized outputs are propagated back from the top layer to lower layers to be used at these layers to modify connection weights. This is why the term *backpropagation network* is also often used to describe this type of neural network.

If you perform the procedure above once for every pattern and category pair in your data set you have performed 1 epoch of learning.

The hope is that eventually, probably after many epochs, the neural net will remember these pattern-category pairs. You even hope that the neural net when the learning phase has terminated, will be able to *generalize* and has learned to classify correctly any unknown pattern presented to it.

Because real-life data many times contains noise as well as partly contradictory information these hopes can only be partly fulfilled.

For learning you need to select 3 different objects together: a FFNet (the *classifier*), a Pattern (the *inputs*) and a Categories (the *correct outputs*).

How long will the learning phase take?

In general this question is hard to answer. It depends on the size of the neural network, the number of patterns to be learned, the number of epochs, the tolerance of the minimizer and the speed of your computer, how much computing time the learning phase may take.

If computing time becomes excessive in your interactive environment then consider using the powerful scripting facilities in PRAAT to process your learning job as a batch job.

Links to this page

- [Feedforward neural networks](#)
 - [Feedforward neural networks 1. What is a feedforward neural network?](#)
 - [Feedforward neural networks 1.2. The classification phase](#)
-

© *djmw*, April 28, 2004

Formulas 1.9. Formulas in scripts

In scripts, you can assign numeric expressions to numeric variables, and string expressions to string variables. You can also use numeric and string variables in expressions.

Example: report a square

Choose New Praat script from the Control menu. A script editor window will become visible. Type the following lines into that window:

```
x = 99
x2 = x * x
echo The square of 'x' is 'x2'.
```

This is an example of a simple Praat script; it assigns the results of the numeric formulas 99 and $x * x$ to the numeric variables x and $x2$. Note that the formula $x * x$ itself refers to the variable x . To run (execute) this script, type Command-R or choose **Run** from the **Run** menu. Praat will then write the following text into the Info window:

```
The square of 99 is 9801.
```

For more information on scripts, see the Scripting tutorial.

Example: rename the city of Washington

Type the following text into the script editor window:

```
current$ = "Bush"
previous$ = "Clinton"
famous$ = "Lincoln"
newCapital$ = current$ + mid$ (famous$, 2, 3) + right$ (previous$, 3)
echo The new capital will be 'newCapital$'.
```

This script assigns the results of four string expressions to the four string variables *current\$*, *previous\$*, *famous\$*, and *newCapital\$*. The dollar sign is the notation for a string variable or for a function whose result is a string (like **left\$**). Note that the formula in the fourth line refers to three existing variables.

To see what the new name of the capital will be, choose **Run**.

Example: numeric expressions in settings in scripts

As in real settings windows, you can use numeric expressions in all numeric fields. The example of the previous page becomes:

```
Create Sound... sine 0 10000/22050 22050 0.9 * sin (2*pi*377*x)
```

If the numeric field is not the last field of the settings window, you will want to write the formula without any spaces, e.g. as 10000/22050, since spaces are used to separate the fields.

Example: string expressions in settings in scripts

As in real settings windows, you cannot use string expressions in text fields directly, but you can still use the trick of variable substitution with single quotes (see Scripting 5.1. Variables):

```
soundName$ = "hello"  
fileName$ = soundName$ + ".wav"  
Read from file... 'fileName$'
```

Example: numeric expressions in creation in scripts

Suppose you want to generate a sine wave whose frequency is held in a variable. This is the way:

```
frequency = 377  
Create Sound... sine 0 1 22050 0.9 * sin (2*pi*frequency*x)
```

In this example, Praat will protest if x is a variable as well, because that would be ambiguous with the x that refers to the time in the sound (see Formulas 1.8. Formulas for modification).

Links to this page

- [Formulas](#)
- [Formulas 1. My first formulas](#)
- [Formulas 1.6. Formulas in settings windows](#)

© *ppgb*, March 9, 2003

History mechanism

The easiest way to do scripting. The *history* is the sequence of all menu commands (in the Objects or Picture window or in the editors), action commands (in the dynamic menu), or mouse clicks on objects (in the list of objects), that you performed during your Praat session, together with the settings that you specified in the settings windows that popped up as a result of those commands.

Viewing the history

To view your history, you first open a ScriptEditor with New Praat script or Open Praat script.... You then choose Paste history from the Edit menu.

Recording a macro

To record a sequence of mouse clicks for later re-use, perform the following steps:

1. Choose Clear history from the Edit menu. This makes the history mechanism forget all previous clicks.
2. Perform the actions that you want to record.
3. Choose Paste history from the Edit menu. Because you cleared the history before you started, the resulting script contains only the actions that you performed in step 2. You can now already re-run the actions that you performed in step 2.
4. You can save the recorded actions to a script file by choosing **Save** from the File menu.
5. You can put this script file under a button in the dynamic menu by choosing Add to dynamic menu... from the File menu, or under a button in a fixed menu by choosing Add to fixed menu.... This button will be preserved across Praat sessions.

This macro mechanism is much more flexible than the usual opaque macro mechanism used by most programs, because you can edit the script and make some of the arguments variable by putting them in the **form** clause at the top of the script. In this way, the script will prompt the user for these arguments, just as with all the menu and action commands that end in the three dots (...). See the Scripting tutorial for all the things that you can do in scripts.

Links to this page

- [Scripting 1. My first script](#)
- [Scripting 2. Arguments to commands](#)

© ppgb, April 14, 2004

Praat script

An executable text that consists of menu commands and action commands.

See the Scripting tutorial.

Links to this page

- [Add to dynamic menu...](#)
- [Add to fixed menu...](#)
- [Buttons file](#)
- [Formulas](#)
- [Formulas 1.9. Formulas in scripts](#)
- [Info window](#)
- [Initialization script](#)
- [Open Praat script...](#)
- [Run script...](#)
- [ScriptEditor](#)
- [What's new?](#)

© *ppgb*, August 24, 1998

Programming with Praat

You can extend the functionality of the PRAAT program by adding modules written in C to it. All of PRAAT's source code is available under the General Public Licence (except for the C versions of the PostScript IPA fonts in the **ipa** directory, which you can copy but not modify).

1. Warning

Before trying the task of learning how to write PRAAT extensions, you should be well aware of the possibilities of scripting. Many built-in commands in PRAAT have started their lives as PRAAT scripts, and scripts are easier to write than C extensions.

2. Getting the existing source code

You obtain the PRAAT source code via www.praat.org, in a file with a name like **praat4105_sources.tar.gz** (depending on the PRAAT version), and unpack this with **gunzip** and **tar xvf** (on Unix), or **StuffIt® Expander™** (on Macintosh), or **Aladdin® Expander™** (on Windows). The result will be a set of directories called **ipa**, **GSL**, **sys**, **dwsys**, **fon**, **dwtools**, **LPC**, **FFNet**, **artsynth**, **main**, and **makefiles**, plus a makefile and a Codewarrior project for Macintosh and Windows.

3. Building PRAAT on Unix

To compile and link PRAAT on Unix, you go to the directory that contains the source directories and the makefile, and copy a **makefile.defs** file from the **makefiles** directory. On Linux, for instance, you type

```
> cp makefiles/makefile.defs.linux.dynamic ./makefile.defs
```

The file **makefile.defs** may require some editing after this, since the libraries in your Linux distribution may be different from mine, or your Motif library (**Xm.a** or **Xm.so**) is in a different location. On Silicon Graphics Irix, you will use **makefile.defs.sgi**; on Sparc Solaris, you will probably use **makefile.defs.solaris.cde**; on HPUNIX, you will use **makefile.defs.hpux**; on Intel Solaris, **makefile.defs.solaris.cde.up.ac**.

PRAAT has to be linked with a real Motif version (*not* Lesstif), i.e., there must be a library **Xm.so** or **Xm.a** and an **Xm** include directory. There exists a free Motif library for Linux (**openmotif-devel-2.1.30-1_IC.S.i386.rpm** from ICT); on SGI, Solaris, and HPUNIX, Motif libraries are included with the system, and the include directory comes with the development package.

4. Building PRAAT on Macintosh

Unpack **main/praat_mac.rsrc.sit** to **main/praat_mac.rsrc**. Then open **praat.mcp** in CodeWarrior (version 8.2 or higher), choose the target **praat_macx** (for MacOS X), **praat_mac9** (for MacOS 8.5 to 9.2), or **praat_mac7** (for older systems from 7.1 on), and choose Make or Run.

5. Building PRAAT on Windows

Open **praak.mcp** in CodeWarrior (version 8.0 or higher), choose the target **praak_win**, and choose Make or Run.

PRAAT may compile under Visual C++ as well.

6. Extending PRAAT

You can edit **main/main_Praat.c**. This example shows you how to create a very simple program with all the functionality of the PRAAT program, and a single bit more:

```
#include "praak.h"
DIRECT (HelloFromJane)
    Melder_information ("Hello, I am Jane.");
END
void main (int argc, char **argv) {
    praat_init ("Praat_Jane", argc, argv);
    INCLUDE_LIBRARY (praak_uvafon_init)
    praat_addMenuCommand ("Objects", "Control", "Hello from Jane...",
NULL, 0, DO_HelloFromJane);
    praat_run ();
    return 0;
}
```

7. Learning how to program

To see how objects are defined, take a look at **sys/Thing.h**, **sys/Data.h**, **sys/oo.h**, the **XXX_def.h** files in the **fon** directory, and the corresponding **XXX.c** files in the **fon** directory. To see how commands show up on the buttons in the fixed and dynamic menus, take a look at the large interface description file **fon/praat_Fon.c**.

8. Using the PRAAT shell only

For building the PRAAT shell (the Objects and Picture windows) only, you need only the code in the four directories **ipa**, **GSL**, **sys**, and **dwsys**. You delete the inclusion of **praak_uvafon_init** from **main**. You will be able to build a PRAAT shell, i.e. an Objects and a Picture window, which has no knowledge of the world, i.e., which does not know any objects that can be included in the list of objects. You could use this PRAAT shell for modelling your own world and defining your own classes of objects. For advanced programmers only.

© *ppgb*, July 19, 2003

Modify

The title of a submenu of the dynamic menu for many object types. This submenu usually collects all the commands that can change the selected object.

Links to this page

- [Formulas](#)
 - [Formulas 1.8. Formulas for modification](#)
 - [Formulas 8. Data in objects](#)
-

© *ppgb*, December 4, 2002

Cochleagram

One of the types of objects in PRAAT. It represents the excitation pattern of the basilar membrane in the inner ear (see Excitation) as a function of time.

Links to this page

- [Cochleagram: Formula...](#)
- [Formulas 1.8. Formulas for modification](#)
- [Formulas 7. Attributes of objects](#)
- [spectro-temporal representation](#)

© *ppgb*, March 16, 2003

Excitation

One of the types of objects in PRAAT. It represents the excitation pattern of the basilar membrane in the inner ear.

Inside an Excitation object

With Inspect, you will see the following attributes.

$xmin = 0$

minimum place or frequency (Bark).

$xmax = 25.6$ Bark

maximum place or frequency (Bark).

nx

number of places or frequencies.

$dx = 25.6 / nx$

Place or frequency step (Bark).

$x1 = dx / 2$

centre of first place or frequency band (Bark).

$ymin = ymax = dy = y_1 = 1; ny = 1$

dummies.

$z [1]$

intensity (sensation level) in phon.

Links to this page

- [Cochleagram](#)
- [Excitation: Formula...](#)
- [Excitation: Get loudness](#)
- [Excitations](#)
- [Excitations: To Pattern...](#)
- [Formulas 1.8. Formulas for modification](#)
- [Formulas 7. Attributes of objects](#)

© ppgb, March 16, 2003

Harmonicity

One of the types of objects in PRAAT.

A Harmonicity object represents the degree of acoustic periodicity, also called Harmonics-to-Noise Ratio (HNR). Harmonicity is expressed in dB: if 99% of the energy of the signal is in the periodic part, and 1% is noise, the HNR is $10 \cdot \log_{10}(99/1) = 20$ dB. A HNR of 0 dB means that there is equal energy in the harmonics and in the noise.

Harmonicity can be used as a measure for:

- The signal-to-noise ratio of anything that generates a periodic signal.
- Voice quality. For instance, a healthy speaker can produce a sustained [a] or [i] with a harmonicity of around 20 dB, and an [u] at around 40 dB; the difference comes from the high frequencies in [a] and [i], versus low frequencies in [u], resulting in a much higher sensitivity of HNR to jitter in [a] and [i] than in [u]. Hoarse speakers will have an [a] with a harmonicity much lower than 20 dB. We know of a pathological case where a speaker had an HNR of 40 dB for [i], because his voice let down above 2000 Hz.

Harmonicity commands

Creation:

- Sound: To Harmonicity (cc)...: cross-correlation method (preferred).
- Sound: To Harmonicity (ac)...: autocorrelation method.

Links to this page

- Formulas 1.8. Formulas for modification
- Formulas 7. Attributes of objects
- Get frame number from time...
- Get number of frames
- Get time from frame number...
- Get time step
- Harmonicity: Formula...
- Harmonicity: Get maximum...
- Harmonicity: Get mean...
- Harmonicity: Get minimum...
- Harmonicity: Get standard deviation...
- Harmonicity: Get time of maximum...
- Harmonicity: Get time of minimum...
- Harmonicity: Get value at time...
- Harmonicity: Get value in frame...
- time domain

- Voice 4. Additive noise
 - What's new?
-

© *ppgb*, June 10, 2003

Formulas

You can use numeric expressions and string (text) expressions in many places in Praat:

- in the calculator in Praat's Goodies submenu;
- in the numeric fields of most settings windows;
- in a Praat script.

For some types of objects (mainly Sound and Matrix), you can also apply formulas to all their contents at the same time:

- when you create a Sound or a Matrix from the New menu;
- when you choose Formula... from the Modify submenu for a selected object.

You can read this tutorial sequentially with the help of the "< 1" and "1 >" buttons.

1. My first formulas (where to use)
 - 1.1. Formulas in the calculator
 - 1.2. Numeric expressions
 - 1.3. String expressions
 - 1.4. Representation of numbers
 - 1.5. Representation of strings
 - 1.6. Formulas in settings windows
 - 1.7. Formulas for creation
 - 1.8. Formulas for modification
 - 1.9. Formulas in scripts
2. Operators (+, -, *, /, ^)
3. Constants (pi, e, undefined)
4. Mathematical functions
5. String functions
6. Control structures (if then else fi, semicolon)
7. Attributes of objects
8. Data in objects

Links to this page

- [binomialQ](#)
- [Cochleagram: Formula...](#)
- [Create Configuration...](#)
- [Create Sound...](#)
- [Excitation: Formula...](#)
- [Harmonicity: Formula...](#)
- [Matrix: Formula...](#)
- [Scripting 5.2. Formulas](#)
- [Sound: Formula...](#)

- Spectrogram: Formula...
 - Spectrum: Formula...
 - VocalTract: Formula...
 - What's new?
-

© *ppgb*, April 14, 2004

Formulas 1. My first formulas

- 1.1. Formulas in the calculator
- 1.2. Numeric expressions
- 1.3. String expressions
- 1.4. Representation of numbers
- 1.5. Representation of strings
- 1.6. Formulas in settings windows
- 1.7. Formulas for creation
- 1.8. Formulas for modification
- 1.9. Formulas in scripts

Links to this page

- [Formulas](#)
-

© *ppgb*, April 14, 2004

Calculator

A window that allows you to calculate all kinds of simple or complicated mathematical and string expressions. To show the calculator, type Command-U or choose the Calculator... command. The result will be written to the Info window.

See the Formulas tutorial for all the things that you can calculate with this command.

Links to this page

- [Formulas 1.1. Formulas in the calculator](#)
- [Formulas 1.2. Numeric expressions](#)
- [Formulas 1.3. String expressions](#)
- [Formulas 5. String functions](#)
- [Formulas 7. Attributes of objects](#)
- [Formulas 8. Data in objects](#)
- [undefined](#)

© *ppgb*, December 1, 2002

Query

Query commands give you information about objects.

Most query commands start with the word *Get* or sometimes the word *Count*. You will find these commands in two places: under the *Query* submenu that usually appears if you select an object in the list, and in the Query menus of the editors.

Behaviour

If you click a query command, the answer will be written to the Info window.

Scripting

In a script, you can still use query commands to write the information to the Info window but you can also use any query command to put the information into a variable. (see Scripting 6.4. Query commands). In such a case, the value will not be written into the Info window.

Query commands in the Praat program

The Praat program contains the following query commands:

- Confusion: Get fraction correct
- DurationTier: Get target duration...
- Excitation: Get loudness
- FilterBank: Get frequency in Hertz...
- Formant: Get bandwidth at time...
- Formant: Get maximum...
- Formant: Get mean...
- Formant: Get minimum...
- Formant: Get number of formants
- Formant: Get quantile...
- Formant: Get standard deviation
- Formant: Get time of maximum...
- Formant: Get time of minimum...
- Formant: Get value at time...
- Formulas 7. Attributes of objects
- Get area...
- Get high index from time...
- Get low index from time...
- Get nearest index from time...
- Harmonicity: Get maximum...
- Harmonicity: Get mean...

- Harmonicity: Get minimum...
- Harmonicity: Get standard deviation...
- Harmonicity: Get time of maximum...
- Harmonicity: Get time of minimum...
- Harmonicity: Get value at time...
- Harmonicity: Get value in frame...
- Intensity: Get maximum...
- Intensity: Get mean...
- Intensity: Get minimum...
- Intensity: Get standard deviation...
- Intensity: Get time of maximum...
- Intensity: Get time of minimum...
- Intensity: Get value at time...
- Intensity: Get value in frame...
- Log files
- Ltas: Get band from frequency...
- Ltas: Get band width
- Ltas: Get frequency from band...
- Ltas: Get frequency of maximum...
- Ltas: Get frequency of minimum...
- Ltas: Get frequency range
- Ltas: Get highest frequency
- Ltas: Get lowest frequency
- Ltas: Get maximum...
- Ltas: Get mean...
- Ltas: Get minimum...
- Ltas: Get number of bands
- Ltas: Get standard deviation...
- Ltas: Get value at frequency...
- Ltas: Get value in band...
- PitchTier: Get mean (curve)...
- PitchTier: Get mean (points)...
- PitchTier: Get standard deviation (curve)...
- PitchTier: Get standard deviation (points)...
- PointProcess: Get high index...
- PointProcess: Get interval...
- PointProcess: Get jitter (ddp)...
- PointProcess: Get jitter (local)...
- PointProcess: Get jitter (local, absolute)...
- PointProcess: Get jitter (ppq5)...
- PointProcess: Get jitter (rap)...
- PointProcess: Get low index...
- PointProcess: Get nearest index...

- Sound: Get absolute extremum...
 - Sound: Get energy in air
 - Sound: Get energy...
 - Sound: Get intensity (dB)
 - Sound: Get maximum...
 - Sound: Get mean...
 - Sound: Get minimum...
 - Sound: Get nearest zero crossing...
 - Sound: Get power in air
 - Sound: Get power...
 - Sound: Get root-mean-square...
 - Sound: Get standard deviation...
 - Sound: Get time of maximum...
 - Sound: Get time of minimum...
 - Sound: Get value at sample number...
 - Sound: Get value at time...
-

© *ppgb*, December 18, 2002

Matrix

One of the types of objects in PRAAT. A Matrix object represents a function $z(x, y)$ on the domain $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$. The domain has been sampled in the x and y directions with constant sampling intervals (dx and dy) along each direction. The samples are thus $z[i_y][i_x]$, $i_x = 1 \dots n_x$, $i_y = 1 \dots n_y$. The samples represent the function values $z(x_1 + (i_x - 1) dx, y_1 + (i_y - 1) dy)$.

Matrix commands

Creation:

- Create Matrix...
- Create simple Matrix...
- Read from file...
- Read Matrix from raw text file...
- **Read Matrix from LVS AP file...**

Drawing:

- **Matrix: Draw rows...**
- **Matrix: Draw contours...**
- **Matrix: Paint contours...**
- **Matrix: Paint cells...**
- **Matrix: Scatter plot...**
- Matrix: Draw as squares...
- **Matrix: Draw value distribution...**
- **Matrix: Paint surface...**

Modification:

- Matrix: Formula...
- **Matrix: Scale...**

Inside a Matrix object

With Inspect, you will see the following attributes.

$xmin, xmax \geq xmin$
 x domain.

$nx \geq 1$
 number of columns.

$dx > 0.0$
 distance between columns.

$x1$
 x value associated with first column.
 $ymin, ymax \geq ymin$
 y domain.
 $ny \geq 1$
 number of rows.
 $dy > 0.0$
 distance between rows.
 $y1$
 y value associated with first row.
 $z [1..ny] [1..nx]$
 The sample values.

After creation of the **Matrix**, $xmin$, $xmax$, $ymin$, $ymax$, nx , ny , dx , dy , $x1$, and $y1$ do not usually change. The contents of z do.

Normally, you will want $xmin \leq x1$ and $xmax \geq x1 + (nx - 1) dx$.

Example: simple matrix

If a simple matrix has x equal to column number and y equal to row number, it has the following attributes:

$xmin = 1; xmax = nx; dx = 1; x1 = 1;$
 $ymin = 1; ymax = ny; dy = 1; y1 = 1;$

Example: sampled signal

If the matrix represents a sampled signal of 1 second duration with a sampling frequency of 10 kHz, it has the following attributes:

$xmin = 0.0; xmax = 1.0; nx = 10000 ; dx = 1.0 \cdot 10^{-4}; x1 = 0.5 \cdot 10^{-4};$
 $ymin = 1; ymax = 1; ny = 1; dy = 1; y1 = 1;$

Example: complex signal

If the matrix represents a complex spectrum derived with an FFT from the sound of example 2, it has the following attributes:

$xmin = 0.0; xmax = 5000.0; nx = 8193 ; dx = 5000.0 / 8192; x1 = 0.0;$
 $ny = 2$ (real and imaginary part);
 $ymin = 1$ (first row, real part);
 $ymax = 2$ (second row, imaginary part);
 $dy = 1; y1 = 1;$ (so that y is equal to row number)

Links to this page

- [BarkFilter](#)
- [CC: To Matrix](#)
- [Eigen & Matrix: Project...](#)
- [FFNet: Activation](#)
- [FFNet: Pattern](#)
- [FormantFilter](#)
- [Formulas 1.7. Formulas for creation](#)
- [Formulas 7. Attributes of objects](#)
- [LPC: To Matrix](#)
- [Matrix: Paint cells...](#)
- [Matrix: Set value...](#)
- [Matrix: Solve equation...](#)
- [Matrix: To TableOfReal](#)
- [MelFilter](#)
- [Pattern](#)
- [Spectrum](#)

© *ppgb*, February 16, 2003

Table

One of the types of objects in PRAAT. See the Statistics tutorial.

Links to this page

- [Create formant table \(Peterson & Barney 1952\)](#)
- [Create formant table \(Pols & Van Nierop 1973\)](#)
- [Formulas 7. Attributes of objects](#)
- [What's new?](#)

© *ppgb*, March 16, 2003

Create Sound from tone complex...

A command in the New menu to create a Sound as the sum of a number of sine waves with equidistant frequencies.

Arguments

Name

the name of the resulting Sound object.

Starting time, End time (s)

the time domain of the resulting Sound.

Sampling frequency (Hz)

the sampling frequency of the resulting Sound.

Phase

determines whether the result is a sum of *sines* or a sum of *cosines*, i.e., whether the zero crossings or the maxima of the components are synchronized. This choice has little perceptual consequences.

Frequency step (Hz)

the distance between the components. In first approximation, this is the perceived fundamental frequency.

First frequency (Hz)

the lowest frequency component. If you supply a value of 0, *firstFrequency* is taken equal to *frequencyStep*.

Ceiling (Hz)

the frequency above which no components are used. If you supply a value of 0 or a value above the Sound's Nyquist frequency, *ceiling* is taken equal to the Nyquist frequency.

Number of components

determines how many sinusoids are used. If you supply a value of 0 or a very high value, the maximum number of components is used, limited by *ceiling*.

Example 1: a pulse train

A series of pulses at regular intervals, sampled after low-pass filtering at the Nyquist frequency, can be regarded as a sum of cosine waves. For instance, a 100-Hz pulse train, sampled at 22050 Hz, can be created with:

```
Create Sound from tone complex... train 0 1 22050 Cosine 100 0 0 0
```

Supplying the value 0 for *firstFrequency* yields an unshifted harmonic complex.

Example 2: a shifted harmonic complex

Some experiments on human pitch perception (*residue pitch*) use a number of sinusoidal components with harmonically related frequencies that are all shifted by a constant amount.

For instance, to get a sum of sine waves with frequencies 105 Hz, 205 Hz, and 305 Hz, you would use:

```
Create Sound from tone complex... train 0.3 1 22050 Sine 100 105 0 3
```

or

```
Create Sound from tone complex... train 0.3 1 22050 Sine 100 105 350 0
```

whichever you prefer.

Some of these experiments are described in Plomp (1967) and Patterson & Wightman (1976).

Algorithm

For the 'sine' phase, the resulting Sound is given by the following formula:

$$x(t) = \sum_{i=1..numberOfComponents} \sin(2\pi \cdot (firstFrequency + (i-1) \cdot frequencyStep) \cdot t)$$

More flexibility?

Suppose you wanted to vary the relative strengths of the frequency components. You could achieve this by creating a Sound with the command discussed here, take its Fourier transform, run a formula on the resulting Spectrum, and take the inverse Fourier transform.

A more general approach is described shortly.

Suppose you need a sum of sine waves with frequencies 105, 205, 305, ..., 1905 Hz, and with relative amplitudes 1, 1/2, 1/3, ..., 1/19. You could build a script that computes the various components, and add them to each other as you go along. Instead of calling 19 scripts, however, you can achieve this with the following more general script:

```
form Add waves with decreasing amplitudes
  Number_of_components 19
endform
# Create a Matrix with frequency and amplitude information in each row:
Create simple Matrix... freqAndGain number_of_components 2 0
Formula... if col = 1 then row * 100 + 5 else 1 / row fi
# Create a large Matrix with all the component sine waves:
Create Matrix... components 0 1 10000 1e-4 0.5e-4 1
number_of_components number_of_components 1 1 0
Formula... Matrix_freqAndGain [2] * sin (2 * pi * Matrix_freqAndGain
[1] * x)
# Integrate:
Formula... self + self [row - 1, col]
# Publish last row:
To Sound (slice)... number_of_components
Scale amplitudes... 0.99
```

© ppgb, December 23, 2002

Create Sound from gamma-tone...

A command to create a Sound as a gamma-tone.

Arguments

Name

the name of the resulting Sound object.

Minimum time, Maximum time (s)

the start and end time of the resulting Sound.

Sample rate (Hz)

the sampling frequency of the resulting Sound.

Gamma

the exponent of the polynomial.

Frequency (Hz), Bandwidth (Hz)

determine the frequency and damping of the cosine wave in the gamma-tone.

Initial phase (radians)

the initial phase of the sine wave.

Addition factor (default: 0)

determines the degree of asymmetry in the spectrum of the gamma-tone. The zero default value gives a gamma-tone. A value unequal to zero results in a so called *gamma*-chirp. A negative value is used in auditory filter modeling to guarantee the usual direction of filter asymmetry, which corresponds to an upward glide in instantaneous frequency.

Scale amplitudes

determines whether the amplitudes will be scaled to fit in the range (-1, 1).

Purpose

to create a Sound according to the following formula:

$$t^{\gamma-1} e^{-2\pi \cdot \text{bandwidth} \cdot t} \cos(2\pi \cdot \text{frequency} \cdot t + \text{additionFactor} \cdot \ln(t) + \text{initialPhase}),$$

The *gamma chirp* function has a monotonically frequency-modulated carrier (the chirp) with instantaneous frequency

$$\text{instantaneousFrequency}(t) = \text{frequency} + \text{additionFactor} / (2 \cdot \pi \cdot t)$$

and an envelope that is a gamma distribution function. It is a theoretically optimum auditory filter, in the sense that it leads to minimal uncertainty in the joint time and scale representation of auditory signal analysis.

For faithful modeling of the inner ear, Irino & Patterson (1996) conclude that a value of approximately 1.5 * ERB (*frequency*) is appropriate for *bandwidth*. ERB stands for equivalent rectangular bandwidth. Their formula for ERB is:

$$\text{ERB}(f) = 6.23 \cdot 10^{-6} f^2 + 93.39 \cdot 10^{-3} f + 28.52.$$

To avoid aliasing in the chirp sound, a sound is only generated during times where the instantaneous frequency is greater than zero and smaller than the Nyquist frequency.

Links to this page

- [New menu](#)
-

© *djmw*, April 7, 2004

Create Sound from Shepard tone...

One of the commands that create a Sound.

Arguments

Name

the name of the resulting Sound object.

Minimum time, Maximum time (s)

the start and end time of the resulting Sound.

Sample rate (Hz)

the sampling frequency of the resulting Sound.

Lowest frequency (Hz)

the frequency of the lowest component.

Number of components

the number of frequency components.

Frequency change (semitones/s)

determines how long it takes to change all frequencies by one octave.

Amplitude range (dB)

determines the relative size of the maximum and the minimum amplitude of components.

Purpose

to create a Sound that is a continuous variant of the sound sequences used by Shepard (1964) in his experiment about the circularity in judgments of relative pitch.

The tone consists of many sinusoidal components whose frequencies increase exponentially in time. All frequencies are always at successive intervals of an octave and sounded simultaneously. Thus the frequency of each component above the lowest is at each moment in time exactly twice the frequency of the one just below. The amplitudes are large for the components of intermediate frequency only, and tapered off gradually to subthreshold levels for the components at the highest and lowest extremes of frequency.

The Sound is generated according to the following specification:

$$s(t) = \sum_{i=1..numberOfComponents} A_i(t) \sin(\arg_i(t)), \text{ where}$$

$$\arg_i(t) = \int 2\pi f_i(t) dt, \text{ and}$$

$$f_i(t) = \text{lowestFrequency} \cdot 2^{(i-1 + t/(12/\text{frequencyChange_st}))}, \text{ with}$$

$A_i(t) = L_{\min} + (1 - L_{\min}) (1 - \cos 2\pi\theta_i(t)) / 2$, with,

$\theta_i(t) = \ln ((f_i(t) + 1 \text{ Hz}) / (\text{lowestFrequency} + 1 \text{ Hz})) / \ln ((\text{maximumFrequency} + 1 \text{ Hz}) / (\text{lowestFrequency} + 1 \text{ Hz}))$

$L_{\min} = 10^{-\text{amplitudeRange}/10}$, and,

$\text{maximumFrequency} = \text{lowestFrequency} \cdot 2^{\text{numberOfComponents}-1}$

Links to this page

- [New menu](#)
-

© djmw, April 7, 2004

Sound: Play

A command to play Sound objects.

Availability

You can choose this command after selecting one or more Sounds.

Purpose

To play the selected Sounds through the internal or external loudspeakers, the headphones, or the analog or digital outputs of your computer.

Behaviour

All of the Sounds selected are played, in the order in which they appear in the list. If the sampling frequency of the Sound does not match any of the system's sampling frequencies, a fast but inaccurate conversion is performed via linear interpolation.

Usage

The choice of the output device(s) depends on the settings in your Audio Control Panel or (on HP) the setting of the **Use internal loudspeaker...** preference (in the 'Control' menu).

© ppgb, September 11, 1996

PointProcess: Hum

A command to hear a PointProcess.

Algorithm

A Sound is created with the algorithm described at PointProcess: To Sound (hum)....

This sound is then played.

© *ppgb*, March 30, 1997

PointProcess: Play

A command to hear a PointProcess.

Algorithm

A Sound is created with the algorithm described at PointProcess: To Sound (pulse train)....

This sound is then played.

© *ppgb*, March 30, 1997

PointEditor

One of the Editors in PRAAT, for viewing and manipulating a PointProcess object, which is optionally shown together with a Sound object.

Objects

The editor shows:

- The Sound, if you selected a Sound object together with the PointProcess object before you clicked "Edit".
- The PointProcess; vertical blue lines represent the points.

Playing

To play (a part of) the *resynthesized* sound (pulse train): click on any of the 8 buttons below and above the drawing area, or choose a Play command from the View menu.

To play the *original* sound instead, use Shift-click.

Adding a point

Click at the desired time location, and choose "Add point at cursor" or type **Command-P**.

Removing points

To remove one or more points, make a time selection and choose **Remove point(s)** from the **Point** menu. If there is no selection, the point nearest to the cursor is removed.

Links to this page

- [Types of objects](#)

© ppgb, March 16, 2003

PitchTierEditor

One of the Editors in PRAAT, for viewing and manipulating a PitchTier object, which is optionally shown together with a Sound object.

Objects

The editor shows:

- The Sound, if you selected a Sound object together with the PitchTier object before you clicked **Edit**.
- The PitchTier: blue points connected with blue lines.

Playing

To play (a part of) the *resynthesized* sound: click on any of the 8 buttons below and above the drawing area, or choose a Play command from the View menu.

To play the *original* sound instead, use Shift-click.

Adding a point

Click at the desired time location, and choose **Add point at cursor** or type **Command-P**.

Removing points

To remove one or more pitch points, make a time selection and choose **Remove point(s)** from the **Point** menu. If there is no selection, the point nearest to the cursor is removed.

Links to this page

- [Types of objects](#)

© ppgb, March 16, 2003

Get number of samples

A command that becomes available in the **Query** menu if you select a Sound or LongSound object.

The Info window will tell you the total number of time samples in this object.

Example

If the sampling frequency is 44100 Hertz, a recording with a duration of 60 seconds will contain 2,646,000 samples.

Details for hackers

If you select a Sound or LongSound and click Inspect, you can see how the number of samples is stored in the object: it is the **nx** attribute.

© *ppgb*, April 20, 2004

Get sampling period

A command available in the **Query** menu if you select a Sound. The Info window will tell you the sampling period in seconds.

Usage

You will not often choose this command with the mouse, since the sampling period is included in the information that you get by clicking the **Info** button. This command is probably more useful in a Praat script:

```
select Sound hello
samplingPeriod = Get sampling period
```

Details for hackers

With Inspect, you can see how the sampling period is stored in a **Sound** object: it is the **dx** attribute.

Links to this page

- [Get sampling frequency](#)
-

© *ppgb*, April 20, 2004

Get sampling frequency

A command available in the **Query** menu if you select a Sound. The Info window will tell you the sampling frequency in Hertz.

Usage

You will not often choose this command with the mouse, since the sampling frequency is included in the information that you get by clicking the **Info** button. This command is probably more useful in a Praat script:

```
select Sound hello
samplingFrequency = Get sampling frequency
```

Algorithm

The sampling frequency is defined as $1 / (\Delta t)$, where Δt is the sampling period. See Get sampling period.

© *ppgb*, April 20, 2004

Get time from sample number...

A command that becomes available in the **Query** menu if you select a Sound or LongSound object.

The Info window will tell you the time (in seconds) associated with the sample number that you specify.

Setting

Sample number

the sample number whose time is sought.

Algorithm

the result is

$$t_1 + (\text{sample_number} - 1) \cdot \Delta t$$

where t_1 is the time associated with the first sample, and Δt is the sampling period.

Details for hackers

If you select a Sound or LongSound and click Inspect, you can see how the relation between sample numbers and times is stored in the object: t_1 is the **x1** attribute, and Δt is the **dx** attribute.

© ppgb, April 20, 2004

Get sample number from time...

A command that becomes available in the **Query** menu if you select a Sound or LongSound object.

The Info window will tell you the sample number belonging to the time that you specify. The result is presented as a real number.

Setting

Time (seconds)

the time for which you want to know the sample number.

Example

If the sound has a sampling frequency of 10 kHz, the sample number associated with a time of 0.1 seconds will usually be 1000.5.

Scripting

You can use this command to put the nearest sample number into a script variable:

```
select Sound hallo
sampleNumber = Get sample number from time... 0.1
nearestSample = round (sampleNumber)
```

In this case, the value will not be written into the Info window. To round down or up, use

```
leftSample = floor (sampleNumber)
rightSample = ceiling (sampleNumber)
```

Algorithm

the result is

$$1 + (time - t_1) / \Delta t$$

where t_1 is the time associated with the first sample, and Δt is the sampling period.

Details for hackers

If you select a Sound or LongSound and click Inspect, you can see how the relation between sample numbers and times is stored in the object: t_1 is the **x1** attribute, and Δt is the **dx** attribute.

© *ppgb*, May 5, 2004

Sound: Get value at time...

A query to the selected Sound object.

Return value

an estimate of the amplitude (sound pressure in Pascal) at a specified time. If that time is outside the samples of the Sound, the result is equal to the value of the nearest sample; otherwise, the result is an interpolated value.

Arguments

Time (s)

the time at which the value is to be evaluated.

Interpolation

the interpolation method, see vector value interpolation. The standard is Sinc70 because a Sound object is normally a sampled band-limited signal, which can be seen as a sum of sinc functions.

© ppgb, September 16, 2003

Sound: Get value at sample number...

A query to the selected Sound object.

Behaviour

the amplitude (sound pressure in Pascal) at a specified sample number. If the sample number is less than 1 or greater than the number of samples, the result is undefined; otherwise, it is $z[1][sample_number]$.

Setting

Sample number

the sample number at which the value is to be evaluated.

© ppgb, April 20, 2004

Sound: Get minimum...

A query to the selected Sound object.

Return value

the minimum amplitude (sound pressure in Pascal) within a specified time window.

Arguments

From time (s), *To time* (s)

the time window. Values outside this window are ignored, except for purposes of interpolation. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Sinc70 because a Sound object is normally a sampled band-limited signal, which can be seen as a sum of sinc functions.

© ppgb, September 16, 2003

Sound: Get time of minimum...

A query to the selected Sound object.

Behaviour

the time (in seconds) associated with the minimum amplitude.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored, except for interpolation. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Sinc70 because a Sound object is normally a sampled band-limited signal, which can be seen as a sum of sinc functions.

© ppgb, September 16, 2003

Sound: Get maximum...

A query to the selected Sound object.

Return value

the maximum amplitude (sound pressure in Pascal) within a specified time window.

Arguments

From time (s), *To time* (s)

the time window. Values outside this window are ignored, except for purposes of interpolation. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Sinc70 because a Sound object is normally a sampled band-limited signal, which can be seen as a sum of sinc functions.

© ppgb, September 16, 2003

Sound: Get time of maximum...

A query to the selected Sound object.

Behaviour

the time (in seconds) associated with the maximum amplitude.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored, except for interpolation. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Sinc70 because a Sound object is normally a sampled band-limited signal, which can be seen as a sum of sinc functions.

© ppgb, September 16, 2003

Sound: Get absolute extremum...

A query to the selected Sound object.

Return value

the absolute extremum (in Pascal) within a specified time window.

Arguments

From time (s), *To time* (s)

the time window. Values outside this window are ignored, except for interpolation. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Interpolation

the interpolation method (None, Parabolic, Sinc) of the vector peak interpolation. The standard is Sinc70 because a Sound object is normally a sampled band-limited signal, which can be seen as a sum of sinc functions.

© ppgb, September 16, 2003

Sound: Get nearest zero crossing...

A query to the selected Sound object.

Return value

the time associated with the zero crossing nearest to a specified time. It is undefined if there are no zero crossings or if the specified time is outside the time domain of the sound. Linear interpolation is used between sample points.

Argument

Time (s)

the time for which you want to get the time of the nearest zero crossing.

Links to this page

- [What's new?](#)
-

© ppgb, December 12, 2002

Sound: Get mean...

A query to the selected Sound object.

Return value

the mean amplitude (sound pressure in Pascal) within a specified time window.

Arguments

From time (s), *To time* (s)

the time window. Values outside this window are ignored. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Algorithm

The mean amplitude between the times t_1 and t_2 is defined as

$$1/(t_2 - t_1) \int_{t_1}^{t_2} dt x(t)$$

where $x(t)$ is the amplitude of the sound. For our discrete Sound object, this mean is approximated by

$$1/n \sum_{i=m..m+n-1} x_i$$

where n is the number of sample centres between t_1 and t_2 .

© ppgb, October 16, 1999

Sound: Get root-mean-square...

A query to the selected Sound object.

Return value

the root-mean-square (rms) value of the sound pressure, expressed in Pascal.

Arguments

From time (s), *To time* (s)

the time window. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Algorithm

The root-mean-square value is defined as

$$\sqrt{\left(1/T \int dt x^2(t) \right)}$$

where $x(t)$ is the amplitude of the sound, and T its duration. For our discrete Sound object, the rms value is computed as

$$\sqrt{\left(1/n \sum_{i=1..n} x_i^2 \right)}$$

where n is the number of samples.

© ppgb, October 16, 1999

Sound: Get standard deviation...

A query to the selected Sound object.

Return value

the standard deviation (in Pascal) of the sound pressure within a specified window. If the sound contains less than 2 samples, the value is undefined.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Algorithm

The standard deviation is defined as

$$1/(t_2 - t_1) \int_{t_1}^{t_2} dt (x(t) - \mu)^2$$

where $x(t)$ is the amplitude of the sound, T its duration, and μ its mean. For our discrete Sound object, the standard deviation is approximated by

$$1/(n-1) \sum_{i=m..m+n-1} (x_i - \mu)^2$$

where n is the number of sample centres between t_1 and t_2 . Note the "minus 1".

© ppgb, October 16, 1999

Sound: Get energy...

A query to the selected Sound object.

Return value

the energy. If the unit of sound amplitude is Pa (Pascal), the unit of energy will be Pa²·s.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Algorithm

The energy is defined as

$$\int dt x^2(t)$$

where $x(t)$ is the amplitude of the sound. For our discrete Sound object, the energy is computed as

$$\Delta t \sum_{i=1..n} x_i^2$$

where Δt is the sampling period and n is the number of samples.

See also

For an interpretation of the energy as the sound energy in air, see Sound: Get energy in air. For the power, see Sound: Get power....

© ppgb, October 16, 1999

Sound: Get power...

A query to the selected Sound object.

Return value

the power within a specified time window. If the unit of sound amplitude is Pa (Pascal), the unit of power will be Pa².

Arguments

From time (s), To time (s)

the time window. Values outside this window are ignored. If *To time* is not greater than *From time*, the entire time domain of the sound is considered.

Algorithm

The power is defined as

$$1/T \int dt x^2(t)$$

where $x(t)$ is the amplitude of the sound, and T its duration. For our discrete Sound object, the power is computed as

$$1/n \sum_{i=1..n} x_i^2$$

where n is the number of samples.

See also

For an interpretation of the power as the sound power in air, see Sound: Get power in air. For the total energy, see Sound: Get energy....

© ppgb, October 16, 1999

Sound: Get energy in air

A query to the selected Sound object.

Return value

The energy in air, expressed in Joule/m².

Algorithm

The energy of a sound in air is defined as

$$1 / (\rho c) \int dt x^2(t)$$

where $x(t)$ is the sound pressure in units of Pa (Pascal), ρ is the air density (apx. 1.14 kg/m³), and c is the velocity of sound in air (apx. 353 m/s). For our discrete Sound object, the energy is computed as

$$\Delta t / (\rho c) \sum_{i=1..n} x_i^2$$

where Δt is the sampling period and n is the number of samples.

See also

For an air-independent interpretation of the energy, see Sound: Get energy.... For the power, see Sound: Get power in air.

© ppgb, October 16, 1999

Sound: Get power in air

A query to the selected Sound object.

Return value

the power in air, expressed in Watt/m².

Algorithm

The power of a sound in air is defined as

$$1 / (\rho c T) \int dt x^2(t)$$

where $x(t)$ is the sound pressure in units of Pa (Pascal), ρ is the air density (apx. 1.14 kg/m³), c is the velocity of sound in air (apx. 353 m/s), and T is the duration of the sound. For our discrete Sound object, the power is computed as

$$1 / (\rho c n) \sum_{i=1..n} x_i^2$$

where n is the number of samples.

For an air-independent interpretation of the power, see Sound: Get power.... For the energy, see Sound: Get energy in air. For the intensity in dB, see Sound: Get intensity (dB).

© ppgb, October 16, 1999

Sound: Get intensity (dB)

A query to the selected Sound object.

Return value

the intensity in air, expressed in dB relative to the auditory threshold.

Algorithm

The intensity of a sound in air is defined as

$$10 \log_{10} (1 / (T P_0^2) \int dt x^2(t))$$

where $x(t)$ is the sound pressure in units of Pa (Pascal), T is the duration of the sound, and $P_0 = 2 \cdot 10^{-5}$ Pa is the auditory threshold pressure. For our discrete Sound object, the intensity is computed as

$$10 \log_{10} (1 / (n P_0^2) \sum_{i=1..n} x_i^2)$$

where n is the number of samples.

See also

For the intensity in Watt/m², see Sound: Get power in air. For an auditory intensity, see Excitation: Get loudness.

© ppgb, October 16, 1999

Matrix: Formula...

A command for changing the data in all selected Matrix objects.

See the Formulas tutorial for examples and explanations.

Links to this page

- [Create Matrix...](#)
- [Create simple Matrix...](#)
- [Formant: Formula \(bandwidths\)...](#)
- [Formant: Formula \(frequencies\)...](#)
- [Formula...](#)
- [Matrix: Draw as squares...](#)
- [Sound](#)
- [Sound: Filter \(formula\)...](#)
- [TableOfReal: Select columns where row...](#)

© *ppgb*, December 6, 2002

Sound: Set value at sample number...

A command to change a specified sample of the selected Sound object.

Settings

Sample number

the sample whose value is to be changed. Specify any value between 1 and the number of samples in the Sound. If you specify a value outside that range, you will get an error message.

New value

the value that is to be put into the specified sample.

Scripting

Example:

```
select Sound hallo
Set value at sample number... 100 1/2
```

This sets the value of the 100th sample to 0.5.

© ppgb, April 20, 2004

Sound: To Pitch (ac)...

A command that creates a Pitch object from every selected Sound object.

Purpose

to perform a pitch analysis based on an autocorrelation method.

Usage

Normally, you will instead use Sound: To Pitch..., which uses the same method. The command described here is mainly for experimenting with the parameters, or for the analysis of non-speech signals, which may require different standard settings of the parameters.

Algorithm

The algorithm performs an acoustic periodicity detection on the basis of an accurate autocorrelation method, as described in Boersma (1993). This method is more accurate, noise-resistant, and robust, than methods based on cepstrum or combs, or the original autocorrelation methods. The reason why other methods were invented, was the failure to recognize the fact that if you want to estimate a signal's short-term autocorrelation function on the basis of a windowed signal, you should divide the autocorrelation function of the windowed signal by the autocorrelation function of the window:

$$r_x(\tau) \approx r_{xw}(\tau) / r_w(\tau)$$

Arguments

The arguments that control the recruitment of the candidates are:

Time step (standard value: 0.0)

the measurement interval (frame duration), in seconds. If you supply 0, PRAAT will use a time step of $0.75 / (\textit{pitch floor})$, e.g. 0.01 seconds if the pitch floor is 75 Hz; in this example, PRAAT computes 100 pitch values per second.

Pitch floor (standard value: 75 Hz)

candidates below this frequency will not be recruited. This parameter determines the effective length of the analysis window: it will be 3 longest periods long, i.e., if the pitch floor is 75 Hz, the window will be effectively $3/75 = 0.04$ seconds long.

Note that if you set the time step to zero, the analysis windows for consecutive measurements will overlap appreciably: PRAAT will always compute 4 pitch values within one window length, i.e., the degree of *oversampling* is 4.

Very accurate (standard value: *off*)

if *off*, the window is a Hanning window with a physical length of $3 / (\textit{pitch floor})$. If *on*, the window is a Gaussian window with a physical length of $6 / (\textit{pitch floor})$, i.e. twice the effective length.

A post-processing algorithm seeks the cheapest path through the candidates. The arguments that determine the cheapest path are:

Pitch ceiling (standard value: 600 Hz)

candidates above this frequency will be ignored.

Silence threshold (standard value: 0.03)

frames that do not contain amplitudes above this threshold (relative to the global maximum amplitude), are probably silent.

Voicing threshold (standard value: 0.45)

the strength of the unvoiced candidate, relative to the maximum possible autocorrelation. To increase the number of unvoiced decisions, increase this value.

Octave cost (standard value: 0.01 per octave)

degree of favouring of high-frequency candidates, relative to the maximum possible autocorrelation. This is necessary because even (or: especially) in the case of a perfectly periodic signal, all undertones of F_0 are equally strong candidates as F_0 itself. To more strongly favour recruitment of high-frequency candidates, increase this value.

Octave-jump cost (standard value: 0.35)

degree of disfavouring of pitch changes, relative to the maximum possible autocorrelation. To decrease the number of large frequency jumps, increase this value. In contrast with what is described in the article, this value will be corrected for the time step: multiply by $0.01 \text{ s} / \textit{TimeStep}$ to get the value in the way it is used in the formulas in the article.

Voiced / unvoiced cost (standard value: 0.14)

degree of disfavouring of voiced/unvoiced transitions, relative to the maximum possible autocorrelation. To decrease the number of voiced/unvoiced transitions, increase this value. In contrast with what is described in the article, this value will be corrected for the time step: multiply by $0.01 \text{ s} / \textit{TimeStep}$ to get the value in the way it is used in the formulas in the article.

Links to this page

- [Advanced pitch settings...](#)
 - [FAQ: Pitch analysis](#)
 - [Fast Fourier Transform](#)
 - [Periodicity submenu](#)
 - [PitchEditor](#)
 - [Sound: To Pitch \(cc\)...](#)
 - [Sound: To PointProcess \(periodic, cc\)...](#)
 - [Sound: To PointProcess \(periodic, peaks\)...](#)
 - [What's new?](#)
-

© ppgb, September 16, 2003

Sound: To Pitch (cc)...

A command that creates a Pitch object from every selected Sound object.

Purpose

to perform a pitch analysis based on a cross-correlation method.

Algorithm

The algorithm performs an acoustic periodicity detection on the basis of a forward cross-correlation analysis.

Arguments

Time step (standard value: 0.0)

the measurement interval (frame duration), in seconds. If you supply 0, PRAAT will use a time step of $0.25 / (\textit{pitch floor})$, e.g. 0.00333333 seconds if the pitch floor is 75 Hz; in this example, PRAAT computes 300 pitch values per second.

Pitch floor (standard value: 75 Hz)

candidates below this frequency will not be recruited. This parameter determines the length of the analysis window: it will be 1 longest period long, i.e., if the pitch floor is 75 Hz, the window will be $1/75 = 0.01333333$ seconds long.

Note that if you set the time step to zero, the analysis windows for consecutive measurements will overlap appreciably: PRAAT will always compute 4 pitch values within one window length, i.e., the degree of *oversampling* is 4.

The other arguments are the same as for Sound: To Pitch (ac)...

Usage

The preferred method for speech is Sound: To Pitch.... The command described here is mainly for experimenting, or for applications where you need short time windows.

Links to this page

- [Periodicity submenu](#)

© ppgb, September 16, 2003

Sound: To Pitch (shs)...

A command that creates a Pitch object from every selected Sound object.

Purpose

to perform a pitch analysis based on a spectral compression model. The concept of this model is that each spectral component not only activates those elements of the central pitch processor that are most sensitive to the component's frequency, but also elements that have a lower harmonic relation with this component. Therefore, when a specific element of the central pitch processor is most sensitive at a frequency f_0 , it receives contributions from spectral components in the signal at integral multiples of f_0 .

Algorithm

The spectral compression consists of the summation of a sequence of harmonically compressed spectra. The abscissa of these spectra is compressed by an integral factor, the rank of the compression. The maximum of the resulting sum spectrum is the estimate of the pitch. Details of the algorithm can be found in Hermes (1988)

Arguments

Time step (default 0.01 s)

the measurement interval (frame duration), in seconds.

Minimum pitch (default 50 Hz)

candidates below this frequency will not be recruited. This parameter determines the length of the analysis window.

Max. number of candidates (default 15)

The maximum number of candidates that will be recruited.

Maximum frequency (default 1250 Hz)

higher frequencies will not be considered.

Max. number of subharmonics (default 15)

the maximum number of harmonics that add up to the pitch.

Compression factor (default 0.84)

the factor by which successive compressed spectra are multiplied before the summation.

Number of points per octave (default 48)

determines the sampling of the logarithmic frequency scale.

Ceiling (default 500 Hz)

candidates above this frequency will be ignored.

© djmw, April 2, 1997

Sound: To Harmonicity (ac)...

A command that creates a Harmonicity object from every selected Sound object.

Purpose

to perform a short-term HNR analysis.

Algorithm

The algorithm performs an acoustic periodicity detection on the basis of an accurate autocorrelation method, as described in Boersma (1993).

Arguments

Time step (standard value: 0.01 s)

the measurement interval (frame duration), in seconds.

Minimum pitch (standard value: 75 Hz)

determines the length of the analysis window.

Silence threshold (standard value: 0.1)

frames that do not contain amplitudes above this threshold (relative to the global maximum amplitude), are considered silent.

Number of periods per window (standard value: 4.5)

4.5 is best for speech: HNR values up to 37 dB are guaranteed to be detected reliably; 6 periods per window raises this figure to more than 60 dB, but the algorithm becomes more sensitive to dynamic changes in the signal.

Usage

You will normally use Sound: To Harmonicity (cc)... instead of this command, because that has a much better time resolution (though its sensitivity is 60, not 80 dB).

Links to this page

- [Periodicity submenu](#)
- [Voice 4. Additive noise](#)

© ppgb, September 16, 2003

Sound: To Harmonicity (cc)...

A command that creates a Harmonicity object from every selected Sound object.

Purpose

to perform a short-term HNR analysis.

Algorithm

The algorithm performs an acoustic periodicity detection on the basis of a forward cross-correlation analysis. For information on the arguments, see Sound: To Harmonicity (ac)....

Links to this page

- [Periodicity submenu](#)
- [Voice 4. Additive noise](#)

© *ppgb*, October 3, 1996

Sound: To PointProcess (periodic, cc)...

A command that analyses the selected Sound objects, and creates PointProcess objects.

This command combines the actions of Sound: To Pitch (ac)... and Sound & Pitch: To PointProcess (cc).

© *ppgb*, March 9, 2003

Sound: To PointProcess (periodic, peaks)...

A command that analyses the selected Sound objects, and creates PointProcess objects.

This command combines the actions of Sound: To Pitch (ac)... and Sound & Pitch: To PointProcess (peaks)...

Links to this page

- [What's new?](#)

© *ppgb*, March 9, 2003

Sound & Pitch: To PointProcess (cc)

A command to create a PointProcess from the selected Sound and Pitch objects.

Purpose

to interpret an acoustic periodicity contour as the frequency of an underlying point process (such as the sequence of glottal closures in vocal-fold vibration).

Algorithm

The voiced intervals are determined on the basis of the voiced/unvoiced decisions in the Pitch object. For every voiced interval, a number of *points* (or glottal pulses) is found as follows:

1. The first point t_1 is the absolute extremum of the amplitude of the **Sound**, between $t_{mid} - T_0 / 2$ and $t_{mid} + T_0 / 2$, where t_{mid} is the midpoint of the interval, and T_0 is the period at t_{mid} , as can be interpolated from the **Pitch** contour.
2. From this point, we recursively search for points t_i to the left until we reach the left edge of the interval. These points must be located between $t_{i-1} - 1.2 T_0(t_{i-1})$ and $t_{i-1} - 0.8 T_0(t_{i-1})$, and the cross-correlation of the amplitude in its environment $[t_i - T_0(t_i) / 2; t_i + T_0(t_i) / 2]$ with the amplitude of the environment of the existing point t_{i-1} must be maximal (we use parabolic interpolation between samples of the correlation function).
3. The same is done to the right of t_1 .
4. Though the voiced/unvoiced decision is initially taken by the **Pitch** contour, points are removed if their correlation value is less than 0.3; furthermore, one extra point may be added at the edge of the voiced interval if its correlation value is greater than 0.7.

Links to this page

- [Manipulation](#)
- [Sound & Pitch: To PointProcess \(peaks\)...](#)
- [Sound: To PointProcess \(periodic, cc\)...](#)

© ppgb, March 22, 1998

Sound & Pitch: To PointProcess (peaks)...

A command to create a PointProcess from the selected Sound and Pitch objects.

Purpose

to interpret an acoustic periodicity contour as the frequency of an underlying point process (such as the sequence of glottal closures in vocal-fold vibration).

Algorithm

The voiced intervals are determined on the basis of the voiced/unvoiced decisions in the Pitch object. For every voiced interval, a number of *points* (or glottal pulses) is found as follows:

1. The first point t_1 is the absolute extremum (or the maximum, or the minimum, depending on your *Include maxima* and *Include minima* settings) of the amplitude of the **Sound**, between $t_{mid} - T_0 / 2$ and $t_{mid} + T_0 / 2$, where t_{mid} is the midpoint of the interval, and T_0 is the period at t_{mid} , as can be interpolated from the **Pitch** contour.
2. From this point, we recursively search for points t_i to the left until we reach the left edge of the interval. These points are the absolute extrema (or the maxima, or the minima) between the times $t_{i-1} - 1.2 T_0(t_{i-1})$ and $t_{i-1} - 0.8 T_0(t_{i-1})$.
3. The same is done to the right of t_1 .

The periods that are found in this way are much more variable than those found by Sound & Pitch: To PointProcess (cc), and therefore less useful for PSOLA analysis.

Links to this page

- [Sound: To PointProcess \(periodic, peaks\)...](#)

© ppgb, March 9, 2003

Sound: To Formant (sl)...

A command that creates a Formant object from every selected Sound object. Not recommended for general use.

Purpose

to perform a short-term spectral analysis, approximating the spectrum of each frame by a number of formants.

Arguments

The same as with Sound: To Formant (burg)....

Algorithm

The algorithm is based on the implementation of the 'Split Levinson' algorithm by Willems (1986). This algorithm will always find the requested number of formants in every frame, even if they do not exist. The standard routine (Sound: To Formant (burg)...) yields much more reliable formant values, though it is more sensitive to the *Maximum formant* argument.

Because of the general funny behaviour of the Split-Levinson algorithm, we did not bother to implement an analysis of the bandwidths. They are all set arbitrarily to 50 Hz.

Links to this page

- [Formants & LPC submenu](#)

© ppgb, December 15, 2002

Sound: To LPC (autocorrelation)...

With this command you create a new LPC from every selected Sound, using "the autocorrelation" method.

Warning

You are advised not to use this command for formant analysis. For formant analysis, instead use **Sound: To Formant (burg)...**, which also works via LPC (linear predictive coding). This is because **Sound: To Formant (burg)...** lets you specify a maximum frequency, whereas the **To LPC** commands automatically use the Nyquist frequency as their maximum frequency. If you do use one of the **To LPC** commands for formant analysis, you may therefore want to downsample the sound first. For instance, if you want five formants below 5500 Hz but your Sound has a sampling frequency of 44100 Hz, you have to downsample the sound to 11000 Hz with the **Sound: Resample...** command. After that, you can use the **To LPC** commands, with a prediction order of 10 or 11.

Settings

Prediction order

the number of linear prediction coefficients, also called the *number of poles*. Choose this number at least twice as large as the number of spectral peaks that you want to detect.

Analysis window duration (s)

the effective duration of each analysis frame, in seconds.

Time step (s)

the time step between two consecutive analysis frames.

Pre-emphasis frequency (Hz)

a +6dB / octave filtering will be applied above this frequency. A pre-emphasis frequency of 48.47 Hz for a signal with a sampling frequency of 10 kHz approximately corresponds to a value of $a = 0.97$ for the filter $y_n = x_n - a \cdot x_{n-1}$. The relation between a and the pre-emphasis frequency is: $a = \exp(-2 \cdot \pi \cdot \text{preemphasisFrequency} / \text{samplingFrequency})$. If you do not want pre-emphasis, choose a frequency greater than the Nyquist frequency.

Algorithm

The autocorrelation algorithm is described in Markel & Gray (1976).

Links to this page

- [Formants & LPC submenu](#)
- [Sound: LPC analysis](#)

Sound: To LPC (covariance)...

With this command you create a new LPC from every selected Sound, using "the covariance" method.

Warning

You are advised not to use this command for formant analysis. For formant analysis, instead use **Sound: To Formant (burg)...**, which also works via LPC (linear predictive coding). This is because **Sound: To Formant (burg)...** lets you specify a maximum frequency, whereas the **To LPC** commands automatically use the Nyquist frequency as their maximum frequency. If you do use one of the **To LPC** commands for formant analysis, you may therefore want to downsample the sound first. For instance, if you want five formants below 5500 Hz but your Sound has a sampling frequency of 44100 Hz, you have to downsample the sound to 11000 Hz with the **Sound: Resample...** command. After that, you can use the **To LPC** commands, with a prediction order of 10 or 11.

Settings

Prediction order

the number of linear prediction coefficients, also called the *number of poles*. Choose this number at least twice as large as the number of spectral peaks that you want to detect.

Analysis window duration (s)

the effective duration of each analysis frame, in seconds.

Time step (s)

the time step between two consecutive analysis frames.

Pre-emphasis frequency (Hz)

a +6dB / octave filtering will be applied above this frequency. A pre-emphasis frequency of 48.47 Hz for a signal with a sampling frequency of 10 kHz approximately corresponds to a value of $a = 0.97$ for the filter $y_n = x_n - a \cdot x_{n-1}$. The relation between a and the pre-emphasis frequency is: $a = \exp(-2 \cdot \pi \cdot \text{preemphasisFrequency} / \text{samplingFrequency})$. If you do not want pre-emphasis, choose a frequency greater than the Nyquist frequency.

Algorithm

The covariance algorithm is described in Markel & Gray (1976).

Links to this page

- Formants & LPC submenu
- Sound: LPC analysis

Sound: To LPC (marple)...

With this command you create a new LPC from every selected Sound, using "Marple's" method.

Warning

You are advised not to use this command for formant analysis. For formant analysis, instead use **Sound: To Formant (burg)...**, which also works via LPC (linear predictive coding). This is because **Sound: To Formant (burg)...** lets you specify a maximum frequency, whereas the **To LPC** commands automatically use the Nyquist frequency as their maximum frequency. If you do use one of the **To LPC** commands for formant analysis, you may therefore want to downsample the sound first. For instance, if you want five formants below 5500 Hz but your Sound has a sampling frequency of 44100 Hz, you have to downsample the sound to 11000 Hz with the **Sound: Resample...** command. After that, you can use the **To LPC** commands, with a prediction order of 10 or 11.

Settings

Prediction order

the number of linear prediction coefficients, also called the *number of poles*. Choose this number at least twice as large as the number of spectral peaks that you want to detect.

Analysis window duration (s)

the effective duration of each analysis frame, in seconds.

Time step (s)

the time step between two consecutive analysis frames.

Pre-emphasis frequency (Hz)

a +6dB / octave filtering will be applied above this frequency. A pre-emphasis frequency of 48.47 Hz for a signal with a sampling frequency of 10 kHz approximately corresponds to a value of $a = 0.97$ for the filter $y_n = x_n - a \cdot x_{n-1}$. The relation between a and the pre-emphasis frequency is: $a = \exp(-2 \cdot \pi \cdot \text{preemphasisFrequency} / \text{samplingFrequency})$. If you do not want pre-emphasis, choose a frequency greater than the Nyquist frequency.

Tolerance 1

stop the iteration when $E(m) / E(0) < \text{Tolerance 1}$, where $E(m)$ is the prediction error for order m .

Tolerance 2

stop the iteration when $(E(m) - E(m-1)) / E(m-1) < \text{Tolerance 2}$.

Algorithm

The algorithm is described in Marple (1980).

Links to this page

- [Formants & LPC submenu](#)
 - [Sound: LPC analysis](#)
-

© *djmw*, January 26, 1997

Sound: Lengthen (PSOLA)...

A command to convert each selected Sound object into a longer new Sound object.

Arguments

Minimum frequency (Hz)

the minimum pitch used in the periodicity analysis. The standard value is 75 Hz. For the voice of a young child, set this to 150 Hz. The shortest voiceless interval in the PSOLA decomposition is taken as 1.5 divided by *minimum frequency*.

Maximum frequency (Hz)

the maximum pitch used in the periodicity analysis. The standard value is 600 Hz. For an adult male voice, set this to 300 Hz.

Factor

the factor with which the sound will be lengthened. The standard value is 1.5. If you take a value less than 1, the resulting sound will be shorter than the original. A value larger than 3 will not work.

Algorithm

Pitch-synchronous overlap-and-add.

Links to this page

- [What's new?](#)

© ppgb, September 16, 2003

Sound: Deepen band modulation...

A command to enhance the fast spectral changes, like F_2 movements, in each selected Sound object.

Arguments

Enhancement (dB)

the maximum increase in the level within each critical band. The standard value is 20 dB.

From frequency (Hz)

the lowest frequency that shall be manipulated. The bottom frequency of the first critical band that is to be enhanced. The standard value is 300 Hertz.

To frequency (Hz)

the highest frequency that shall be manipulated (the last critical band may be narrower than the others). The standard value is 8000 Hz.

Slow modulation (Hz)

the frequency f_{slow} below which the intensity modulations in the bands should not be expanded. The standard value is 3 Hz.

Fast modulation (Hz)

the frequency f_{fast} above which the intensity modulations in the bands should not be expanded. The standard value is 30 Hz.

Band smoothing (Hz)

the degree of overlap of each band into its adjacent bands. Prevents *ringing*. The standard value is 100 Hz.

Algorithm

Suppose we have the standard settings of the arguments. The resulting sound will be composed of the unfiltered part of the original sound, plus all manipulated bands.

First, the resulting sound becomes the original sound, stop-band filtered between 300 and 8000 Hz: after a forward Fourier transform, all values in the Spectrum at frequencies between 0 and 200 Hz and between 8100 Hz and the Nyquist frequency of the sound are retained unchanged. The spectral values at frequencies between 400 and 7900 Hz are set to zero. Between 200 and 400 Hz and between 7900 and 8100 Hz, the values are multiplied by a raised sine, so as to give a smooth transition without ringing in the time domain (the raised sine also allows us to view the spectrum as a sum of spectral bands). Finally, a backward Fourier transform gives us the filtered sound.

The remaining part of the spectrum is divided into *critical bands*, i.e. frequency bands one Bark wide. For instance, the first critical band runs from 300 to 406 Hz, the second from 406 to 520 Hz, and so on. Each critical band is converted to a pass-band filtered sound by means of the backward Fourier transform.

Each filtered sound will be manipulated, and the resulting manipulated sounds are added to the stop-band filtered sound we created earlier. If the manipulation is the identity transformation, the resulting sound will be equal to the original sound. But, of course, the manipulation does something different. Here are the steps.

First, we compute the local intensity of the filtered sound $x(t)$:

$$intensity(t) = 10 \log_{10} (x^2(t) + 10^{-6})$$

This intensity is subjected to a forward Fourier transform. In the frequency domain, we administer a band filter. We want to enhance the intensity modulation in the range between 3 and 30 Hz. We can achieve this by comparing the very smooth intensity contour, low-pass filtered at $f_{slow} = 3$ Hz, with the intensity contour that has enough temporal resolution to see the place-discriminating F_2 movements, which is low-pass filtered at $f_{fast} = 30$ Hz. In the frequency domain, the filter is

$$H(f) = \exp(-(\alpha f / f_{fast})^2) - \exp(-(\alpha f / f_{slow})^2)$$

where α equals $\sqrt{\ln 2} \approx 1 / 1.2011224$, so that $H(f)$ has its -6 dB points at f_{slow} and f_{fast} :

[sorry, no pictures yet in the web version of this manual]

Now, why do we use such a flat filter? Because a steep filter would show ringing effects in the time domain, dividing the sound into 30-ms chunks. If our filter is a sum of exponentials in the frequency domain, it will also be a sum of exponentials in the time domain. The backward Fourier transform of the frequency response $H(f)$ is the impulse response $h(t)$. It is given by

$$h(t) = 2\pi\sqrt{\pi} f_{fast} / \alpha \exp(-(\pi t f_{fast} / \alpha)^2) - 2\pi\sqrt{\pi} f_{slow} / \alpha \exp(-(\pi t f_{slow} / \alpha)^2)$$

This impulse response behaves well:

[sorry, no pictures yet in the web version of this manual]

We see that any short intensity peak will be enhanced, and that this enhancement will suppress the intensity around 30 milliseconds from the peak. Non-Gaussian frequency-domain filters would have given several maxima and minima in the impulse response, clearly an undesirable phenomenon.

After the filtered band is subjected to a backward Fourier transform, we convert it into power again:

$$power(t) = 10^{filtered / 2}$$

The relative enhancement has a maximum that is smoothly related to the basilar place:

$$ceiling = 1 + (10^{enhancement / 20} - 1) \cdot (1/2 - 1/2 \cos(\pi f_{midbark} / 13))$$

where $f_{midbark}$ is the mid frequency of the band. Clipping is implemented as

$$factor(t) = 1 / (1 / power(t) + 1 / ceiling)$$

Finally, the original filtered sound $x(t)$, multiplied by this factor, is added to the output.

Links to this page

- [What's new?](#)
-

© *ppgb*, September 16, 2003

Sounds: Concatenate

A command to concatenate all selected Sound objects into a single large Sound.

All sounds must have equal sampling frequencies. They are concatenated in the order that they appear in the list of objects.

How to concatenate directly to a file

If the resulting sound does not fit into memory, use one of the commands in the Write menu. See How to concatenate sound files.

© *ppgb*, January 23, 2000

PSOLA

Pitch-Synchronous Overlap and Add, a method for manipulating the pitch and duration of an acoustic speech signal.

PSOLA synthesis

When a Sound is created from a Manipulation object, the following steps are performed:

1. From the PitchTier, new points are generated along the entire time domain, with the method of PitchTier: To PointProcess.
2. The period information in the pulses is used to remove from the new *pulses* all points that lie within voiceless intervals (i.e., places where the distance between adjacent points in the original *pulses* is greater than 20 ms.
3. The voiceless parts are copied from the source Sound to the target Sound, re-using some parts if the local duration is greater than 1.
4. For each *target* point, we look up the nearest source point. A piece of the source Sound, centred around the source point, is copied to the target Sound at a location determined by the target point, using a bell-shaped window whose left-hand half-length is the minimum of the left-hand periods adjacent to the source and target points (and analogously for the right-hand half-length).

Links to this page

- [Manipulation: Get resynthesis \(PSOLA\)](#)
- [Manipulation: Play \(PSOLA\)](#)
- [ManipulationEditor](#)
- [Sound & Pitch: Change gender...](#)
- [Sound: Change gender...](#)
- [Types of objects](#)
- [What's new?](#)

© ppgb, March 30, 2001

Spectrum: To Sound (fft)

A command for creating a Sound object from every selected Spectrum object.

Algorithm

The algorithm is the continuous interpretation of the inverse Fast Fourier Transform. If the Spectrum is expressed in Pa/Hz, the Sound will be in Pascal. The frequency integral over the Sound equals the time integral over the Spectrum.

Behaviour

If you perform this command on a Spectrum object that was created earlier with Sound: To Spectrum (fft), the resulting Sound is equal to the Sound that was input to **Sound: To Spectrum (fft)**.

Links to this page

- [Sound: Filter \(formula\)...](#)
- [Sound: Filter \(pass Hann band\)...](#)
- [Sound: Filter \(stop Hann band\)...](#)

© *ppgb*, May 26, 2002

PointProcess: To Sound (hum)...

A command to convert every selected PointProcess into a Sound.

Algorithm

A Sound is created with the algorithm described at PointProcess: To Sound (pulse train).... This sound is then run through a sequence of second-order filters that represent five formants.

Links to this page

- [PointProcess: Hum](#)

© *ppgb*, March 30, 1997

AIFF and AIFC files

Ways for storing a Sound object on disk.

File format

Reading

Read from file... recognizes AIFF and AIFC files with 8-bit and 16-bit encoding, but not compressed AIFC files. It recognizes any sampling frequency. The two channels of stereo files are averaged. **Read two Sounds from AIFF file...** reads both channels separately and names them *left* and *right*.

The sample values are divided by $2^{(\text{numberOfBitsPerSample} - 1)}$, so that the amplitude of the resulting Sound is between -1.0 and +1.0; the maximum sound pressure level for a sine wave is therefore: $20 \cdot 10 \log(\sqrt{2} / 2 \cdot 10^{-5}) = 91$ dB.

The resulting Sound will appear in the List of Objects; its name will be equal to the file name, without extension.

Writing

With Write to AIFF file....

The samples of the Sound are multiplied by 32768 and quantized between -32768 and 32767.

To avoid clipping, keep the absolute amplitude below 1.000. If the maximum sound pressure level is 91 dB (top = 32767), the quantization threshold is (top = 1/2) -5 dB.

© ppgb, May 26, 1997

Formulas 1.7. Formulas for creation

With some commands in the New menu, you can supply a formula that Praat will apply to all elements of the new object.

Creating a Sound from a formula

Choose Create Sound... and type the following into the *Formula* field:

```
1/2 * sin (2 * pi * 377 * x)
```

When you click OK, a new Sound object will appear in the list. After you click *Edit* and zoom in a couple of times, you will see that the sound is a sine wave with a frequency of 377 Hertz (cycles per second). This worked because the x in the formula represents the time, i.e. the formula was applied to every sample separately, with a different value of x for each sample.

Creating a Matrix from a formula

Choose Create simple Matrix... and type the following into the *Formula* field:

```
8
```

When you click OK, a new Matrix object will appear in the list. When you click Info, you will see that it is a matrix with 10 rows and 10 columns, and that all the 100 cells contain the value 8 (you can see this because both the minimum and the maximum are reported as being 8).

A more interesting example is the formula

```
row * col
```

For the resulting Matrix, choose Paint cells... and click OK. The Picture window will show a 10×10 matrix whose elements are the product of the row and column numbers, i.e., they have values between 1 and 100. Beside *row* and *col*, you can use x for the distance along the horizontal axis and y for the distance along the vertical axis; see the following page for examples.

Links to this page

- [Formulas](#)
- [Formulas 1. My first formulas](#)

© ppgb, March 16, 2003

How to concatenate sound files

You can concatenate any combination of AIFF, AIFC, WAV, NeXT/Sun, and NIST audio files, and other files that you have read into memory.

For instance, if you want to concatenate a 30-minute AIFF file, a 4-minute Kay sound file, and a 60-minute Next/Sun file, by writing them into a 94-minute WAV file, you do the following:

1. Open the AIFF file with Open long sound file... from the Read menu. A LongSound object will appear in the list.
2. Read the Kay sound file into memory with Read from file.... A Sound object will appear in the list.
3. Open the AIFF file with Open long sound file... from the Read menu. A second LongSound object will appear in the list.
4. Select the three objects and choose Write to WAV file... from the Write menu.

This only works if all the sounds have the same sampling frequency and the same number of channels (a Sound object is always mono, of course).

Available formats

The format of the original sound files may be 16-bit linear (with big-endian or little-endian byte order), 8-bit linear (signed or unsigned), 8-bit μ -law, or 8-bit A-law. The format of the resulting sound file is always 16-bit linear, with an appropriate default byte order. The following commands are available in the Write menu of you select any combination of LongSound and/or Sound objects:

- Write to WAV file... (little-endian)
- Write to AIFF file... (big-endian)
- Write to AIFC file... (big-endian)
- Write to NeXT/Sun file... (big-endian)
- Write to NIST file... (little-endian)

Links to this page

- Sounds: Concatenate
-

© ppgb, January 23, 2000

Macintosh sound files

A way for storing a Sound object on disk.

File format

The double-clickable sound file of the Macintosh (8 bits per sample). Cannot be ported to other machines, because the sound is in the resource fork.

Reading

With Read from file....

To read a Sound from a Macintosh sound file on disk, use Read from file... (Macintosh only).

The 8-bit sample values are divided by 128 so that the amplitude of the resulting Sound is between -1.0 and +1.0.

The resulting Sound will appear in the list of objects; its name will be equal to the file name, without extension.

Writing

With **Write to Mac sound file...** Praat asks you for a file name. After you click OK, 0.5 is added to the samples of the Sound, they are multiplied by 128 and quantized between 0 and 255; the result is written to the file in 8-bit linear Macintosh sound-file format.

To avoid clipping, keep the absolute amplitude below 1.000. If the maximum sound pressure level is 91 dB (top = 127), the quantization threshold for a sine wave is (top = 1/2) 43 dB.

© ppgb, May 27, 1997

Manipulation: Extract original sound

A command to copy the original sound in each selected Manipulation object to a new Sound object.

© *ppgb*, March 30, 2001

Manipulation: Replace original sound

A command to replace the original sound in the selected Manipulation object with the selected Sound object.

© *ppgb*, March 30, 2001

NIST files

A way for storing a Sound object on disk.

File format

The compressed sound files of the Timit database, and the Groningen speech corpus.

Reading

With Read from file....

Writing

With Write to NIST audio file...

© *ppgb*, September 11, 1996

sampling frequency

The sampling frequency (or *sample rate*) is the number of samples per second in a Sound. For example: if the sampling frequency is 44100 Hertz, a recording with a duration of 60 seconds will contain 2,646,000 samples.

Usual values for the sampling frequency are 44100 Hz (CD quality) and 22050 Hz (just enough for speech, since speech does not contain relevant frequencies above 11025 Hz; see aliasing).

To get the sampling frequency of a selected **Sound**, click **Info** or choose Get sampling frequency.

Links to this page

- [sampling period](#)
-

© *ppgb*, April 15, 2004

sampling period

The sampling period is the time difference between two consecutive samples in a Sound. It is the inverse of the sampling frequency. For example: if the sampling frequency is 44100 Hz, the sampling period is $1/44100 = 2.2675736961451248e-05$ seconds: the samples are spaced approximately 23 microseconds apart.

To get the sampling period of a selected **Sound**, click **Info** or choose Get sampling period.

Links to this page

- [Get sampling frequency](#)
-

© *ppgb*, April 15, 2004

Sesam/LVS files

A way for storing a Sound object on disk.

File format

The sound files used by the SESAM and LVS programs. Each sample is normally quantized into 12 bits.

Reading

To read a **Sound** from a Sesam file on disk, use Read from file.... The file name is expected to end in ".sdf" or ".SDF".

The 12-bit sample values are divided by 2048 so that the amplitude of the resulting Sound is between -1.0 and +1.0.

The resulting **Sound** will appear in the List of Objects; its name will be equal to the file name, without extension.

If the sound was encoded in 16 bits per sample, you should divide by 16 after reading (with `Formula... self/16`)

Writing

With **Write to Sesam file...** Praat then asks you for a file name. After you click OK, the samples of the Sound are multiplied by 2048 and quantized between -2048 and 2047; the result is written to the file in 12-bit LVS and Sesam format.

To avoid clipping, keep the absolute amplitude below 1.000. If the maximum sound pressure level is 91 dB (top = 2047), the quantization threshold is (top = 1/2) 19 dB.

If you prefer 16-bit encoding, you should multiply by 16 before writing (with `Formula... self*16`)

© ppgb, September 11, 1996

Sound & Pitch: Change gender...

A command to create a new Sound object with manipulated characteristics from the selected Sound and Pitch.

With this command you can have finer grained control over the pitch than with the Sound: Change gender... command. Accurate pitch measurement determines the quality of the PSOLA synthesis.

Arguments

The settings are described in Sound: Change gender... (except that we don't need the *Maximum pitch* here, since it will be determined from the selected Pitch).

© djmw, February 8, 2003

Sound & Pitch: To FormantFilter...

A command that creates a FormantFilter object from the selected Sound and Pitch objects by band filtering in the frequency domain with a bank of filters whose bandwidths depend on the Pitch.

The filter functions used are:

$$H(f, F_0) = 1 / (((f_c^2 - f^2) / fB(F_0)))^2 + 1),$$

where f_c is the central (resonance) frequency of the filter. $B(F_0)$ is the bandwidth in Hz and determined as

$$B(F_0) = \text{relativeBandwidth} \cdot F_0,$$

where F_0 is the fundamental frequency as determined from the Pitch object. Whenever the value of F_0 is undefined, a value of 100 Hz is taken.

Links to this page

- [Sound: To FormantFilter...](#)
-

© djmw, April 4, 2001

Sound: Change gender...

A command to create a new Sound with manipulated characteristics.

Arguments

The quality of the manipulation depends on the pitch measurement.

The arguments that control the pitch measurement are:

Minimum pitch (default 75 Hz)

pitch candidates below this frequency will not be considered.

Maximum pitch (default 600 Hz)

pitch candidates above this frequency will be ignored.

The arguments that control the manipulation are:

Formant shift ratio

determines the frequencies of the formants in the newly created Sound. If this ratio equals 1 no frequency shift will occur and the formant frequencies will not change. A ratio of 1.2 will change a male voice to a voice with approximate female formant characteristics. A ratio of 1/1.2 will change a female voice to a voice with approximate male formant characteristics.

New pitch median (default 0.0 Hz: same as original)

determines what the median pitch of the new Sound will be. The pitch values in the newly created Sound will be calculated from the pitch values in the selected Sound by multiplying them by a factor $newPitchMedian / oldPitchMedian$. This factor equals 1.0 if the default value for the new pitch median (0.0) is chosen.

Pitch range factor (default 1.0)

determines an *extra* scaling of the new pitch values around the *new* pitch median. A factor of 1.0 means that no additional pitch modification will occur (except the obvious one described above). A factor of 0.0 monotonizes the new sound to the new pitch median.

Duration factor (default 1.0)

The factor with which the sound will be lengthened. The default is 1.0. If you take a value less than 1.0, the resulting sound will be shorter than the original. A value larger than 3.0 will not work.

If you want more control over the synthesis you can supply your own Pitch object and use the Sound & Pitch: Change gender... command.

Algorithm

The shifting of frequencies is done via manipulation of the sampling frequency. Pitch and duration changes are generated with PSOLA synthesis.

The new pitch values are calculated in a two step process. We first multiply all the pitches with the factor $newPitchMedian / oldPitchMedian$ according to:

$$newPitch = pitch * newPitchMedian / oldPitchMedian.$$

It follows that if the $newPitchMedian$ equals the $oldPitchMedian$ no change in pitch values will occur in the first step.

Subsequently, the pitch range scale factor determines the final pitch values in the following linear manner:

$$finalPitch = newPitchMedian + (newPitch - newPitchMedian) * pitchRangeScaleFactor$$

Hence, it follows that no further scaling occurs if $pitchRangeScaleFactor$ equals 1.0.

Links to this page

- [What's new?](#)

© djmw, February 5, 2003

Sound: To BarkFilter...

A command that creates a BarkFilter object from every selected Sound object by band filtering in the frequency domain with a bank of filters.

The filter functions used are:

$$10 \log \mathbf{H}(z) = 7 - 7.5 * (z_c - z - 0.215) - 17.5 * \sqrt{0.196 + (z_c - z - 0.215)^2}$$

where z_c is the central (resonance) frequency of the filter in Bark. The bandwidths of these filters are constant and equal 1 Bark.

© *djmw*, April 4, 2001

Sound: To Formant (keep all)...

A command that creates a Formant object from every selected Sound object. Not recommended for general use.

Purpose

to perform a short-term spectral analysis, approximating the spectrum of each frame by a number of formants.

Arguments

The same as with Sound: To Formant (burg)....

Algorithm

The same as with Sound: To Formant (burg).... In contrast with that command, however, all formant values are kept, even those below 50 Hz and those above *Maximum formant* minus 50 Hz. Although this makes the identification of the traditional F1 and F2 more difficult, this might give better results in resynthesis (see Sound & Formant: Filter), but it usually generates funny values instead.

Links to this page

- [Formants & LPC submenu](#)

© ppgb, February 10, 2000

Sound: To FormantFilter...

A command that creates a FormantFilter object from every selected Sound object by band filtering in the frequency domain with a bank of filters whose bandwidths depend on the pitch of the signal.

The analysis proceeds in two steps:

1. We perform a pitch analysis (see Sound: To Pitch... for details).
2. We perform a filter bank analysis on a linear frequency scale. The bandwidth of the filters depends on the measured pitch (see Sound & Pitch: To FormantFilter... for details).

© *djmw*, April 4, 2001

Sound: To MelFilter...

A command that creates a MelFilter object from every selected Sound object by band filtering in the frequency domain with a bank of filters.

The filter functions used are triangular in shape on a *linear* frequency scale. The filter function depends on three parameters, the lower frequency f_l , the central frequency f_c and the higher frequency f_h . On a *mel* scale, the distances $f_c - f_l$ and $f_h - f_c$ are the same for each filter and are equal to the distance between the f_c 's of successive filters. The filter function is:

$$H(f) = 0 \text{ for } f \leq f_l \text{ and } f \geq f_h$$

$$H(f) = (f - f_l) / (f_c - f_l) \text{ for } f_l \leq f \leq f_c$$

$$H(f) = (f_h - f) / (f_h - f_c) \text{ for } f_c \leq f \leq f_h$$

Links to this page

- [Sound: To MFCC...](#)

© djmw, April 4, 2001

Sound: To MFCC...

A command that creates a MFCC object from every selected Sound object.

The analysis proceeds in two steps:

1. We perform a filter bank analysis on a mel frequency scale (see Sound: To MelFilter... for details).
2. We convert the filter values to mel frequency cepstral coefficients (see MelFilter: To MFCC... for details).

Links to this page

- [Formants & LPC submenu](#)
-

© *djmw*, April 10, 2001

Sound: To Spectrum (dft)

A command to create a Spectrum object from every selected Sound object, by an over-all spectral analysis.

Algorithm

The algorithm is the continuous interpretation of the discrete Fourier Transform, with a negative exponent:

$$X(f) = \int_0^T x(t) e^{-2\pi ift} dt$$

If the Sound is expressed in Pascal (Pa), the Spectrum is expressed in Pa·s, or Pa/Hz.

Links to this page

- [What's new?](#)
-

© *ppgb*, May 5, 2004

SpellingChecker

One of the types of objects in PRAAT. For checking the spelling in texts and TextGrid objects.

1. How to create a SpellingChecker object

You normally read in a SpellingChecker with Read from file... from the Read menu.

2. How to check the spelling of a TextGrid

A SpellingChecker object can be used for purposes of spelling checking. In order to check the spellings in a TextGrid object, you first view the TextGrid in an editor window by selecting the TextGrid together with the SpellingChecker object, and clicking Edit. In most cases, you will also want to select a Sound or LongSound object before clicking Edit, so that a representation of the sound is also visible (and audible) in the editor. Thus, you typically select three objects and click Edit. The editor then allows you to check the spellings (command **Check spelling** from the Search menu).

3. How to create a SpellingChecker object for the first time

If you are the maintainer of a word list for spelling checking, you will want to convert this list to a SpellingChecker object that you can distribute among the transcribers of your corpus.

The first step is to create a WordList object from your text file, as described on the WordList man page. Then you simply click **To SpellingChecker**. A button labelled **Edit...** appears. This command allows you to set the following attributes of the SpellingChecker object:

Allow all parenthesized

this flag determines whether text between parentheses is ignored in spelling checking. This would allow the transcriber to mark utterances in foreign languages, which cannot be found in the lexicon.

Separating characters

determines the set of characters (apart from the space character) that separate words. The standard is ".,:()". If a string like "error-prone" should be considered two separate words, you will like to change this to ".,:()-". For the Corpus of Spoken Dutch (CGN), the hyphen is not a separator, since words like "mee-eter" should be checked as a whole. If a string like "Mary's" should be considered two separate words, include the apostrophe.

Allow all names

determines whether all words that start with a capital are allowed. For the CGN, this is on, since the lexicon does not contain many names.

Name prefixes

a space-separated list that determines what small groups of characters can precede names. For the CGN, this is "'s- d' l'", since names like 's-Gravenhage, d'Ancona, and l'Hôpital should be ignored by the spelling checker.

Allow all words containing

a space-separated list of strings that make a word correct even if not in the lexicon. For the CGN, this is "* xxx", since words like *keuje*d* and *verxxxing* should be ignored by the spelling checker.

Allow all words starting with

a space-separated list of prefixes that make a word correct even if not in the lexicon. For the CGN, this is empty.

Allow all words ending in

a space-separated list of suffixes that make a word correct even if not in the lexicon. For the CGN, this is "-", since the first word in *verzekerings- en bankwezen* should be ignored by the spelling checker.

Links to this page

- [TextGridEditor](#)
- [What's new?](#)

© ppgb, September 16, 2003

TextTier

One of the types of objects in PRAAT.

A TextTier object represents a marked point process, i.e., it contains a series of (*time*, *text*) points, sorted by time.

Each point is marked with a text string. Though this is a system-independent ASCII text string, it may, as everywhere in Praat, contain Special symbols like Greek letters, mathematical symbols, style information, and Phonetic symbols, which will be visible when the string is drawn into the Picture window or viewed in a TextGridEditor.

Creating a TextTier object in Praat

A TextTier object is usually extracted from a TextGrid object, which may contain several tiers.

From scratch:

1. Select an object with a time domain, e.g., a Sound or a Pitch.
2. Click "To TextTier".

The resulting TextTier will have the same time domain as the original Sound (or Pitch), e.g., if the Sound object starts at 0.0 seconds and has a duration of 2.1 seconds, the resulting TextTier will also have an *xmin* of 0.0 and an *xmax* of 2.1 seconds.

The resulting TextTier will have 0 points in it.

Editing a TextTier in Praat

To edit a TextTier, first convert it into a TextGrid. You invoke a TextGridEditor in either of two ways:

- Select a TextGrid and click **Edit**.
- Select a TextGrid and a Sound and click **Edit**.

In the latter case, a copy of the Sound object will be visible in the editor.

The TextGridEditor will allow you to add marks to a TextTier object, move marks around, edit the texts in the marks, and remove marks.

Drawing a TextTier

You can draw a TextTier together with a Pitch object, by selecting one TextTier and one Pitch object, and choosing "Draw...". The mark texts will appear just above the pitch contour.

Class description

If you select a TextTier and click **Inspect** or **Write to console**, you will see that a TextTier object contains the following attributes:

x_{min}

the starting time.

x_{max}

the end of the time domain.

points

a collection of text points, sorted by their times.

Links to this page

- [Get high index from time...](#)
- [Get low index from time...](#)
- [Get nearest index from time...](#)
- [Intensity & TextTier: To IntensityTier...](#)
- [IntervalTier\(s\): To TextGrid](#)
- [Pitch & TextTier: To PitchTier...](#)
- [PointProcess: Up to TextTier...](#)
- [Remove point near...](#)
- [Remove point...](#)
- [Remove points between...](#)
- [TextGrid & TextTier: Append](#)
- [TextTier\(s\) & IntervalTier\(s\): To TextGrid](#)
- [TextTier\(s\): To TextGrid](#)
- [TextTier: Add point...](#)
- [TextTier: Down to PointProcess](#)

© ppgb, March 16, 2003

Editors

Many types of objects in PRAAT can be viewed and edited in their own windows.

Editors

- SoundEditor
- LongSoundEditor
- TextGridEditor
- ManipulationEditor
- SpectrumEditor
- PitchEditor
- PointEditor
- PitchTierEditor
- IntensityTierEditor
- DurationTierEditor
- **SpectrogramEditor**
- **ArtwordEditor**
- OTGrammarEditor
- OTAnyGrammarEditor
- **DataEditor** (see Inspect)

How to raise an editor

To bring up a class-specific editor window, select the appropriate object and choose **Edit** (if the **Edit** button exists, it is usually at the top of the Dynamic menu). The name of the object will appear as the title of the editor window.

Some objects may have different commands, like **View** (read-only) or **Surf** (hypertext-based).

To bring up a DataEditor, select one object and click the Inspect button.

General behaviour

Changes that you make to an object in its editor window will take effect immediately (you do not have to close the editor window before going on with the changed object).

If you Remove an object that you are viewing or editing from the List of Objects, the editor window will automatically disappear from the screen.

All editors are independent windows: you can minimize and maximize them; if an editor window goes hiding behind another window, you can raise it by choosing the **Edit** command again.

If you rename an object that you are viewing or editing (with Rename..., the title of the editor window will be changed to the new name.

Ways to control an editor

- Click
- Shift-click
- Drag
- Shift-drag
- Time selection
- Keyboard shortcuts

Links to this page

- Edit
 - File menu
 - Query
 - Query menu
 - View
-

© *ppgb*, March 16, 2003

Formant analysis...

The analysis parameters for formant contours in the SoundEditor, LongSoundEditor, and TextGridEditor windows. These settings will be remembered across Praat sessions.

For the meaning of these settings, see Sound: To Formant (burg).... The *number of poles* setting determines the number of poles in the linear predictive analysis. The standard value is 10, which means that 5 formants will be detected.

© ppgb, September 16, 2003

Log files

With some commands in the Query menu of the SoundEditor and TextGridEditor, you can write combined information about times, pitch values, formants, and intensities to the Info window and to a log file.

A log file is a text file on disk. It consists of a number of similar lines, whose format you determine with the log settings in the Query menu.

Every time you press F12 (or choose **Log 1** from the Query menu, Praat writes a line to log file 1. If you press Shift-F12, Praat writes a line to log file 2.

With the **log settings** dialog, you determine the following:

Log 1 to Info window

this determines whether your log line will be written to the Info window or not.

Log 1 to log file

this determines whether your log line will be written to the log file or not.

Log file 1

the name of the log file. On Windows, this has to be a complete path name, such as C:\WINDOWS\DESKTOP\Pitch Log.txt. On Unix and MacOS X, it can either be a complete path name, e.g. /home/mary/pitch_log, or a home-relative name such as ~/Desktop/Pitch log.

Log 1 format

the format of the line that Praat will write. See below.

The same goes for log file 2.

Usage

The logging facility has been implemented in Praat especially for former users of Kay CSL, who have been used to doing it for years and like to continue doing it in Praat. Otherwise, you may prefer to use the TextGridEditor to mark time points and run an automatic analysis afterwards.

If you do want to use the logging facility, you typically start by deleting any old log file (by choosing **Delete log file 1** or **Delete log file 2**), if you want to re-use the file name. Otherwise, you can change the log file name (with **Log settings...**). After this, you will move the cursor to various time locations and press F12 (or Shift-F12) each time, so that information about the current time will be written to the log file.

Example 1: pitch logging

Suppose you want to log the time of the cursor and the pitch value at the cursor. You could use the following log format:

Time 'time:6' seconds, pitch 'f0:2' Hertz

If you now click at 3.456789876 seconds, and the pitch happens to be 355.266 Hertz at that time, the following line will be appended to the log file and/or to the Info window:

Time 3.456790 seconds, pitch 355.27 Hertz.

The parts ":6" and ":2" denote the number of digits after the decimal point. If you leave them out, the values will be written with a precision of 17 digits.

The words 'time' and 'f0' mean exactly the same as the result of the commands **Get cursor** and **Get pitch**. Therefore, if instead of setting a cursor line you selected a larger piece of the sound, 'time' will give the centre of the selection and 'f0' will give the mean pitch in the selection.

Beware of the following pitfall: if your pitch units are not Hertz, but semitones, then 'f0' will give the result in semitones. A format as in this example will then be misleading.

Example 2: formant logging

Suppose you want to log the start and finish of the selection, its duration, and the mean values of the first three formants, all separated by tab stops for easy importation into Microsoft® Excel™. You could use the following log format:

```
't1:4''tab$''t2:4''tab$''f1:0''tab$''f2:0''tab$''f3:0'
```

You see that 't1' and 't2' are the start and finish of the selection, respectively, and that they are written with 4 digits after the decimal point. By using ":0", the three formant values are rounded to whole numbers in Hertz. The word 'tab\$' is the tab stop.

Loggable values

The following values can be logged:

- 'time': the time of the cursor, or the centre of the selection.
- 't1': the start of the selection ("B").
- 't2': the end of the selection ("E").
- 'dur': the duration of the selection.
- 'f0': the pitch at the cursor time, or the mean pitch in the selection.
- 'f1', 'f2', 'f3', 'f4', 'f5': the first/second/third/fourth/fifth formant at the cursor time, or the mean first/second/third/fourth/fifth formant in the selection.
- 'b1', 'b2', 'b3', 'b4', 'b5': the bandwidth of the first/second/third/fourth/fifth formant at the cursor time or at the centre of the selection.
- 'intensity': the intensity at the cursor time, or the mean intensity in the selection.
- 'tab\$': the tab stop.

More flexibility in logging

You may sometimes require information in your log file that cannot be generated directly by the loggable values above. Suppose, for instance, that you want to log the values for F1 and F2-F1 at the points where you click. You could write the following script:

```
f1 = Get first formant
f2 = Get second formant
f21 = f2 - f1
printline 'f1:0' 'f21:0'
fileappend "D:\Praat logs\Formant log.txt"
'f1:0''tab$''f21:0''newline$'
```

With this script, the information would be appended both to the Info window and to the file "Formant log.txt" on your desktop.

You can make this script accessible with Option-F12 (or Command-F12) by saving the script and specifying the name of the script file in the **Log script 3** (or **4**) field in the **Log settings...** dialog.

These scripts may take arguments. Suppose, for instance, that you want to specify a vowel symbol as you press Option-F12. The following script will take care of that:

```
form Save vowel and formants
  word Vowel a
endform
f1 = Get first formant
f2 = Get second formant
f21 = f2 - f1
printline 'vowel$' 'f1:0' 'f21:0'
fileappend "~/Praat logs/Vowels and formants log"
'vowel$''f1:0''tab$''f21:0''newline$'
```

Beware of the following pitfall: because of the nature of scripts, you should not try to do this when you have two editor windows with the same name. I cannot predict which of the two windows will answer the **Get** queries...

Links to this page

- [What's new?](#)

© ppgb, January 16, 2003

Time step settings...

A command in the **View** menu of the SoundEditor and TextGridEditor to determine the time interval between consecutive measurements of pitch, formants, and intensity.

Automatic time steps

It is recommended that you set the *Time step strategy* to **Automatic**. In this way, Praat computes just enough pitch, formant, and intensity values to draw reliable pitch, formant, and intensity contours. In general, Praat will compute 4 values within an analysis window ("four times oversampling").

As described in Sound: To Pitch..., Praat's standard time step for pitch analysis is 0.75 divided by the pitch floor, e.g., if the pitch floor is 75 Hz, the time step will be 0.01 seconds. In this way, there will be 4 pitch measurements within an analysis window, which is $3 / (75 \text{ Hz}) = 40$ milliseconds long.

As described in Sound: To Formant (burg)..., Praat's standard time step for formant measurements is the *Window length* divided by 4, e.g. if the window length is 0.025 seconds, the time step will be 6.25 milliseconds.

As described in Sound: To Intensity..., Praat's standard time step for intensity measurements is 0.8 divided by the pitch floor, e.g. if the pitch floor is 75 Hz, the time step will be 10.6666667 milliseconds. In this way, there will be 4 intensity measurements within an intensity analysis window, which is $3.2 / (75 \text{ Hz}) = 42.6666667$ milliseconds long.

Fixed time step

You can override the automatic time step by setting the *Time step strategy* to **Fixed**. The *Fixed time step* setting then determines the time step that Praat will use: if you set it to 0.001 seconds, Praat will compute pitch, formant, and intensity values for every millisecond. Beware that this can slow down the editor window appreciably, because this step is much smaller than usual values of the automatic time step (see above).

Enlarging the time step to e.g. 0.1 seconds will speed up the editor window but may render the pitch, formant, and intensity curves less exact (they become *undersampled*), which will influence your measurements and the locations of the pulses.

If there are fewer than 2.0 pitch measurement points per analysis window, Praat will draw the pitch curve as separate little blue disks rather than as a continuous blue curve, in order to warn you of the undersampling. E.g. if the pitch floor is 75 Hz, Praat will draw the pitch curve as disks if the time step is greater than 0.02 seconds.

View-dependent time step

Another way to override the standard time step is by setting the *Time step strategy* to **View-dependent**. The *Number of time steps per view* setting then determines the time step that Praat will use: if you set it to 100, Praat will always compute 100 pitch, formant, and intensity values within the view window. More precisely: if you zoom the view window to 3 seconds, Praat will show you 100 pitch, formant, and intensity points at distances of 0.03 seconds (or fewer than 100, if you are near the left or right edge of the signal). As with the *Fixed time step* setting, Praat will draw the pitch as separate disks in case of undersampling. You may want to use this setting if you want the pitch curve to be drawn equally fast independently of the degree of zooming.

Links to this page

- [What's new?](#)
-

© ppgb, October 3, 2003

starting time

- the beginning of the time domain (see there).

© *ppgb*, April 20, 2004

finishing time

- the end of the time domain (see there).

© *ppgb*, April 20, 2004

total duration

- the extent of the time domain (see there).

© *ppgb*, May 5, 2004

Polygon

One of the types of objects in PRAAT.

A Polygon object represents a sequence of points (x_i, y_i) in a two-dimensional space.

© *ppgb, March 16, 2003*

Strings

One of the types of objects in PRAAT. Represents an ordered list of strings.

Creation

The difficult way is to create a **Strings** object from a generic Praat text file:

```
File type = "ooTextFile"  
Object class = "Strings"  
5 (number of strings)  
"Hello"  
"Goodbye"  
"Auf wiedersehen"  
"Tsch\u" "ss"  
"Arrivederci"
```

In this example, we see that a double quote within a string should be written twice; the fourth string will therefore be read as **Tsch\u"ss**, and will be shown in info messages or in graphical text as **Tschüss** (see special symbols). This file can be read simply with the generic Read from file... command from the Read menu.

An easier way is to use the special command Read Strings from raw text file.... The file can then simply look like this:

```
Hello  
Goodbye  
Auf wiedersehen  
Tsch\u"ss  
Arrivederci
```

In this example, all the strings are in the generic system-independent ASCII format that is used everywhere in Praat (messages, graphical text) for special symbols. You could also have supplied the strings in a native format, which is ISO-Latin1 encoding on Unix and Windows computers, or Mac encoding on Macintosh computers. The file would then have simply looked like:

```
Hello  
Goodbye  
Auf wiedersehen  
Tschüss  
Arrivederci
```

To convert this into the generic system-independent ASCII format, use the **Genericize** command.

You can also create a **Strings** object from a directory listing or from some other objects:

- [Create Strings as file list...](#)
- [Distributions: To Strings...](#)
- [OTGrammar: Generate inputs...](#)
- [OTGrammar & Strings: Inputs to outputs...](#)
- [OTAnyGrammar: Generate inputs...](#)
- [OTAnyGrammar & Strings: Inputs to outputs...](#)

Links to this page

- [OT learning 2.9. Output distributions](#)
- [OT learning 3.1. Data from a pair distribution](#)
- [OT learning 3.2. Data from another grammar](#)
- [OT learning 4. Learning an ordinal grammar](#)
- [OT learning 5. Learning a stochastic grammar](#)
- [OT learning 7. Learning from overt forms](#)
- [OTAnyGrammar examples](#)
- [PairDistribution: To Stringses...](#)
- [Strings: To Distributions](#)
- [WordList](#)

© *ppgb*, March 16, 2003

Distributions

One of the types of objects in PRAAT. Inherits most actions from TableOfReal.

Actions

Distributions: To Strings...

Links to this page

- [OT learning 2.9. Output distributions](#)
- [OT learning 7. Learning from overt forms](#)
- [OTAnyGrammar examples](#)
- [OTGrammar: Input to outputs...](#)
- [OTGrammar: To output Distributions...](#)
- [Strings: To Distributions](#)

© *ppgb*, March 16, 2003

PairDistribution

One of the types of objects in PRAAT. A PairDistribution object represents the relative probabilities with which the specified pairs of strings occur.

Class description

struct-list pairs

a list of relative string-pair probabilities. Each element consists of:

string *string1*

the first string.

string *string2*

the second string.

real *weight*

the relative probability associated with the string pair. This value cannot be negative.

Links to this page

- [OT learning 3.1. Data from a pair distribution](#)
- [OT learning 3.2. Data from another grammar](#)
- [OT learning 6. Shortcut to OT learning](#)
- [PairDistribution: To Stringses...](#)

© *ppgb*, March 16, 2003

Sequence

One of the types of objects in PRAAT.

An object of type Sequence holds a particular ordering of integer numbers.

© *djmw*, May 9, 1997

ParamCurve

One of the types of objects in PRAAT.

An object of class **ParamCurve** represents a sequence of time-stamped points $(x(t_i), y(t_i))$ in a two-dimensional space.

© ppgb, March 16, 2003

Ltas

One of the types of objects in PRAAT. **Ltas** is short for Long-Term Average Spectrum.

An object of class Ltas represents the power spectrum as a function of frequency, expressed in dB (in air).

Inside an Ltas object

With Inspect, you will see the following attributes:

x_{min}

the bottom of the frequency domain, in Hertz. Usually 0.

x_{max}

the top of the frequency domain, in Hertz.

n_x

the number of frequency bands (≥ 1).

dx

the frequency step, or *bin width*, in Hertz.

x_1

the frequency associated with the first bin, in Hertz. Usually equals $dx / 2$, because the first bin tends to start at 0 Hertz.

$z_{1i}, i = 1 \dots n_x$

the power spectrum, expressed in dB.

Links to this page

- [Ltas: Get band from frequency...](#)
- [Ltas: Get band width](#)
- [Ltas: Get frequency from band...](#)
- [Ltas: Get frequency of maximum...](#)
- [Ltas: Get frequency of minimum...](#)
- [Ltas: Get frequency range](#)
- [Ltas: Get highest frequency](#)
- [Ltas: Get lowest frequency](#)
- [Ltas: Get maximum...](#)
- [Ltas: Get mean...](#)
- [Ltas: Get minimum...](#)
- [Ltas: Get number of bands](#)
- [Ltas: Get standard deviation...](#)
- [Ltas: Get value at frequency...](#)
- [Ltas: Get value in band...](#)
- [Spectrum: To Ltas \(1-to-1\)](#)
- [What's new?](#)

© *ppgb*, March 16, 2003

BarkFilter

One of the types of objects in PRAAT.

An object of type BarkFilter represents an acoustic time-frequency representation of a sound: the power spectral density $P(z, t)$, expressed in dB's. It is sampled into a number of points around equally spaced times t_i and frequencies z_j (on a Bark scale).

Inside a BarkFilter

With Inspect you will see that this type contains the same attributes a Matrix.

Links to this page

- [Sound: To BarkFilter...](#)
- [What's new?](#)

© *djmw*, April 4, 2001

MelFilter

One of the types of objects in PRAAT.

An object of type MelFilter represents an acoustic time-frequency representation of a sound: the power spectral density $P(f, t)$, expressed in dB's. It is sampled into a number of points around equally spaced times t_i and frequencies f_j (on a Mel frequency scale).

Inside a MelFilter

With Inspect you will see that this type contains the same attributes a Matrix.

Links to this page

- [MelFilter: To MFCC...](#)
- [MFCC: To MelFilter...](#)
- [Sound: To MelFilter...](#)
- [What's new?](#)

© *djmw*, April 4, 2001

FormantFilter

One of the types of objects in PRAAT.

An object of type FormantFilter represents an acoustic time-frequency representation of a sound: the power spectral density $P(f, t)$, expressed in dB's. It is sampled into a number of points around equally spaced times t_i and frequencies f_j (on a linear frequency scale).

Inside a FormantFilter

With Inspect you will see that this type contains the same attributes a Matrix.

Links to this page

- [Sound & Pitch: To FormantFilter...](#)
- [Sound: To FormantFilter...](#)
- [What's new?](#)

© *djmw*, April 4, 2001

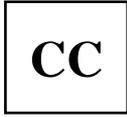
Cepstrum

One of the types of objects in PRAAT.

Description

An object of type Cepstrum represents the complex cepstrum.

© *djmw*, February 19, 2001



One of the types of objects in PRAAT.

Any object that represents cepstral coefficients as a function of time.

Links to this page

- [CC: To Matrix](#)

© *djmw*, February 19, 2001

LFCC

One of the types of objects in PRAAT.

An object of type LFCC represents cepstral coefficients on a linear frequency scale as a function of time. The coefficients are represented in frames with constant sampling period.

LFCC commands

Creation:

- [LPC: To LFCC...](#)

Links to this page

- [CC: To DTW...](#)
- [LFCC: To LPC...](#)
- [What's new?](#)

© *djmw*, April 21, 2004

MFCC

One of the types of objects in PRAAT.

An object of type MFCC represents mel frequency cepstral coefficients as a function of time. The coefficients are represented in frames with constant sampling period.

MFCC commands

Creation:

- Sound: To MFCC...
- MelFilter: To MFCC...

Links to this page

- CC: To DTW...
 - MFCC: To MelFilter...
 - What's new?
-

© *djmw*, April 11, 2001

Excitations

A collection of objects of type Excitation. You can create an **Excitations** by selecting one or more **Excitation**'s and selecting **To Excitations**. You can add one or more **Excitation**'s to an **Excitations** by selecting one **Excitations** and one or more **Excitation**'s and selecting **Add to Excitations** (the **Excitation**'s will be removed from the list of objects).

Links to this page

- [Excitations: Append](#)
- [Excitations: To Pattern...](#)
- [Types of objects](#)

© *djmw*, September 18, 1996

IntervalTier

One of the types of objects in PRAAT. An IntervalTier represents a series of contiguous intervals in time. Each interval contains a a string.

Creating a IntervalTier object in Praat

An IntervalTier object is normally extracted from a TextGrid object, which may contain several tiers.

From scratch:

1. Select an object with a time domain, e.g., a Sound or a Pitch.
2. Click **To IntervalTier**.

The resulting IntervalTier will have the same time domain as the original Sound (or Pitch), e.g., if the Sound object starts at 0.0 seconds and has a duration of 2.1 seconds, the resulting IntervalTier will also have an *xmin* of 0.0 and an *xmax* of 2.1 seconds.

The resulting IntervalTier will have one interval in it, extending from *xmin* to *xmax*.

Editing an IntervalTier in Praat

You would edit an IntervalTier by converting it to a TextGrid and invoking a TextGridEditor, perhaps with a Sound.

Drawing an IntervalTier

You can draw an IntervalTier alone or together with a Sound or Pitch object, after converting it to a TextGrid.

Inside an IntervalTier

With Inspect, you will see the following attributes:

x_{min}

the starting time, in seconds.

x_{max}

the end of the time domain, in seconds.

intervals

a sorted collection of text intervals.

The attributes of a text interval are:

x_{min}

the starting time, in seconds.

x_{max}

the end of the time domain, in seconds.

text

an ASCII text string.

Though *text* is a system-independent ASCII text string, it may, as everywhere in Praat, contain Special symbols like Greek letters, mathematical symbols, style information, and Phonetic symbols, which will be visible when the string is drawn into the Picture window or viewed in a TextGridEditor.

Links to this page

- [IntervalTier\(s\): To TextGrid](#)
- [TextGrid & IntervalTier: Append](#)
- [TextTier\(s\) & IntervalTier\(s\): To TextGrid](#)
- [TextTier\(s\): To TextGrid](#)

© *ppgb*, March 16, 2003

VocalTract

One of the types of objects in PRAAT.

A VocalTract object represents the area function of the vocal tract, expressed in m^2 , running from the glottis to the lips.

Links to this page

- [Create Vocal Tract from phone...](#)
- [LPC: To VocalTract \(slice\)...](#)
- [VocalTract: Formula...](#)

© *ppgb*, March 16, 2003

FFNet

One of the types of objects in PRAAT.

A **FFNet** models a feedforward neural net. A feedforward neural net can *learn* associations between its *input* and its *output*. The Feedforward neural networks tutorial gives you an introduction to feedforward neural nets.

Links to this page

- [Create FFNet \(linear outputs\)...](#)
- [Create FFNet...](#)
- [Create iris example...](#)
- [epoch](#)
- [Feedforward neural networks 2. Quick start](#)
- [FFNet & Pattern & Categories: Learn slow...](#)
- [FFNet & Pattern & Categories: Learn...](#)
- [FFNet & Pattern: To Categories...](#)
- [FFNet: Draw cost history...](#)
- [FFNet: Draw topology](#)
- [FFNet: Draw weights...](#)
- [FFNet: Extract weights...](#)
- [FFNet: Get number of hidden units...](#)
- [FFNet: Get number of hidden weights...](#)
- [FFNet: Get number of inputs](#)
- [FFNet: Get number of outputs](#)
- [FFNet: Principal components](#)
- [FFNet: Reset...](#)
- [FFNet: Select biases...](#)
- [Pattern & Categories: To FFNet...](#)

© *djmw*, October 15, 1996

Pattern

One of the types of objects in PRAAT.

An object of type Pattern represents a sequence of patterns that can serve as as input patterns for a neural net.

Pattern commands

Creation:

- **Create Pattern with zeroes...**

Synthesis:

- FFNet & Pattern: To Categories...
- Pattern & Categories: To FFNet...

inside a Pattern

With Inspect you will see that this type contains the same attributes a Matrix.

Links to this page

- Create iris example...
- Discriminant & Pattern: To Categories...
- Excitations: To Pattern...
- Feedforward neural networks 2. Quick start
- FFNet & Pattern & Categories: Learn slow...
- FFNet & Pattern & Categories: Learn...
- FFNet: Activation
- FFNet: Categories
- FFNet: Pattern
- TableOfReal: To Pattern and Categories...

© djmw, September 18, 1996

Categories

One of the types of objects in PRAAT.

An object of type `Categories` represents an ordered collection of categories. Each category is a simple text string.

Categories commands

Creation:

- **Create an empty Categories**
- FFNet & Pattern: To Categories...

Viewing and editing:

- CategoriesEditor

Analysis:

- Categories: To Confusion
- Categories: Difference

Synthesis

- Categories: Append
- **Categories: Permute...**
- **Categories: To unique Categories**

Inside a Categories

With `Inspect` you will see the following attributes:

size

the number of simple categories.

item[]

the categories. Each category is an object of type **SimpleString**.

Links to this page

- Create iris example...
- Feedforward neural networks 2. Quick start
- Feedforward neural networks 3. FFNet versus discriminant classifier
- FFNet & Pattern & Categories: Learn slow...
- FFNet & Pattern & Categories: Learn...

- FFNet: Categories
 - Pattern & Categories: To FFNet...
 - TableOfReal: To Pattern and Categories...
-

© *djmw*, September 18, 1996

Eigen

One of the types of objects in PRAAT.

An object of type Eigen represents the eigen structure of a matrix whose eigenvalues and eigenvectors are real.

Inside an Eigen

With Inspect you will see the following attributes:

numberOfEigenvalues

the number of eigenvalues and eigenvectors

dimension

the dimension of an eigenvector.

eigenvalues[1..numberOfEigenvalues]

the real eigenvalues.

eigenvectors[1..numberOfEigenvalues][1..dimension]

the real eigenvectors, stored by row.

Links to this page

- [Eigen & Matrix: Project...](#)
- [Eigen & SSCP: Project](#)
- [Eigen & TableOfReal: Project...](#)
- [Eigen: Draw eigenvalues...](#)
- [Eigen: Draw eigenvector...](#)
- [Eigen: Get contribution of component...](#)
- [Eigen: Get cumulative contribution of components...](#)
- [Eigen: Get eigenvalue...](#)
- [Eigen: Get eigenvector element...](#)
- [FFNet: Principal components](#)
- [PCA](#)

© djmw, November 2, 1998

Polynomial

One of the types of objects in PRAAT.

An object of type Polynomial represents a polynomial function on a domain.

A polynomial of degree n is defined as:

$$p(x) = c_1 + c_2 x + c_3 x^2 + \dots + c_{n+1} x^n.$$

The real numbers c_k are called the polynomial *coefficients*.

Commands

Creation

- Create Polynomial... (in the **New menu**)
- LPC: To Polynomial (slice)... (from prediction coefficients)
- LegendreSeries: To Polynomial
- ChebyshevSeries: To Polynomial

Drawing

- **Draw...**

Queries

- Get function value...: get $p(x)$
- **Get coefficient value...: get c_i**
- Get minimum...: minimum of $p(x)$ on an interval
- Get x of minimum...
- Get maximum...: maximum of $p(x)$ on an interval
- Get x of maximum...
- Get area...

Modification

- **Set domain...: new domain**
- **Set coefficient value...: change one coefficient**

Conversion

- To Spectrum... (evaluation over unit-circle)
- To Polynomial (derivative)
- To Polynomial (primitive)
- To Roots: roots of polynomial

Links to this page

- Polynomial: Scale x...
 - Polynomials: Multiply
 - Roots
 - What's new?
-

© *djmw*, June 8, 1999

Roots

One of the types of objects in PRAAT.

An object of type Roots represents the (complex) roots of a polynomial function.

Commands

Creation

- Polynomial: To Roots

Drawing

- **Draw...** (in the complex plane)

Queries

- **Get root...**: get complex root
- **Get real part of root...**
- **Get imaginary part of root...**

Links to this page

- LPC: Draw poles...
- What's new?

© djmw, June 8, 1999

ChebyshevSeries

One of the types of objects in PRAAT.

An object of type ChebyshevSeries represents a linear combination of Chebyshev polynomials $T_k(x)$.

$$\text{ChebyshevSeries}(x) = \sum_{k=1..numberOfCoefficients} c_k T_k(x)$$

Links to this page

- [ChebyshevSeries: To Polynomial](#)
- [Create ChebyshevSeries...](#)

© *djmw*, June 20, 1999

LegendreSeries

One of the types of objects in PRAAT.

An object of type LegendreSeries represents a linear combination of Legendre polynomials $P_k(x)$.

$$\text{LegendreSeries}(x) = \sum_{k=1..numberOfCoefficients} c_k P_k(x)$$

Links to this page

- [Create LegendreSeries...](#)
 - [LegendreSeries: To Polynomial](#)
-

© *djmw*, June 20, 1999

ISpline

One of the types of objects in PRAAT.

An object of type ISpline represents a linear combination of basis ispline functions. Each basis *ispline* is a monotonically increasing polynomial function of degree p .

$$\text{ISpline}(x) = \sum_{k=1..numberOfCoefficients} c_k \text{ispline}_k(x)$$

Links to this page

- [Create ISpline...](#)
-

© *djmw*, June 27, 1999

MSpline

One of the types of objects in PRAAT.

An object of type MSpline represents a linear combination of basis mspline functions. Each basis *mspline* is a positive polynomial function of degree p .

$$\text{MSpline}(x) = \sum_{k=1..numberOfCoefficients} c_k \text{mspline}_k(x)$$

Links to this page

- [Create MSpline...](#)
-

© djmw, June 27, 1999

ClassificationTable

One of the types of objects in PRAAT.

An object of type ClassificationTable represents the result of a classification experiment. The numbers in a row show how well a particular input matches the classes represented by the column labels. The higher the number the better the match.

Links to this page

- [Discriminant & TableOfReal: To ClassificationTable...](#)
-

© *djmw*, May 25, 1999

DTW

One of the types of objects in PRAAT.

An object of type DTW represents the dynamic time warp structure of two objects.

Commands

Creation:

- CC: To DTW... (from 2 objects with cepstral coefficients)
- **Spectrogram: To DTW...** (from 2 Spectrogram objects)

Query:

- DTW: Get time along path...

Links to this page

- [What's new?](#)
-

© *djmw*, May 25, 2000

Similarity

One of the types of objects in PRAAT.

An object of type Similarity represent a one-way table of similarities between "objects".

Commands

Creation

- Confusion: To Similarity...

Drawing

- **Draw as numbers...**
- **Draw as squares...**

Query

- **Get column mean (index)...**
- **Get column mean (label)...**
- **Get column stdev (index)...**
- **Get column stdev (label)...**

Modification

- **Formula...**
- **Remove column (index)...**
- **Insert column (index)...**
- **Set row label (index)...**
- **Set row label (label)...**
- **Set column label (index)...**
- **Set column label (label)...**

Analysis

- Similarity: To Dissimilarity...

Links to this page

- Configuration: To Similarity (cc)
-

© djmw, October 8, 1996

ScalarProduct

One of the types of objects in PRAAT.

An object of type ScalarProduct represents scalar products b_{ij} between objects i and j in a metrical space.

$$b_{ij} = \sum_{k=1..numberOfDimensions} x_{ik}x_{jk},$$

where x_{ik} and x_{jk} are the coordinates of the k -th dimension of points i and j , respectively. From this definition one can see that scalar products, in contrast to distances, do change when the origin is shifted.

Creation

- Distance: To ScalarProduct...
-

© djmw, January 25, 1998

Correspondence analysis

Correspondence analysis provides a method for representing data in an Euclidean space so that the results can be visually examined for structure. For data in a typical two-way ContingencyTable both the row variables and the column variables are represented in the same space. This means that one can examine relations not only among row or column variables but also between row and column variables.

In correspondence analysis the data matrix is first transformed by dividing each cell by the square root of the corresponding row and column totals. The transformed matrix is then decomposed with singular value decomposition resulting in the singular values (which in this case are canonical correlations) and a set of row vectors and column vectors. Next the row and column vectors are rescaled with the original total frequencies to obtain optimal scores. These optimal scores are weighted by the square root of the singular values and become the coordinates of the points in the Configuration.

Examples can be found in the books by Weller & Romney (1990) and Gifi (1990).

Links to this page

- [ContingencyTable: To Configuration \(ca\)...](#)
- [Types of objects](#)

© *djmw*, December 16, 1997

ContingencyTable

One of the types of objects in PRAAT.

In a two-way contingency table, cell f_{ij} contains the frequency with which row category i co-occurs with column category j . Necessarily, all $f_{ij} \geq 0$.

Commands

Creation

- **TableOfReal: To ContingencyTable**

Query

- **ContingencyTable: Get chi squared probability**
- **ContingencyTable: Get Cramer's statistic**
- **ContingencyTable: Get contingency coefficient**

Analysis

- **ContingencyTable: To Configuration (ca)...**

Links to this page

- [Correspondence analysis](#)
-

© djmw, December 16, 1997

WordList

One of the types of objects in PRAAT. An object of class WordList contains a sorted list of strings in a system-independent format. WordList objects can be used for spelling checking after conversion to a SpellingChecker object.

1. How to create a WordList object

You will normally create a WordList object by reading a binary WordList file. You'll use the generic Read from file... command from the Read menu.

See below under 3 for how to create such a file.

2. What you can do with a Wordlist object

The main functionality of a WordList is its ability to tell you whether it contains a certain string. If you select a WordList, you can query the existence of a specific word by using the **Has word** command. You supply the word and press OK. If the WordList does contain the word, the value "1" will be written to the Info window; otherwise, the value "0" will be written.

3. How to create a binary WordList file

You can create a binary (compressed) WordList file from a simple text file that contains a long list of words. Perhaps such a text file has been supplied by a lexicographic institution in your country; because of copyright issues, such word lists cannot be distributed with the Praat program. To convert the simple text file into a compressed WordList file, you basically take the following steps:

```
Read Strings from raw text file... lexicon.iso
Genericize
Sort
To WordList
Write to binary file... lexicon.WordList
```

I'll explain these steps in detail. For instance, a simple text file may contain the following list of words:

```
cook
cooked
cookie
cookies
cooking
cooks
Copenhagen
København
München
Munich
```

ångström

These are just 11 words, but the procedure will work fine if you have a million of them, and enough memory in your computer.

You can read the file into a Strings object with Read Strings from raw text file... from the Read menu in the Objects window. The resulting Strings object contains 11 strings in the above order, as you can verify by viewing them with Inspect.

In general, the Strings object will occupy a lot of memory, and be slow to read in. For instance, a certain list of more than 300,000 Dutch word forms occupies 3.6 MB on disk, and will occupy at least 7 MB of memory after it is read in. The extra 3.4 MB arise because the Strings object contains a pointer to each of the strings, and each of the strings is in a separately allocated part of the memory heap. Moreover, it takes 8 seconds on an average 1999 computer to read this object into memory. For these reasons, we will use the WordList object if we need a sorted list for spelling checking.

If you select the Strings, you can click the **To WordList** button. However, you will get the following complaint:

```
String "København" not generic. Please genericize first.
```

This complaint means that the strings are still in your computer's native text format, which is ISO-Latin1 for Unix and Windows computers, or Mac encoding for Macintosh computers.

So you press the **Genericize** button. You can see that the Strings object changes to

```
cook  
cooked  
cookie  
cookies  
cooking  
cooks  
Copenhagen  
K\o/bnhavn  
M\u"nchen  
Munich  
\aongstr\o"m
```

The strings are now in the generic system-independent format that is used everywhere in Praat to draw strings (see Special symbols).

You can again try to click the **To WordList** button. However, you will get a complaint again:

```
String "Copenhagen" not sorted. Please sort first.
```

This complaint means that the strings have not been sorted in ASCII sorting order. So you click **Sort**, and the Strings object becomes:

```
Copenhagen
K\o/bnhavn
M\u"nchen
Munich
\aongstr\o"m
cook
cooked
cookie
cookies
cooking
cooks
```

The strings are now in the ASCII order, in which capitals come before lower-case letters, and backslashes come in between these two series.

Clicking **To WordList** now succeeds, and a WordList object appears in the list. If you write it to a text file (with the Write menu), you will get the following file:

```
File type = "ooTextFile"
Object class = "WordList"
string = "Copenhagen
K\o/bnhavn
M\u" "nchen
Munich
\aongstr\o" "m
cook
cooked
cookie
cookies
cooking
cooks"
```

Note that the double quotes (") that appear inside the strings, have been doubled, as is done everywhere inside strings in Praat text files.

After you have created a WordList text file, you can create a WordList object just by reading this file with Read from file... from the Read menu.

The WordList object has two advantages over the Strings object. First, it won't take up more memory than the original word list. This is because the WordList is stored as a single string: a contiguous list of strings, separated by new-line symbols. Thus, our 300,000-word list will take up only 3.6 MB, and be read in 4 seconds.

However, disk storage and reading can again be improved by compressing the word list. We can take advantage of the sorting, by noting for each entry how many leading characters are equal to those of the previous entry. The list then becomes something equivalent to

Copenhagen
0 K\o/bnhavn
0 M\u"nchen
1 unich
0 \aongstr\o"m
0 cook
4 ed
4 ie
6 s
5 ng
4 s

You can write the WordList compressed in this way to a binary file with Write to binary file.... For our 300,000-word list, this file takes up only 1.1 MB and can be read into memory (with Read from file...) in a single second. When read into memory, the WordList object is again expanded to 3.6 MB to allow rapid searching.

© ppgb, November 29, 1999

AffineTransform

One of the types of objects in PRAAT.

An affine transform is a combination of a linear transformation A and a translation t that transforms a vector x to a new vector y in the following way:

$$y = A x + t$$

Links to this page

- [AffineTransform: Invert](#)
- [Configuration & AffineTransform: To Configuration](#)
- [Configurations: To AffineTransform \(congruence\)...](#)
- [Procrustus](#)

© *djmw*, September 27, 2001



One of the types of objects in PRAAT.

An object of type CCA represents the Canonical correlation analysis of two multivariate datasets.

Commands

Creation:

- [TableOfReal: To CCA...](#)

Links to this page

- [CCA & Correlation: To TableOfReal \(loadings\)](#)
- [CCA & TableOfReal: To TableOfReal \(loadings\)](#)
- [CCA & TableOfReal: To TableOfReal \(scores\)...](#)
- [CCA: Get zero correlation probability...](#)
- [SSCP: To CCA...](#)

© *djmw*, March 23, 2002

Procrustus

One of the types of objects in PRAAT.

An object of type Procrustus represents the special affine transform that consists of a combination of a translation, a shape preserving transformation and a scaling (this scaling is often called *dilation*). Because the transformation has to be shape preserving, only a combination of a rotation and a reflection is allowed. A configuration matrix X is transformed in the following way to a new configuration matrix Y :

$$Y = s X T + \mathbf{1}t',$$

where s is the scaling factor, T is the shape preserving transformation matrix, t is the translation vector, and $\mathbf{1}$ is the vector with only ones as its elements.

For more information about the Procrustus transform and its algorithm see chapter 19 in Borg & Groenen (1997).

Links to this page

- [Configuration & Procrustus: To Configuration](#)
- [Configurations: To Procrustus](#)

© *djmw*, September 27, 2001

Proximity

One of the types of objects in PRAAT.

An object of type **Proximity** represents proximities between objects.

Inside a Proximity

With Inspect you will see the following attributes:

numberOfRows, numberOfColumns

the number of objects (*numberOfRows* and *numberOfColumns* are equal and ≥ 1).

rowLabels, columnLabels

the names associated with the objects (*rowLabels* and *columnLabels* are equal).

data [1..numberOfRows][1..numberOfColumns]

the proximities between the objects.

© djmw, October 8, 1996

Matrix: Solve equation...

Solve the general matrix equation $\mathbf{A} \mathbf{x} = \mathbf{b}$ for \mathbf{x} .

The matrix \mathbf{A} can be any general $m \times n$ matrix, \mathbf{b} is a m -dimensional and \mathbf{x} a n -dimensional vector. The Matrix contains \mathbf{A} as its first n columns and \mathbf{b} as its last column. The n -dimensional solution is returned as a **Matrix** with n columns.

When the number of equations (m) is *greater* than the number of unknowns (n) the algorithm gives the best least-squares solution. If on the contrary you have *fewer* equations than unknowns the solution will not be unique.

Method

Singular value decomposition with backsubstitution. Zero will be substituted for eigenvalues smaller than $tolerance \cdot maximum_eigenvalue$ (when the user-supplied *tolerance* equals 0.0 a value of $2.2 \cdot 10^{-16} \cdot number_of_unknowns$ will be used as *tolerance*).

See for more details: Golub & van Loan (1996) chapters 2 and 3.

Links to this page

- [What's new?](#)

© djmw, October 6, 1996

Formulas 5. String functions

String functions are functions that either return a text string or have at least one text string as an argument. Since string computations are not very useful in the calculator, in settings windows, or in creation and modification formulas, this page only gives examples of strings in scripts, so that the example may contain string variables.

length (a\$)

gives the length of the string. After

```
string$ = "hallo"  
length = length (string$ + "dag")
```

the variable *length* contains the number 8. From this example, you see that variables can have the same names as functions, without any danger of confusing the interpreter).

left\$ (a\$, n)

gives a string consisting of the first *n* characters of *a\$*. After

```
head$ = left$ ("hallo", 3)
```

the variable *head\$* contains the string "hal".

right\$ (a\$, n)

gives a string consisting of the last *n* characters of *a\$*. After

```
english$ = "he" + right$ ("hallo", 3)
```

the variable *english\$* contains the string "hello".

mid\$ ("hello" , 3, 2)

gives a string consisting of 2 characters from "hello", starting at the third character. Outcome: ll.

index (a\$, b\$)

gives the index of the first occurrence of the string *b\$* in the string *a\$*. After

```
where = index ("hallo allemaal", "al")
```

the variable *where* contains the number 2, because the first "al" starts at the second character of the longer string. If the first string does not contain the second string, *index* returns 0.

rindex (a\$, b\$)

gives the index of the last occurrence of the string *b\$* in the string *a\$*. After

```
where = rindex ("hallo allemaal", "al")
```

the variable *where* contains the number 13, because the last "al" starts at the 13th character. If the first string does not contain the second string, *rindex* returns 0.

fixed\$ (number, precision)

formats a number as a string with *precision* digits after the decimal point. Thus, *fixed\$* (72.65687, 3) becomes the string 72.657, and *fixed\$* (72.65001, 3) becomes the string 72.650. In these examples, we see that the result can be rounded up and that trailing zeroes

are kept. At least one digit of precision is always given, e.g. `fixed$ (0.0000157, 3)` becomes the string `0.00002`. The number 0 always becomes the string 0.

percent\$ (number, precision)

the same as **fixed\$**, but with a percent sign. For instance, `percent$(0.157, 3)` becomes `15.700%`, `percent$(0.000157, 3)` becomes `0.016%`, and `percent$(0.000000157, 3)` becomes `0.00002%`. The number 0 always becomes the string 0.

date\$ ()

gives the date and time in the following format:

```
Mon Jun 24 17:11:21 2002
```

To write the day of the month into the Info window, you type:

```
date$ = date$ ()
day$ = mid$ (date$, 9, 2)
echo The month day is 'day$'.
```

extractNumber ("Type: Sound'newline\$'Name: hello there'newline\$'Size: 44007", "Size:")

looks for a number after the first occurrence of "Size:" in the long string. Outcome: 44007. This is useful in scripts that try to get information from long reports, as the following script that runs in the Sound editor window:

```
report$ = Settings report
maximumFrequency = extractNumber (report$, "Spectrogram maximum
frequency:")
```

extractWord\$ ("Type: Sound'newline\$'Name: hello there'newline\$'Size: 44007", "Type:")

looks for a word without spaces after the first occurrence of "Type:" in the long string. Outcome: Sound.

extractLine\$ ("Type: Sound'newline\$'Name: hello there'newline\$'Size: 44007", "Name: ")

looks for the rest of the line (including spaces) after the first occurrence of "Name: " in the long string. Outcome: hello there. Note how "Name: " includes a space, so that the 'rest of the line' starts with the *h*.

Links to this page

- [Formulas](#)
- [What's new?](#)

© ppgb, April 14, 2004

Manual

The documentation system for the PRAAT program.

You will get a manual window every time you choose anything from a **Help** menu or press a **Help** button.

How to find what you are looking for

You can navigate the manual in several ways:

- To go to the Intro, use the **H** button.
- To go to the information behind a *link* (a piece of blue text), **click** on it.
- To go forward and backward through a tutorial with numbered pages, use "**1 >**" and "**< 1**".
- To *revisit* previous pages, use the **<** and **>** buttons.
- To browse *alphabetically*, use the horizontal scroll bar and the buttons named "**< 1**" and "**1 >**", or the **Search for page (list)...** command in the **Go to** menu.
- To find a page with a *known title*, use the **Search for page...** command.

The fastest way to find what you want is often the **Search** button.

Search

In the text field after the Search button, you can type strings, separated by spaces. When you press the **Return** (or **Enter**) key, or click the **Search** button, all manual pages are searched for the combination of strings that you typed. The titles of the 20 best matching pages are displayed as links.

Example: to know how to create a pitch contour from a sound, type

```
sou pit
```

and press **Return**. The best matches should appear on top. These should include **Sound: To Pitch (ac)...** and **Sound: To Pitch (cc)...**

The search is case-insensitive. For instance, the search string "script" will give you all the pages that contain the words *script*, *Script*, *description*, *PostScript*, or *SCRIPT*, and so on.

Background. The search algorithm uses the following heuristics:

- A match in the page title is better than one in the rest of the text.
- Pages with many matches are better than those with few.

Your own man pages

To create your own man pages, create ManPages text files.

Links to this page

- [File menu](#)
- [View](#)
- [What's new?](#)

© *ppgb*, February 9, 2004

T-test

A test on the mean of a normal variate when the variance is unknown.

In Praat, the t-test is used to query a Covariance object and:

1. get the significance of one mean. See Covariance: Get significance of one mean....
2. get the significance of the *difference* between two means. See Covariance: Get significance of means difference....

You should use a t-test when you want to test a hypothesis about the mean of one column in a TableOfReal object, or, if you want to test a hypothesis about the difference between the means of two columns in this object.

You can perform these t-tests in Praat by first transforming the TableOfReal object into a Covariance object (see TableOfReal: To Covariance) and then choosing the appropriate query method on the latter object.

Links to this page

- [What's new?](#)

© djmw, January 17, 2002

Correlation: Confidence intervals...

Calculates confidence intervals for the correlation coefficients from the selected Correlation object(s) and saves these intervals in a new TableOfReal object.

Arguments

Confidence level

the confidence level you want for the confidence intervals.

Number of tests

determines the Bonferroni correction for the significance level. If the default value (zero) is chosen, it will be set equal to the number of correlations involved (a matrix of dimension n has $n \cdot (n-1)/2$ correlations).

Approximation

defines the approximation that will be used to calculate the confidence intervals. It is either Fisher's z transformation or Ruben's transformation. According to Boomsma (1977), Ruben's approximation is more accurate than Fisher's.

Algorithm

We obtain intervals by the large-sample conservative multiple tests with Bonferroni inequality and the Fisher or Ruben transformation. We put the upper values of the confidence intervals in the upper triangular part of the matrix and the lower values of the confidence intervals in lower triangular part of the resulting TableOfReal object.

In *Fisher's approximation*, for each element r_{ij} of the correlation matrix the confidence interval is:

$$[\tanh (z_{ij} - z_{\alpha} / \sqrt{N - 3}) , \tanh (z_{ij} + z_{\alpha} / \sqrt{N - 3})],$$

where z_{ij} is the Fisher z -transform of the correlation r_{ij} :

$$z_{ij} = 1/2 \ln ((1 + r_{ij}) / (1 - r_{ij})),$$

z_{α} , the Bonferroni corrected z -value $z_{\alpha / (2 \cdot \text{numberOfTests})}$,

$$\alpha = 1 - \text{confidenceLevel},$$

and N the number of observations that the correlation matrix is based on.

In *Ruben's approximation* the confidence interval for element r_{ij} is:

$$[x_1 / \sqrt{1 - x_1^2}, x_2 / \sqrt{1 - x_2^2}]$$

in which x_1 and x_2 are the smallest and the largest root from

$$a x^2 + b x + c = 0, \text{ with}$$

$$a = 2N - 3 - z\alpha,^2$$

$$b = -2 r' \sqrt{(2N - 3)(2N - 5)}$$

$$c = (2N - 5 - z\alpha,^2) r'^2 - 2z\alpha,^2, \text{ and}$$

$$r' = r_{ij} / \sqrt{1 - r_{ij}^2},$$

Links to this page

- [What's new?](#)

© djmw, April 7, 2004

SSCP: Get diagonality (bartlett)...

Tests the hypothesis that the selected SSCP matrix object is diagonal.

Arguments

Number of constraints

modifies the number of independent observations. The default value is 1.

Algorithm

The test statistic is $|\mathbf{R}|^{N/2}$, the $N/2$ -th power of the determinant of the correlation matrix. Bartlett (1954) developed the following approximation to the limiting distribution:

$$\chi^2 = -(N - \text{numberOfConstraints} - (2p + 5) / 6) \ln |\mathbf{R}|$$

In the formula's above, p is the dimension of the correlation matrix, $N - \text{numberOfConstraints}$ is the number of independent observations. Normally *numberOfConstraints* would equal 1, however, if the matrix has been computed in some other way, e.g., from within-group sums of squares and cross-products of k independent groups, *numberOfConstraints* would equal k .

We return the probability α as

$$\alpha = \text{chiSquareQ}(\chi^2, p(p-1)/2).$$

A very low α indicates that it is very improbable that the matrix is diagonal.

Links to this page

- [What's new?](#)

© djmw, November 11, 2001

LPC: To Matrix

Copies the linear prediction coefficients of the selected LPC object to a newly created Matrix object.

Behaviour

$$z_{ji} = a_{ij}, \text{ with } 1 \leq i \leq nx \text{ and } 1 \leq j \leq nCoefficients_i,$$

where z_{ji} is the matrix element in row j and column i and a_{ij} is the j -th linear prediction coefficient in frame i .

Links to this page

- [What's new?](#)

© *djmw*, November 23, 2001

CC: To Matrix

Copies the cepstral coefficients of the selected CC object to a newly created Matrix object.

Behaviour

$z_{ji} = c_{ij}$, with $1 \leq i \leq nx$ and $1 \leq j \leq numberOfCoefficients_i$,

where z_{ji} is the matrix element in row j and column i and c_{ij} is the j -th cepstral coefficient in frame i .

Links to this page

- [What's new?](#)
-

© djmw, November 23, 2001

Special symbols

When drawing text into the Picture window or into an editor, you can use *backslash sequences* to display various kinds of special symbols.

European symbols

To get the symbol "â" (a-circumflex), you type "\a^", i.e., a sequence of backslash + a + circumflex. In this way, you can get a hundred non-ASCII symbols that are used in the alphabets of many European languages. You can also use these symbols in info messages sent from scripts.

```

ä \a" ë \e" ì \i" ö \o" ü \u" ÿ \y" Ä \A" È \E" Ì \I" Ö \O" Ü \U" ? \Y"
á \a' é \e' í \i' ó \o' ú \u' Á \A' É \E' Í \I' Ó \O' Ú \U'
à \a` è \e` ì \i` ò \o` ù \u` À \A` È \E` Ì \I` Ò \O` Ù \U`
â \a^ ê \e^ î \i^ ô \o^ û \u^ Â \A^ Ê \E^ Î \I^ Ô \O^ Û \U^
ã \a~ ñ \n~ õ \o~ Ã \A~ Ñ \N~ Õ \O~
æ \ae ø \o/ å \ao Æ \Ae Ø \O/ Å \Ao ç \c, Ç \C, ß \ss
ı \!d ı \?d
¤ \cu (currency), £ \Lp (sterling), ¥ \Y= (yen), f \fd (Dutch florin), ¢ \c/ (cent)
§ \SS (section), ¶ \|| (paragraph)
© \co (copyright), ® \re (registered), ™ \tm (trademark)
ª \a_ (ordfeminine), º \o_ (ordmasculine)
« \<< (guillemotleft), » \>> (guillemotright)

```

Mathematical symbols

```

· \c (periodcentered), × \xx (multiply), ÷ \:- (divide), / \d (fraction)
° \dg (degree), ' \p (minute or prime), " \p (second or double prime)
-- (minus), -|-- (endash), ± \+- (plusminus)
≤ \<_ (lessequal), ≥ \>_ (greaterequal), ≠ \=/ (notequal)
\no \no (logicalnot), \an \an (logicaland), \or \or (logicalor)
\At \At (universal), \Er \Er (existential), \3 \3 (therefore)
∝ \oc (proportional), ≡ \=3 (equivalence), ≈ \~~ (approxequal)
√ \Vr (radical)
<- \<- (arrowleft), -> \-> (arrowright), <-> \<> (arrowboth)
<=> \<=> (arrowdblleft), => \=> (arrowdblright), <=> \eq (arrowdblboth)
\^ \^ (arrowup), ≈ \≈ (congruent), \_ \_ (arrowdown)
∞ \oo (infinity), ⊥ \Tt (perpendicular)
∅ \O| (emptyset), ∩ \ni (intersection), ∪ \uu (union), ⊂ \c= \c= (probersubset), ∈ \e= (element)
∂ \dd (partialdiff)
⊗ \ox (circlemultiply), ⊕ \o+ \o+ (circleplus)
∑ \su (summation), ∫ \in (integral)

```

Greek letters

To get $\epsilon\rho\eta\kappa\alpha$, you type `\ep\up\rro\et\ka\al`.

α \al A \Al *alpha*

β \be B \Be *beta*

γ \ga Γ \Ga *gamma*

δ \de Δ \De *delta*

ϵ \ep E \Ep *epsilon*

ζ \ze Zeta \Ze *zeta*

η \et H \Et *eta*

θ \te Θ \Te *theta* θ \t2

ι \io I \Io *iota*

κ \ka K \Ka *kappa*

λ \la Λ \La *lambda*

μ \muM \Mu *mu*

ν \nu N \Nu *nu*

ξ \xi Ξ \Xi *xi*

\omicron \on O \On *omicron*

π \pi Π \Pi *pi*

ρ \ro P \Ro *rho*

σ \si Σ \Si *sigma* ς \s2

`\ta T \Ta tau`

`\up \Upsilon \Up \upsilon`

`\phi \Phi \phi \phi \f2`

`\chi \Chi \chi`

`\psi \Psi \psi`

`\omega \Omega \omega \o2`

Phonetic symbols

See Phonetic symbols

Miscellaneous

`\bs backslash`

`\bu bullet`

`\cl \cl (club), \di \di (diamond), \he \he (heart), \sp \sp (spade)`

See also

Text styles

Links to this page

- [Create tongue-root grammar...](#)
- [Font menu](#)
- [IntervalTier](#)
- [ManPages](#)
- [OTAnyGrammar examples](#)
- [OTGrammar_tongueRoot](#)
- [Strings](#)
- [Text...](#)
- [TextGridEditor](#)
- [TextTier](#)
- [Viewport text...](#)
- [What's new?](#)
- [WordList](#)

© *ppgb*, November 2, 2003

Regular expressions

This tutorial describes the *syntax* of regular expressions in PRAAT

Introduction

A *regular expression* is a text string that describes a *set* of strings. Regular expressions (regex) are useful as a way to search for patterns in text strings and, optionally, replace them by another pattern.

Some regex match only one string, i.e., the set they describe has only one member. For example, the regex "ab" matches the string "ab" and no others. Other regex match more than one string, i.e., the set they describe has more than one member. For example, the regex "a*" matches the string made up of any number (including zero) of "a"s. As you can see, some characters match themselves (such as "a" and "b") and these characters are called *ordinary* characters. The characters that don't match themselves, such as "*", are called *special* characters or *meta* characters. Many special characters are only special characters in the *search* regex and are ordinary characters in the substitution regex.

You can read the rest of this tutorial sequentially with the help of the "<1" and ">1" buttons.

1. Special characters (`\ ^ $ { } [] () . + ? | - &`)
2. Quantifiers (how often do we match).
3. Anchors (where do we match)
4. Special constructs with parenthesis (grouping constructs)
5. Special control characters (difficult-to-type characters like `\n`)
6. Convenience escape sequences (`\d \D \l \L \s \S \w \W \B`)
7. Octal and hexadecimal escapes (things like `\053` or `\X2B`)
8. Substitution special characters (`\1..\9 \U \u \L \l &`)

More in depth coverage of regular expressions can be found in Friedl (1997).

Links to this page

- Confusion: Condense...
- TableOfReal: Change column labels...
- TableOfReal: Change row labels...
- What's new?

© djmw, July 6, 2001

Matrix: To TableOfReal

A command to convert every selected Matrix to a TableOfReal.

This command is available from the Cast submenu. The resulting TableOfReal has the same number of rows and columns as the original Matrix, and the same data in the cells. However, it does not yet have any row or column labels; you can add those with some commands from the TableOfReal Modify submenu.

Links to this page

- [What's new?](#)

© *ppgb*, October 30, 1999

TableOfReal: Draw box plots...

A command to draw a box plot for each column in the selected TableOfReal object.

Arguments

From row, To row, From column, To column

determine the part of the table that you want to analyse.

Ymin and Ymax

determine the drawing boundaries.

Links to this page

- [What's new?](#)
-

© *djmw, May 23, 2000*

Covariance: To TableOfReal (random sampling)...

Generate a TableOfReal object by random sampling from a multi-variate normal distribution whose Covariance matrix is the selected object.

Arguments

Number of data points

determines the number of data points that will be generated. Each data point occupies one row in the generated table.

Algorithm

The algorithm proceeds as follows:

1. Calculate the eigenvalues v_i and eigenvectors e_i of the $m \times m$ Covariance matrix. In general there will also be m of these. Let E be the $m \times m$ matrix with eigenvector e_j in column j ($j=1..m$).
2. Generate a vector \mathbf{x} whose elements x_k equal $x_k = \text{randomGauss}(0, \sqrt{v_k})$. Each x_k is a random deviate drawn from a Gaussian distribution with mean zero and standard deviation equal to the square root of the corresponding eigenvalue v_k .
3. Calculate the vector $\mathbf{y} = E \mathbf{x}$, obtained by multiplying the vector \mathbf{x} with the matrix E .
4. Add the centroid to \mathbf{y} and copy the elements of \mathbf{y} to the corresponding row of the TableOfReal object.
5. Repeat steps 2, 3 and 4 until the desired number of data points has been reached.
6. Copy the column labels from the Covariance object to the TableOfReal object.

Links to this page

- [What's new?](#)

© djmw, April 7, 2004

SSCP: Get sigma ellipse area...

A command to query the selected SSCP object for the area of a sigma ellipse.

Algorithm

The algorithm proceeds as follows:

1. The four array elements in the SSCP-matrix that correspond to the chosen dimensions are copied into a two-dimensional matrix S (symmetric of course).
2. The eigenvalues of S are determined, call them s_1 and s_2 .
3. The lengths l_i of the axes of the ellipse can be obtained as the square root of the s_i multiplied by a scale factor: $l_i = scaleFactor \cdot \sqrt{s_i}$, where $scaleFactor = numberOfSigmas / \sqrt{numberOfObservations - 1}$.
4. The area of the ellipse will be $\pi \cdot l_1 \cdot l_2$.

Links to this page

- Discriminant: Get concentration ellipse area...
- What's new?

© djmw, May 25, 2000

Formulas 4. Mathematical functions

abs (x)

absolute value

round (x)

nearest integer; round (1.5) = 2

floor (x)

round down: highest integer value not greater than x

ceiling (x)

round up: lowest integer value not less than x

sqrt (x)

square root: \sqrt{x} , $x \geq 0$

min (x, ...)

the minimum of a series of numbers, e.g. min (7.2, -5, 3) = -5

max (x, ...)

the maximum of a series of numbers, e.g. max (7.2, -5, 3) = 7.2

imin (x, ...)

the location of the minimum, e.g. imin (7.2, -5, 3) = 2

imax (x, ...)

the location of the maximum, e.g. imax (7.2, -5, 3) = 1

sin (x)

sine

cos (x)

cosine

tan (x)

tangent

arcsin (x)

arcsine, $-1 \leq x \leq 1$

arccos (x)

arccosine, $-1 \leq x \leq 1$

arctan (x)

arctangent

arctan2 (y, x)

argument angle

exp (x)

exponentiation: e^x ; same as e^x

ln (x)

natural logarithm, base e

log10 (x)

logarithm, base 10

log2 (x)

logarithm, base 2

sinh (x)

hyperbolic sine: $(e^x - e^{-x}) / 2$

cosh (x)

hyperbolic cosine: $(e^x + e^{-x}) / 2$

tanh (x)

hyperbolic tangent: $\sinh (x) / \cosh (x)$

arcsinh (x)

inverse hyperbolic sine: $\ln (x + \sqrt{1+x^2})$

arccosh (x)

inverse hyperbolic cosine: $\ln (x + \sqrt{x^2-1})$

arctanh (x)

inverse hyperbolic tangent

sigmoid (x)

$\mathbf{R} \rightarrow (0,1)$: $1 / (1 + e^{-x})$ or $1 - 1 / (1 + e^x)$

erf (x)

the error function: $2/\sqrt{\pi} \int_0^x \exp(-t^2) dt$

erfc (x)

the complement of the error function: $1 - \text{erf} (x)$

randomUniform (min, max)

uniform random deviate between *min* (inclusive) and *max* (exclusive)

randomInteger (min, max)

uniform random deviate between *min* and *max* (inclusive)

randomGauss (μ , σ)

Gaussian random deviate with mean μ and standard deviation σ

randomPoisson (mean)

Poisson random deviate

lnGamma (x)

logarithm of the Γ function

gaussP (z)

the area under the Gaussian distribution between $-\infty$ and z

gaussQ (z)

the area under the Gaussian distribution between z and $+\infty$: the one-tailed "statistical significance p " of a value that is z standard deviations away from the mean of a Gaussian distribution

invGaussQ (q)

the value of z for which $\text{gaussQ} (z) = q$

chiSquareP (chiSquare, df)

the area under the χ^2 distribution between 0 and *chiSquare*, for *df* degrees of freedom

chiSquareQ (chiSquare, df)

the area under the χ^2 distribution between *chiSquare* and $+\infty$, for *df* degrees of freedom: the "statistical significance p " of the χ^2 difference between two distributions in *df*+1 dimensions

invChiSquareQ (q, df)

the value of χ^2 for which $\text{chiSquareQ} (\chi^2, df) = q$

studentP (t, df)

the area under the student T-distribution from $-\infty$ to t

studentQ (t, df)

the area under the student T-distribution from t to $+\infty$

invStudentQ (q, df)

the value of t for which $\text{studentQ}(t, df) = q$

fisherP ($f, df1, df2$)

the area under Fisher's F-distribution from 0 to f

fisherQ ($f, df1, df2$)

the area under Fisher's F-distribution from f to $+\infty$

invFisherQ ($q, df1, df2$)

the value of f for which $\text{fisherQ}(f, df1, df2) = q$

binomialP (p, k, n)

the probability that in n experiments, an event with probability p will occur at most k times

binomialQ (p, k, n)

the probability that in n experiments, an event with probability p will occur at least k times; equals $1 - \text{binomialP}(p, k - 1, n)$

invBinomialP (P, k, n)

the value of p for which $\text{binomialP}(p, k, n) = P$

invBinomialQ (Q, k, n)

the value of p for which $\text{binomialQ}(p, k, n) = Q$

hertzToBark (x)

from acoustic frequency to Bark-rate (perceptual spectral frequency; place on basilar membrane): $7 \ln(x/650 + \sqrt{1 + (x/650)^2})$

barkToHertz (x)

$650 \sinh(x/7)$

hertzToMel (x)

from acoustic frequency to perceptual pitch: $550 \ln(1 + x/550)$

melToHertz (x)

$550 (\exp(x/550) - 1)$

hertzToSemitones (x)

from acoustic frequency to a logarithmic musical scale, relative to 100 Hz: $12 \ln(x/100) / \ln 2$

semitonesToHertz (x)

$100 \exp(x \ln 2 / 12)$

erb (f)

the perceptual *equivalent rectangular bandwidth* (ERB) in Hertz, for a specified acoustic frequency (also in Hertz): $6.23 \cdot 10^{-6} f^2 + 0.09339 f + 28.52$

hertzToErb (x)

from acoustic frequency to ERB-rate: $11.17 \ln((x + 312) / (x + 14680)) + 43$

erbToHertz (x)

$(14680 d - 312) / (1 - d)$ where $d = \exp((x - 43) / 11.17)$

phonToDifferenceLimens (x)

from perceptual loudness (intensity sensation) level in phon, to the number of intensity difference limens above threshold: $30 \cdot ((61/60)^x - 1)$.

differenceLimensToPhon (x)

the inverse of the previous: $\ln(1 + x/30) / \ln(61/60)$.

beta (x, y)

bessell (n, x)

besselK (n, x)

Links to this page

- [Formulas](#)
- [What's new?](#)

© *ppgb*, July 26, 2003

sendpraat

See Scripting 8. Controlling Praat from another program.

Links to this page

- [Scripting 8.3. The sendpraat directive](#)
- [What's new?](#)

© *ppgb*, September 27, 2000

ManPages

You can create a documentation or education system with files that you and others can read into PRAAT (with the Read from file... command). Your files will become a hypertext system very similar to the usual Manual.

Example 1: a single document

If you create a single ManPages file, it will look like a manual with a single page. Here is an example:

```
ManPagesTextFile
"Welkom" "miep" 19970820 0
<intro> "Hallo allemaal!"
<entry> "Belangrijk..."
<normal> "Hoogge\e"erd publiek!"
<normal> "Einde."
```

A ManPages file should start with the following information:

1. The word "ManPagesTextFile" on the first line.
2. The title of the man page, between double quotes. This will be drawn at the top of the page. The name of the ManPages file should be derived from the title (see below).
3. The author of the man page, between double quotes. This will be drawn at the bottom of the page.
4. The date you created or modified the page, in the format year - month (two digits) - day (two digits), without spaces.
5. The recording time. If this is not zero, three sound buttons (see below) will appear at the top of the page.
6. A sequence of paragraph types and texts. You put the types between < and >, and the texts between double quotes (if your text contains a double quote, you should write two double quotes).

The format of a ManPages text file is rather free, as long as the first line is correct, the four required pieces of information are there in the correct order, and there is a correct alternation between paragraph texts and types. If you put multiple elements on a line, there should be at least one space between them. You may distribute texts across multiple lines, as long as you do not add any spaces:

```
<normal> "Hoogge\e"erd
publiek!"
```

This will have exactly the same effect as above.

Example 2: multiple documents

The above example with a single document is not very useful. You will usually want to refer to other documents:

```

ManPagesTextFile
"Welcome" "ppgb" 19970820 1.0
<intro> "Welcome to our transcription course."
<entry> "Groups of speech sounds"
<normal> "You can listen to the following sounds
from the languages of the world,
pronounced by a single speaker (me):"
<list_item> "@Vowels, quite problematic for Dutch students!"
<list_item> "@@Dorsal fricatives@, equally problematic!"

```

With the symbol '@', you create a *link* to another ManPages file. A link will be drawn in blue on your screen. In this example, you have created links to the files **Vowels.man** and **Dorsal_fricatives.man** in the same directory as the current file (all **.man** files have to be in the same directory; this makes it likely that their names are unique). If the link contains spaces or other non-alphanumeric symbols, you have to use three '@' symbols, as shown; with a single word, you may use a single '@'.

In resolving the file name, the ManPages system replaces spaces and other special symbols with underscores, and converts any initial lower-case character by its upper-case variant. For instance, if your link is "@@back vowels@", the file name will be **Back_vowels.man**.

The *title* attribute of **Back_vowels.man** must be equal to the link name, though capitalization of the first letter is allowed. Thus, the title of **Back_vowels.man** will probably be "Back vowels". Likewise, the starting file with the title "Welcome" should have the name **Welcome.man** if any other files refer to it.

Paragraph types

A normal paragraph will have type <normal>. The hypertext system will leave a blank space between paragraphs with this type. The first paragraph of a man page will normally have the type <intro>. Though this may look the same as <normal>, the search system of the Manual may take account of the distinction.

Headings (like the title "Paragraph types" of this subsection) have type <entry>. This will be drawn in a larger character size.

For lists, you use the type <list_item>. You will often combine this with *button* symbols, like in the following:

```

<normal> "Choose a colour:"
<list_item> "\bu @Red."
<list_item> "\bu @Green."
<list_item> "\bu @Blue."

```

For text that should appear with a fixed character width, you use the type <code>.

For a paragraph that should be connected with the following paragraph without leaving a blank space (probably a list item or a definition), you use the type <tag>.

For a paragraph with a blank left margin, you use the type <definition>.

Special symbols and styles

You can use all of PRAAT's special symbols and text styles, except that some *single* text-style symbols have different meanings:

- A single percent sign introduces a word in italic: %pot gives *pot*.
- A single number sign introduces a word in bold: #pot gives **pot**.
- A single dollar sign introduces a word in monospace: \$pot gives pot.
- A single underscore is rendered as an underscore: a_b gives a_b.

To create a single italic, bold, or subscripted letter, you revert to the usual technique for stretches of italic, bold, or subscripted text. So, to get F_1 , you type %F__1_.

Sound links

Your text may contain links to sound files. They are drawn in blue. The format is:

```
<normal> "You should know that @@\FIct.aifc|ct@ is more open than
@@\FIo.aifc|o@."
```

On your screen, you will see an  and an **o** symbol, both drawn in blue. If you click on one of these sound links, one of the sound files **ct.aifc** or **o.aifc** will be played.

The format of the sound link "@@\FIo.aifc|o@" is to be understood as follows. The pipe symbol separates the link information (\FIo.aifc) from the viewable link text (o). The link information is introduced with a symbol (\FI) that tells the manual system that a sound file name follows. The manual system reads this file, sees that it contains a sound, and plays that sound. You can use relative path names, e.g., \FIsounds/o.aifc refers to the file **o.aifc** in the subdirectory **sounds**, which must be contained in the same directory as the **.man** files; use the forward slash (/) if you want your man pages to run on all platforms (Windows, Macintosh, Unix).

Script links

Your text may contain links to Praat scripts. They are drawn in blue. The format is:

```
<normal> "Here is a script that @@\SCdraw.praat|draws@ what you have
selected."
```

On your screen, you will see the word **draws**, drawn in blue. If you click on it, the script **draw.praat** will be executed. The string "\SC" indicates that a script file name follows. As with sounds, you can use relative file paths. The script can take arguments:

```
<normal> "This script @@\SCload2files.praat hello.wav
hello.TextGrid|loads@ some files."
```

If the script file name or any arguments except the last contain spaces, they have to be enclosed within double quotes, which have to be doubled again in the ManPage code:

```
<normal> "Here is a script that @@\SC" "my scripts/draw.praat" " |draws@  
what you have selected."
```

```
<normal> "This script @@\SC" "my scripts/load2files.praat" " " "my  
sounds/hello.wav" " my textgrids/hello.TextGrid|loads@ some files."
```

How to separate link information and link text

The separation between link information and viewable link text, as seen in the above description of sound and script links, is actually a general mechanism in the hypertext system. For instance, if you want to see the text "x" in blue on your screen, and create a link from it to the page "Dorsal fricatives", you specify the link as @@Dorsal fricatives|x@.

Sound buttons

If the *Recording time* attribute is not 0, three buttons will appear at the top of the page:

1. The **Rec** button allows you to record a sound from the microphone.
2. The **Play** button allows you to play this recorded sound. In this way, you can compare your own utterances with the sounds behind the sound links, for example.
3. The **Pub** button copies the latest sound to the list of objects, allowing you to perform analyses on it, save it to disk, et cetera. The latest sound may be a sound that you recorded with the **Rec** button, a sound that you played with the **Play** button, or a sound that you played by clicking on a sound link, whichever occurred most recently.

Links to this page

- What's new?

© ppgb, February 8, 2004

Create Strings as file list...

A command to create a Strings object from a list of file in a given directory.

Argument

Path

the directory name, with an optional wildcard for selecting files.

Behaviour

The resulting Strings object will contain an alphabetical list of file names, without any preceding directory names. In contrast with what the restrictions in Praat, this object may contain zero strings.

Usage

First, you could specify a directory name only. On Unix, you would type **/usr/people/miep/sounds** or **/usr/people/miep/sounds/**. On Windows, **C:\Miep\Sounds** or **C:\Miep\Sounds**. On Macintosh, **Macintosh HD:Sounds** or **Macintosh HD:Sounds:**. Any of these return a list of all the files in the specified directory.

Secondly, you could specify a wildcard (a single asterisk) for the file names. To get a list of all the files whose names start with "hal" and end in ".aifc", type **/usr/people/miep/sounds/hal*.aifc**, **C:\Miep\Sounds\hal*.aifc**, or **Macintosh HD:Sounds:hal*.aifc**.

Script usage

In a script, you can use this command to cycle through the files in a directory. For instance, to read in all the sound files in a specified directory, you could use the following script:

```
directory$ = "/usr/people/miep/sounds"
Create Strings as file list... list 'directory$'/*.aifc
numberOfFiles = Get number of strings
for ifile to numberOfFiles
  select Strings list
  fileName$ = Get string... ifile
  Read from file... 'directory$'/'fileName$'
endfor
```

Links to this page

- [FAQ: Scripts](#)
- [New menu](#)
- [Scripting 6.5. Files](#)

- What's new?
-

© *ppgb*, June 30, 1998

TextGrid: Count labels...

A command to ask the selected TextGrid object how many of the specified labels it contains in the specified tier.

Arguments

Tier number

the number (1, 2, 3...) of the tier whose labels you want to investigate.

Label text

the text on the labels that you want to count.

Behaviour

The number of intervals or points with label *Label text* in tier *Tier number* is written into the Info window. If the specified tier does not exist, the number will be 0.

Scripting

You can use this command to put the number into a script variable:

```
select TextGrid hallo  
number_of_a = Count labels... 1 a
```

In this case, the value will not be written into the Info window.

Links to this page

- [What's new?](#)

© ppgb, June 30, 1998

PointProcess: To TextGrid (vuv)...

A command to create a TextGrid with voiced/unvoiced information from every selected PointProcess.

Arguments

Maximum period (s)

the maximum interval that will be consider part of a larger voiced interval.

Mean period (s)

half of this value will be taken to be the amount to which a voiced interval will extend beyond its initial and final points. *Mean period* must be less than *Maximum period*, or you may get intervals with negative durations.

Example

If *Maximum period* is 0.02 s, and *Mean period* is 0.01 s, and the point process is 0.1 seconds long, with points at 20, 28, 39, 61, and 72 milliseconds, the resulting TextGrid object will contain an interval tier with "U" intervals at [0 ms, 15 ms], [44 ms, 56 ms], and [77 ms, 100 ms], and "V" intervals at [15 ms, 44 ms] and [56 ms, 77 ms].

Links to this page

- [What's new?](#)

© ppgb, February 10, 1998

Text styles

When drawing text into the Picture window or into an editor, you can use text styles other than regular Roman.

Italic, bold, superscript, subscript

With the following symbols, you introduce stretches of text drawn in special styles:

%: the following letter will be italic.

#: the following letter will be bold.

#%: the following letter will be bold-italic.

^: the following letter will be superscript: %m%c^2 gives mc^2 .

_: the following letter will be subscript. Example: F_0 is typed as %F_0.

%%: the following letters will be italic, until the following %:

Now %%you% tRY gives: Now *you* try.

The same goes for ##, ^^, and __.

\s{...}: small:

W\s{ARP} gives: WARP

To draw a %, #, ^, or _ symbol, you type "\% ", "\# ", "\^ ", or "_ ": a backslash, the symbol, and a space.

On Xwindows machines, the font 'Symbol' will never look bold or italic, but it will be printed correctly.

See also

Special symbols

Links to this page

- [Font menu](#)
- [ManPages](#)
- [OT learning 2.2. Inside the grammar](#)
- [Text...](#)
- [Viewport text...](#)
- [What's new?](#)

© ppgb, October 12, 1996

Create Matrix...

A command in the New menu to create a Matrix with the specified sampling attributes, filled with values from a formula (see Matrix: Formula...).

© *ppgb*, December 12, 2002

Create simple Matrix...

A command in the New menu to create a Matrix with the specified number of rows and columns, filled with values from a formula (see Matrix: Formula...).

Links to this page

- [Formulas 1.7. Formulas for creation](#)
-

© *ppgb*, December 4, 2002

Create empty PointProcess...

A command in the New menu to create an empty PointProcess. The newly created object is put in the list of objects.

© *ppgb*, December 4, 2002

Create Poisson process...

A command to create a PointProcess object that represents a Poisson process.

A Poisson process is a stationary point process with a fixed density λ , which means that there are, on the average, λ events per second.

Arguments

Starting time (seconds)

t_{min} , the beginning of the time domain.

Finishing time (seconds)

t_{max} , the end of the time domain.

Density (Hertz)

the average number of points per second.

Algorithm

First, the number of points N in the time domain is determined. Its expectation value is

$$\lambda = (t_{max} - t_{min}) \cdot density$$

but its actual value is taken from the Poisson distribution:

$$p(n) = (\lambda^n / n!) e^{-\lambda}$$

Then, N points are computed throughout the time domain, according to a uniform distribution:

$$p(t) = 1 / (t_{max} - t_{min}) \text{ for } t \in [t_{min}, t_{max}]$$

$$p(t) = 0 \text{ outside } [t_{min}, t_{max}]$$

C implementation

PointProcess PointProcess_createPoisson (double *tmin*, double *tmax*, double *density*);

Links to this page

- [New menu](#)
-

© *ppgb*, April 2, 1997

Create DurationTier...

A command in the New menu to create an empty DurationTier object.

The resulting object will have the specified name and time domain, but contain no duration points. To add some points to it, use DurationTier: Add point...

Scripting example

To create a tier 0.9 seconds long, with an deceleration around 0.6 seconds, you do:

```
Create DurationTier... dur 0 0.9
Add point... 0.3 1
Add point... 0.6 2.3
Add point... 0.7 1
```

The result will look like

[sorry, no pictures yet in the web version of this manual]

The target duration will be the area under this curve, which is $0.9 + 1/2 \cdot 1.3 \cdot 0.4 = 1.16$ seconds.

Links to this page

- Manipulation: Replace duration tier

© ppgb, December 4, 2002

Add to fixed menu...

A command in the File menu of the ScriptEditor.

With this command, you add a button to any fixed menu in the Object window or in the Picture window. Clicking the added button will invoke the specified Praat script.

Arguments

Window

the name of the window ("Objects" or "Picture") that contains the menu that you want to change.

Menu

the title of the menu that you want to change. If *window* is "Objects", you can specify the Control, New, Read, Help, Goodies, or Preferences menu (for the Write menu, which depends on the objects selected, you would use Add to dynamic menu... instead). If *window* is "Picture", you can specify the File, Edit, Margins, World, Select, Pen, Font, or Help menu.

Command

the title of the new menu button. To get a separator line instead of a command text, you specify a unique string that starts with a hyphen ('-'); the ButtonEditor contains many examples of this.

After command

a button title in the menu or submenu after which you want your new button to be inserted. If you specify the empty string (""), your button will be put in the main menu.

Depth

0 if you want your button in the main menu, 1 if you want it in a submenu.

Script

the full path name of the script to invoke. If you saved the script you are editing, its name will already have been filled in here. If you do not specify a script, you will get a cascading menu title instead.

Example 1

In the **Matrix** submenu of the New menu, you want a separator line followed by the command "Peaks":

```
Add to fixed menu... Objects New "-- peaks --" "Create simple
Matrix..." 1
Add to fixed menu... Objects New "Peaks" "-- peaks --" 1
/u/praaats/demo/peaks.praat
```

Example 2

In the New menu, you want a submenu called "Demo", with a subitem titled "Lorenz...":

```
Add to fixed menu... Objects New "Demo" "" 0
Add to fixed menu... Objects New "Lorenz..." "Demo" 1
/u/praaats/demo/lorenz.praat
```

Usage convention

Please adhere to the convention that commands that take arguments, end in three dots (...).

Using this command in a script

To add a fixed button from a script (perhaps your Initialization script), use the hidden shell command `Add menu command...` instead.

Links to this page

- [History mechanism](#)
-

© *ppgb*, December 4, 2002

Formulas 1.6. Formulas in settings windows

Into numeric fields in settings windows you usually simply type a number. However, you can use any numeric expression instead.

For instance, suppose you want to create a Sound that contains exactly 10000 samples. If the sampling frequency is 22050 Hz, the duration will be $10000/22050$ seconds. You can create such a Sound by choosing Create Sound... from the New menu, then typing

$10000/22050$

into the *Finishing time* field.

Into text fields in settings windows, you can only type text directly; there is no way to use string expressions (except if you use scripts; see Formulas 1.9. Formulas in scripts).

Links to this page

- [Formulas](#)
- [Formulas 1. My first formulas](#)

© ppgb, April 14, 2004

Objects

The things that contain the data in the PRAAT program.

The objects are visible in the List of Objects in the Object window.

Most objects are contained in memory: they are *not* files! Therefore, you may want to save them with one of the commands from the Write menu before you Quit. Exceptions are the LongSound objects in the Praat program and the file-based dictionaries in the ALS program.

You can create an object by choosing a command from the New menu or from the Read menu.

When you select one or more objects, you can perform on them the actions that are shown in the dynamic menu, on the **fixed buttons** below the list of objects, or in the Write menu. You can choose hidden actions with the help of the ButtonEditor.

Links to this page

- [Edit](#)
- [Query](#)
- [Read from file...](#)
- [Remove](#)
- [Write to binary file...](#)
- [Write to short text file...](#)
- [Write to text file...](#)

© *ppgb*, May 28, 2003

Rename...

One of the fixed buttons in the Object window.

Availability

You can choose this command after selecting one object of any class.

Purpose

You can give the selected object a new name.

Behaviour

If you type special symbols or spaces, the Object window will replace them with underscores.

Links to this page

- [Editors](#)
-

© *ppgb*, September 4, 1996

Info

One of the fixed buttons in the Object window.

Availability

You can choose this command after choosing one object.

Purpose

To get some information about the selected object.

Behaviour

The information will appear in the Info window.

Links to this page

- [Feedforward neural networks 2. Quick start](#)
- [Feedforward neural networks 3. FFNet versus discriminant classifier](#)
- [Scripting 6.3. Writing to the Info window](#)

© *ppgb*, January 1, 1998



One of the fixed buttons in the Object window.

Availability

You can choose this command after selecting one object of any class.

Behaviour

The Object window copies the selected object, and all the data it contains, to a new object, which will appear at the bottom of the List of Objects.

Example

If you select "Sound hallo" and click 'Copy...', a dialog will appear, which prompts you for a name; after you click OK, a new object will appear in the list, bearing that name.

Links to this page

- [Formulas 8. Data in objects](#)

© *ppgb*, September 4, 1996

Remove

One of the fixed buttons in the Object window.

You can choose this command after selecting one or more objects in the list.

The selected objects will permanently disappear from the list, and the computer memory that they occupied will be freed.

To save your data before removing, choose a command from the Write menu.

Links to this page

- [Editors](#)

© *ppgb*, December 12, 2002

Run script...

A hidden command in the *Control* menu of the Object window. Runs a Praat script.

Usage

This command is hidden because you would normally open a script with Open Praat script..., so that you can run it several times without selecting a file each time.

In scripts, the command *Run script...* is automatically replaced by the script directive **execute**.

Links to this page

- Initialization script

© ppgb, November 30, 2002

New Praat script

A command in the Control menu for creating a new Praat script. It creates a ScriptEditor with an empty script that you can edit, run, and save.

Links to this page

- [Formulas 1.9. Formulas in scripts](#)
- [History mechanism](#)
- [Object window](#)
- [Open Praat script...](#)
- [Scripting 1. My first script](#)

© *ppgb*, November 30, 2002

Open Praat script...

A command in the Control menu for editing an existing Praat script. It creates a ScriptEditor and asks you to select a file. If you click **OK**, the file is read into the ScriptEditor window, and you can run and edit it; if you click **Cancel**, you get an empty script, as with New Praat script.

Links to this page

- [History mechanism](#)
- [Object window](#)
- [Run script...](#)
- [Scripting 1. My first script](#)

© *ppgb*, November 30, 2002

ButtonEditor

An editor for viewing, hiding, showing, removing, and executing the commands in the fixed and dynamic menus of the PRAAT program. To open it, choose **Buttons...** from the **Control** menu of the Object window.

What the button editor shows

The button editor gives a list of:

1. The five fixed buttons.
2. The built-in and added fixed menu commands, lexicographically sorted by window and menu name.
3. The built-in and added action commands, sorted by the names of the selected objects.

Visibility of built-in commands

Most built-in commands are visible by default, but some are hidden by default (see Hidden commands). The button editor shows these commands as "shown" or "hidden", respectively. You can change the visibility of a command by clicking on the blue "shown" or "hidden" text; this text will then be replaced with "HIDDEN" or "SHOWN", with capitals to signal their non-standard settings. These changes will be remembered in the Buttons file across sessions of your program. To return to the standard settings, click the blue "HIDDEN" or "SHOWN" texts again.

Some built-in commands cannot be hidden. They are marked as "unhidable". The most notable example is the **Buttons...** button (a failure to make the **Commands...** command unhidable in Microsoft Word causes some computer viruses to be very hard to remove...).

Added commands

Commands that you have added to the fixed or dynamic menus (probably with Add to fixed menu... or Add to dynamic menu... in the ScriptEditor), are marked as "ADDED". They are remembered in the Buttons file. You can change the availability of these commands by clicking on the blue "ADDED" text, which will then be replaced with "REMOVED". After this, the added command will no longer be remembered in the Buttons file. To make the command available again, click the blue "REMOVED" text again, before leaving the program.

Start-up commands

Commands that were added in an Initialization script (with Add menu command... or Add action command...) are marked as "START-UP". They are *not* remembered in the Buttons file. You can change the visibility of these commands by clicking on the blue "START-UP" text, which will then be replaced with "HIDDEN". This setting *will* be remembered in the Buttons file. To make the command visible again, click the blue "HIDDEN" text again.

Executing commands

The button editor allows you to choose hidden commands without first making them visible in the fixed or dynamic menus.

The editor shows all the executable commands in blue. These include:

1. The fixed **Remove** button, if one or more objects are selected in the List of Objects.
2. The other fixed buttons, if exactly one object is selected.
3. All of the fixed menu commands, hidden or not, and "removed" or not.
4. Those action commands that match the currently selected objects with respect to class and number.

To execute any of these blue commands, just click on it.

Links to this page

- [Objects](#)
- [Scripting 7.2. Scripting an editor from within](#)

© *ppgb*, September 16, 2003

Add menu command...

One of the hidden commands in the Control menu of the Object window. With this command, you add a button to any of the fixed menus in the Object or Picture window.

Arguments

See Add to fixed menu....

Usage

You can use this command in your Initialization script, and you will see it in your Buttons file after you have added a menu button with this command or with the ScriptEditor.

Normally, however, if you want to add a command to a fixed menu, you would use the command Add to fixed menu... of the ScriptEditor instead.

Links to this page

- ButtonEditor
- Hidden commands

© ppgb, May 18, 1997

Add action command...

One of the hidden commands in the Control menu of the Object window. With this command, you add a button to the dynamic menu in the Object window.

Arguments

See Add to dynamic menu....

Usage

You can use this command in your Initialization script, and you will see it in your Buttons file after you have added an action button with this command or with the ScriptEditor.

Normally, however, if you want to add a command to the dynamic menu, you would use the command Add to dynamic menu... of the ScriptEditor instead.

Links to this page

- ButtonEditor
- Hidden commands

© ppgb, May 18, 1997

Quit

One of the commands in the 'Control' menu of the Object window.

Purpose

To leave the program.

Behaviour

All objects not written to a file will be lost. However, file-based objects (like large lexica) will be saved correctly.

Usage

To save your data to a disk file before quitting, choose a command from the Write menu.

© *ppgb*, September 11, 1997

Action commands

The commands in the Dynamic menu of the Object window.

These commands are only available if the right kinds of objects are selected. They are shown in a scrollable list, or in the **Write** menu if they start with "Write to " or "Append to ".

Links to this page

- [ButtonEditor](#)

© *ppgb*, November 30, 2002

Add to dynamic menu...

A command in the File menu of the ScriptEditor.

With this command, you add a button to the dynamic menu in the Object window. This button will only be visible if the specified combination of objects is selected. Clicking the button will invoke the specified Praat script.

Arguments

Class 1

the name of the class of the object to be selected. For instance, if a button should only appear if the user selects a Sound, this would be "Sound".

Number 1

the number of objects of *class1* that have to be selected. For most built-in commands, this number is unspecified (0); e.g., the user can choose **Draw...** or **To Spectrum** regardless of whether she selected 1, 2, 3, or more Sound objects. If the number of selected objects is different from *number1*, the button will be visible but insensitive.

Class 2

the name of the class of the second object to be selected, different from *class1*. Normally the empty string ("").

Number 2

the number of selected objects of *class2*.

Class 3

the name of the class of the third object to be selected, different from *class1* and *class2*. Normally the empty string ("").

Number 3

the number of selected objects of *class3*.

Command

the title of the new command button (or label, or submenu title). To get a separator line instead of a command text (only in a submenu), you specify a unique string that starts with a hyphen ('-'); the ButtonEditor may contain some examples of this. If the command starts with "Write to ", it will be placed in the Write menu.

After command

a button title in the dynamic menu or submenu where you want your new button. If you specify the empty string (""), your button will be put at the bottom. You can specify a push button, a label (subheader), or a cascade button (submenu title) here.

Depth

0 if you want your button in the main menu, 1 if you want it in a submenu.

Script

the full path name of the script to invoke. If you saved the script you are editing, its name will already have been filled in here. If you do not specify a script, you will get a separating label or cascading menu title instead, depending on the *depth* of the following command.

Example

If one object of class Sound is selected, you want a submenu called "Filters" after the **Convolve** button, containing the commands "Autocorrelation" and "Band filter...", separated by a horizontal separator line:

```
Add to dynamic menu... Sound 0 "" 0 "" 0 "Filters -" "Convolve" 0
Add to dynamic menu... Sound 1 "" 0 "" 0 "Autocorrelation" "Filters -"
1 /u/praaats/demo/autocorrelation.praat
Add to dynamic menu... Sound 0 "" 0 "" 0 "-- band filter --"
"Autocorrelation" 1
Add to dynamic menu... Sound 1 "" 0 "" 0 "Band filter..." "-- band
filter --" 1 /u/praaats/demo/bandFilter.praat
```

Note that "Filters -" will be a submenu title, *because* it is followed by subcommands (*depth* 1). Note that *number1* is 1 only for the executable buttons; for the cascade button and the separator line, this number is ignored.

Usage convention

Please adhere to the convention that commands that take arguments, end in three dots (...).

Using this command in a script

To add a dynamic button from a script (perhaps your Initialization script), use the hidden shell command `Add action command...` instead.

Links to this page

- [Add to fixed menu...](#)
- [History mechanism](#)

© ppgb, January 5, 1998

Calculator...

A command in the Goodies submenu of the Control menu of the Object window. Shortcut: Command-U. Choosing this command brings up Praat's calculator.

Links to this page

- [binomialQ](#)
- [Formulas 1.1. Formulas in the calculator](#)
- [OT learning 2.9. Output distributions](#)

© *ppgb*, December 4, 2002

Control menu

The first menu in the Object window. On MacOS X, this menu is called *Praat*.

Links to this page

- [Calculator...](#)
- [Formulas 1.1. Formulas in the calculator](#)
- [Formulas 1.9. Formulas in scripts](#)
- [Goodies](#)
- [ScriptEditor](#)

© ppgb, December 4, 2002

Fixed menu commands

The commands in the fixed menus of the Object window (**Control, New, Read, Help, Goodies, and Preferences**) and the Picture window (**File, Edit, Margins, World, Select, Pen, Font, Help**).

These commands are always clickable (if not hidden) and scriptable (if not added).

Links to this page

- [ButtonEditor](#)

© *ppgb*, May 18, 1997

Hidden commands

Some commands in PRAAT's fixed and dynamic menus are hidden by default. You can still call hidden commands from scripts, run them by clicking on them in a ButtonEditor, or make them visible with the help of the ButtonEditor.

To hide commands that are visible by default, use the ButtonEditor.

What commands are hidden by default?

Commands that are expected to be of very limited use, are hidden by default. Examples are:

1. The commands **Add menu command...**, **Hide menu command...**, **Show menu command...**, **Add action command...**, **Hide action command...**, and **Show action command...** in the **Control** menu of the Object window. These are used in the Buttons file and could be used by an Initialization script as well; in an interactive session, however, the functionality of these commands is part of the ScriptEditor and the ButtonEditor.
2. The command **Read from old Praat picture file...** in the **File** menu of the Picture window. For reading a file format that was in use before May, 1995.
3. In the Praat program, the action **Sound: Write to Sesam file...** Writes a file format in common use in the Netherlands on Vax machines. In the Dutch phonetics departments, the plugs were pulled from the Vaxes in 1994.
4. In the Praat program, the action **Sound: To Cochleagram (edb)...** Needed by one person in 1994. An interesting, but undocumented procedure (De Boer's gamma-tone filter bank plus Meddis & Hewitt's synapse model), which does not create a normally interpretable Cochleagram object.

© ppgb, May 28, 2003

Feedforward neural networks 2. Quick start

You may create the iris example set with the Create iris example... command that you will find under the **Neural nets** option in the **New** menu. Three new objects will appear in the List of Objects: a FFNet, a Categories and a Pattern.

The **Pattern** contains the iris data set in a 150 rows by 4 columns matrix. To guarantee that every cell in the Pattern is in the [0,1] interval, all measurement values were divided by 10. In the **Categories** the three iris species *setosa*, *versicolor*, and *virginica* were categorized with the numbers **1**, **2** and **3**, respectively. Because there are 4 data columns in the Pattern and 3 different iris species in the Categories, the newly created **FFNet** has 4 inputs and 3 outputs. If you have entered a positive number in one of the fields in the form, the FFNet will have this number of units in a *hidden layer*. The name of the newly created FFNet will reflect its topology. If you did opt for the default, 0 hidden units, the FFNet will be named 4-3.

Learning the iris data

The first thing you probably might want to do is to let the **FFNet** learn the association in each pattern-category pair. To do this select all three objects together and choose Learn.... A form will appear, asking you to supply some settings for the learning algorithm. Learning starts after you have clicked the OK-button. Since the example network does not have too many weights that need to be adjusted and the learning data set is very small and computers nowadays are very fast, this will only take a very short time.

Classify

Now, if you are curious how well the FFNet has learned the iris data, you may select the **FFNet** and the **Pattern** together and choose To Categories.... A new **Categories** appears in the **List of Objects** with the name 4-3_iris (if 4-3 was the name of the FFNet and *iris* the name of the Pattern). We have two different Categories in the list of objects, the topmost one has the original categories, the other the categories as were assigned by the FFNet classifier. The obvious thing to do now is to compare the original categories with the assigned categories by making a confusion table. Select the two **Categories** and choose To Confusion and a newly created Confusion appears. Pressing the Info button will show you an info window with, among others, the fraction correct.

You might also want to compare the FFNet classifier with a discriminant classifier.

Create other neural net topologies

With a **Pattern** and a **Categories** selected together, you can for example create a new **FFNet** of a different topology.

Links to this page

- [Feedforward neural networks](#)
-

© *djmw*, April 26, 2004

Spectrum: Get centre of gravity...

A command to query the selected Spectrum object.

If the complex spectrum is given by $S(f)$, where f is the frequency, the *centre of gravity* is given by

$$\int_0^{\infty} f |S(f)|^p df$$

divided by the "energy"

$$\int_0^{\infty} |S(f)|^p df$$

Thus, the centre of gravity is the average of f over the entire frequency domain, weighted by $|S(f)|^p$. For $p = 2$, the weighting is done by the power spectrum, and for $p = 1$, the weighting is done by the absolute spectrum. A value of $p = 2/3$ has been seen as well.

Argument

Power

the quantity p in the formulas above. Common values are 2, 1, or 2/3.

Interpretation

The spectral centre of gravity is a measure for how high the frequencies in a spectrum are on average. For a sine wave with a frequency of 377 Hz, the centre of gravity is 377 Hz. You can easily check this in Praat by creating such a sine wave (Create Sound...), then converting it to a Spectrum (Sound: To Spectrum (fft)), then querying the mean frequency. For a white noise sampled at 22050 Hz, the centre of gravity is 5512.5 Hz, i.e. one half of the Nyquist frequency.

Related measures

The centre of gravity is used in the computation of spectral moments:

- Spectrum: Get central moment...
- Spectrum: Get standard deviation...
- Spectrum: Get skewness...
- Spectrum: Get kurtosis...

© ppgb, March 23, 2002

Write to console

One of the commands in the Write menu.

You can choose this command after selecting one object. The data that it contains, is written to the Console window, in the same format as with the Write to text file... command, except for the first line, which reads something like:

```
Write to console: class Sound, "name hallo"
```

© *ppgb*, September 4, 1996

LongSound: View

A command to view the selected LongSound object in a LongSoundEditor.

© *ppgb*, July 30, 1998

Fast Fourier Transform

An algorithm for fast computation of the Fourier transform of a sampled signal. It involves increasing the number of samples N to the next-highest power of two, and the computation time scales as $N \log N$.

In Praat, the Fast Fourier Transform is used:

1. For the Fourier transform of an entire sound. See [Sound: To Spectrum \(fft\)](#) and [Spectrum: To Sound \(fft\)](#).
2. For the Fourier transform of consecutive frames in a sound. See [Sound: To Spectrogram...](#)
3. For the fast computation of correlations, e.g. in [Sound: To Pitch \(ac\)...](#)

Links to this page

- [FFT](#)
-

© *ppgb*, November 5, 2003

Spectrogram: Formula...

A command for changing the data in all selected Spectrogram objects.

See the Formulas tutorial for examples and explanations.

© *ppgb*, December 6, 2002

Spectrogram: To Spectrum (slice)...

A command to create a Spectrum object from every selected Spectrogram object.

Purpose

to extract the information contained in a Spectrogram at a certain time.

Algorithm

The Spectrum will be constructed from one frame of the Spectrogram, namely the frame whose centre is closed to the *time* argument.

© ppgb, October 3, 1996

Spectrum: To Spectrogram

A command to create a Spectrogram object from every selected Spectrum object.

Purpose

Format conversion.

Behaviour

The Spectrogram will have only one frame (time slice).

Algorithm

The values are computed as the sum of the squares of the real and imaginary parts of the Spectrum.

© *ppgb*, October 3, 1996

Read from Praat picture file...

One of the commands in the File menu of the Picture window.

Purpose

To read a picture that you saved earlier with Write to Praat picture file....

Behaviour

The picture will be drawn across whatever is currently visible in the Picture window.

Usage

With the help of this command, you can transfer a picture from a Unix machine to a Macintosh. Praat for Macintosh can write the picture to an Encapsulated PostScript file with a screen preview.

© *ppgb*, September 8, 1996

Write to Praat picture file...

One of the commands in the File menu of the Picture window.

Purpose

To save a picture in a format that can be read later with Read from Praat picture file....

Usage

With the help of this command, you can transfer a picture from an Indigo to a Macintosh. Praat for Macintosh can write the picture to an Encapsulated PostScript file with a screen preview.

© *ppgb*, September 20, 1996

Undo

One of the commands in the **Edit** menu of the Picture window.

It erases your most recently created drawing, which could have come from a command in the dynamic menu or from one of the drawing commands in the World and Margins menus.

Behaviour

Your settings of the drawing attributes (line type, line width, font, font size, and colour) will not be undone.

The world window will be what it was after the latest-but-one drawing, so that you can use the Margins menu as if the latest drawing had never happened.

© *ppgb*, August 25, 1998

Erase all

A command in the **Edit** menu of the Picture window.

It erases all your drawings.

© *ppgb*, August 25, 1998

Margins

The space around most of your drawings in the Picture window.

World coordinates

With the commands in the **Margins** menu, you draw text, ticks, numbers, or a rectangle, in the margins around the latest drawing that you made, or you draw dotted lines through or text inside this last drawing.

You specify the positions of these things in world coordinates, i.e., in coordinates that refer to the natural coordinate system of your last drawing.

The numbers that you can mark around your drawing also refer to these coordinates. For instance, after drawing a spectrum with **Spectrum: Draw...**, you can draw a dotted line at 2000 Hz or at 60 dB by choosing **One mark bottom...** or **One mark left...** and typing "2000" or "60", respectively.

Usage

The margin commands work with all the drawings that leave margins around themselves, such as **Sound: Draw...**, **Spectrogram: Paint...**, **Polygon: Paint...**, and more. They do not work properly, however, with the commands that draw vocal tract shapes, like **Art & Speaker: Draw...** and **Artword & Speaker: Draw...**, because these can only be drawn correctly into a square viewport.

Limited validity

The margin commands work only on the latest drawing that you made (unless you Undo that drawing).

Margin size

The size of the margins depends on the font size, so be sure that you have the font size of your choice before you make your drawing. You can set the font size with the Font menu.

Links to this page

- Draw inner box
- Formant: Draw tracks...
- Formant: Speckle...
- Logarithmic marks left/right/top/bottom...
- Marks left/right/top/bottom every...
- Marks left/right/top/bottom...
- Matrix: Draw as squares...
- One logarithmic mark left/right/top/bottom...
- One mark left/right/top/bottom...
- Pen menu

- Text left/right/top/bottom...
 - Text...
-

© *ppgb*, April 5, 1997

Draw inner box

One of the commands in the **Margins** menu of the Picture window.

Purpose

To draw a rectangle inside the drawing area, leaving margins on all four sides for drawing text and marks.

Behaviour

The widths of the margins depend on the current font size.

Links to this page

- [Pen menu](#)
-

© *ppgb*, March 30, 1997

Text left/right/top/bottom...

Four of the commands in the **Margins** menu of the Picture window.

Purpose

To write text into the margins around the drawing area.

Behaviour

The text will be centred along the side. Text at the left or right will be turned by 90 degrees and written up and down, respectively.

© *ppgb*, March 30, 1997

Marks left/right/top/bottom every...

Four of the commands in the **Margins** menu of the Picture window.

Purpose

To draw a number of equally spaced marks into the margins around the drawing area.

Arguments

Units

the units, relative to the standard units, for writing the numbers; for example, if you want time in milliseconds instead of seconds (which is always the standard), *Units* should be 0.001.

Distance

the distance between the equally spaced marks that will be drawn, expressed in *Units*; for example, if you want marks every 20 milliseconds, and *Units* is 0.001, this argument should be 20.

Write numbers

if on, real numbers will be written in the margin, expressed in the domain or range of your latest drawing in the horizontal or vertical direction.

Draw ticks

if on, short line pieces will be drawn in the margin.

Draw dotted lines

if on, dotted lines will be drawn through your drawing.

© ppgb, March 30, 1997

One mark left/right/top/bottom...

Four of the commands in the **Margins** menu of the Picture window.

Purpose

To draw one mark into one of the four margins around the drawing area.

Arguments

Position

the x (for top or bottom) or y (for left or right) position of the mark, expressed in the domain or range of your latest drawing in the horizontal or vertical direction.

Write number

if on, a real number equal to 'Position' will be written in the margin, at an x (for top or bottom) or y (for left or right) position equal to *Position*.

Draw tick

if on, a short line piece will be drawn in the margin, at an x (for top or bottom) or y (for left or right) position equal to *Position*.

Draw dotted line

if on, a dotted line will be drawn through your drawing, at an x (for top or bottom) or y (for left or right) position equal to *Position*.

Draw text

if not empty, this text will be drawn in the margin, at an x (for top or bottom) or y (for left or right) position equal to *Position*.

Example

If you draw a Sound to an amplitude range between -1 and 1, choosing **One mark left...** with a position of 0.0 and *Draw dotted line* on, will give you a horizontal mark "0" and a horizontal dotted line at a y position of 0.

© ppgb, March 30, 1997

Marks left/right/top/bottom...

Four of the commands in the **Margins** menu of the Picture window.

Purpose

To draw any number of equally spaced marks into the margins around the drawing area.

Arguments

Number of marks

the number of equally spaced marks (2 or more) that will be drawn; there will always be marks at the beginning and end of the domain or range.

Write numbers

if on, real numbers will be written in the margin, expressed in the domain or range of your latest drawing in the horizontal or vertical direction.

Draw ticks

if on, short line pieces will be drawn in the margin.

Draw dotted lines

if on, dotted lines will be drawn through your drawing.

Example

If you draw a Sound with a domain between 0 and 1 seconds to an amplitude range between -1 and 1, choosing **Marks left...** with a number of 3 and *Draw dotted lines* on, will give you horizontal marks and horizontal dotted lines at -1, 0, and 1; choosing **Marks bottom...** with a number of 6 and *Draw dotted lines* off, will give you vertical marks at 0, 0.2, 0.4, 0.6, 0.8, and 1.

© ppgb, March 30, 1997

Logarithmic marks left/right/top/bottom...

Four of the commands in the **Margins** menu of the Picture window.

Purpose

To draw a specified number of marks per decade into the margins around the drawing area, along a logarithmic axis.

Arguments

Marks per decade

the number of marks that will be drawn for every decade.

Write numbers

if on, real numbers will be written in the margin, expressed in the domain or range of your latest drawing in the horizontal or vertical direction.

Draw ticks

if on, short line pieces will be drawn in the margin.

Draw dotted lines

if on, dotted lines will be drawn through your drawing.

Behaviour

If your vertical logarithmic axis runs from 10 to 100, and *Marks per decade* is 1, marks will only be drawn at 10 and 100;

if *Marks per decade* is 2, marks will be drawn at 10, 30, and 100;

if it is 3, marks will be drawn at 10, 20, 50, and 100;

if it is 4, marks will be drawn at 10, 20, 30, 50, and 100;

if it is 5, marks will be drawn at 10, 20, 30, 50, 70, and 100;

if it is 6, marks will be drawn at 10, 15, 20, 30, 50, 70, and 100;

if it is 7 (the maximum), marks will be drawn at 10, 15, 20, 30, 40, 50, 70, and 100.

One logarithmic mark left/right/top/bottom...

Four of the commands in the **Margins** menu of the Picture window.

Purpose

To draw one mark into one of the four margins around the drawing area, along a logarithmic axis.

Arguments

Position

the x (for top or bottom) or y (for left or right) position of the mark, expressed in the logarithmic domain or range of your latest drawing in the horizontal or vertical direction.

Write number

if on, a real number equal to *Position* will be written in the margin, at an x (for top or bottom) or y (for left or right) position equal to *Position*.

Draw tick

if on, a short line piece will be drawn in the margin, at an x (for top or bottom) or y (for left or right) position equal to *Position*.

Draw dotted line

if on, a dotted line will be drawn through your drawing, at an x (for top or bottom) or y (for left or right) position equal to *Position*.

Draw text

if not empty, this text will be drawn in the margin, at an x (for top or bottom) or y (for left or right) position equal to *Position*.

Example

After you draw a Pitch logarithmically in a range between 100 and 400 Hz, choosing **One logarithmic mark left...** with a position of 200 and *Draw dotted line* on, will give you a horizontal mark "200" and a horizontal dotted line at a y position of 200, which is exactly halfway between 100 and 400 Hz.

© ppgb, March 30, 1997

Axes...

One of the commands in the **Margins** and **World** menus of the Picture window.

Purpose

To view and change the current world coordinates of the horizontal and vertical axes.

Usage

The axes are normally changed by every drawing operation in the dynamic menu, i.e., by object-specific drawing commands with titles like **Draw...** and **Paint...** (the drawing commands in the Picture window, like **Paint rectangle...**, do not change the axes).

You would use the **Axes...** command if your data are not in an object.

Example

The following script would draw a person's vowel triangle:

```
# Put F1 (between 300 and 800 Hz) along the horizontal axis,  
# and F2 (between 600 and 3600 Hz) along the vertical axis.  
Axes... 300 800 600 3600  
# Draw a rectangle inside the current viewport (selected area),  
# with text in the margins, and tick marks in steps of 100 Hz along the  
F1 axis,  
# and in steps of 200 Hz along the F2 axis.  
Draw inner box  
Text top... no Daniël's Dutch vowel triangle  
Text bottom... yes %F_1 (Hz)  
Text left... yes %F_2 (Hz)  
Marks bottom every... 1 100 yes yes yes  
Marks left every... 1 200 yes yes yes  
# Draw large phonetic symbols at the vowel points.  
Text special... 340 Centre 688 Half Times 24 0 u  
Text special... 481 Centre 1195 Half Times 24 0 \o/  
# Etcetera
```

This example would draw the texts "Daniël's Dutch vowel triangle", " F_1 (Hz)", and " F_2 (Hz)" in the margins, and the texts "u" and "ø" at the appropriate positions inside the drawing area.



One of the commands in the **World** menu of the Picture window.

Purpose

To write text inside the drawing area.

Scope

This works with all the drawings that leave margins around themselves.

Arguments

x

horizontal position, expressed in the horizontal domain of your latest drawing.

y

vertical position, expressed in the vertical range or domain of your latest drawing.

Horizontal alignment

determines the horizontal alignment of the text relative to *x*.

Vertical alignment

determines the vertical alignment of the text relative to *y*.

Text

will be drawn in the current font and font size that you set with the Font menu.

Usage

With the **Text...** command, you can use all special symbols and text styles.

© ppgb, March 30, 1997

Select inner viewport...

One of the commands in the **Select** menu of the Picture window.

Purpose

To change the region where the next graphics output will occur.

The viewport

The viewport is the part of the drawing area where your next drawing will occur. It appears on your screen in a selection colour, which is yellowish on Unix machines, but equals the user-settable selection colour on Macintosh.

The *inner* viewport excludes the margins around the drawing area, unlike the *outer viewport* (see *Select outer viewport...*).

Normally, you select the viewport by dragging your mouse across the drawing area. However, you would use this explicit command:

- from a script;
- if you want a viewport that cannot be expressed in halves of an inch.

© ppgb, September 5, 2004

Select outer viewport...

One of the commands in the **Select** menu of the Picture window.

Purpose

To change the region where the next graphics output will occur.

The viewport

The viewport is the part of the drawing area where your next drawing will occur. It appears on your screen in a selection colour, which is yellowish on Unix machines, but equals the user-settable selection colour on Macintosh.

The *outer* viewport includes the margins around the drawing area, unlike the *inner viewport* (see *Select inner viewport...*).

Normally, you select the viewport by dragging your mouse across the drawing area. However, you would use this explicit command:

- from a script;
- if you want a viewport that cannot be expressed in halves of an inch.

© ppgb, September 5, 2004

Viewport text...

One of the commands in the **Select** menu of the Picture window.

Purpose

To write text inside the viewport, at nine different places, with a rotation between 0 to 360 degrees.

Arguments:

Horizontal alignment

determines the horizontal alignment of the text:

- o *Left* means pushed against the left edge of the viewport;
- o *Right* means pushed against the right edge of the viewport;
- o *Centre* means horizontally centred in the viewport.

Vertical alignment

determines the vertical alignment of the text:

- o *Top* means pushed against the top of the viewport;
- o *Bottom* means pushed against the bottom of the viewport;
- o *Half* means vertically centred in the viewport.

Text

will be drawn in the current font and font size that you set with the Font menu.

Behaviour

For rotated text, the alignment arguments will not only determine the position inside the viewport, but also the alignment in the rotated coordinate system. This gives surprises now and then; so, if you want several rotated texts that align with each other, you should do this by varying the viewport, not the alignment.

Usage

You can use all special symbols and text styles.

© ppgb, March 30, 1997

Pen menu

One of the menus of the Picture window.

Purpose

To choose the default line type and colour to be used in subsequent drawing of lines and painted areas.

Behaviour

The line type used by Draw inner box (plain), and the line type of the dotted lines in the **Mark...** commands will not be affected.

The commands in the Margins menu will always draw in black.

© ppgb, October 6, 1996

Font menu

One of the menus of the Picture window.

It allows you to choose the default font of the text to be used in subsequent drawing, and its size (character height).

Sizes

You can choose any of the sizes 10, 12, 14, 18, or 24 directly from this menu, or fill in any other size in the **Font size...** form.

Unix:

the font size will be rounded to the nearest size available on Xwindows, which is one from 10, 12, 14, 18, or 24 points; PostScript-printing a picture where you specified a font size of 100, however, will still give the correct 100-point character height.

Macintosh and Windows:

all sizes are drawn correctly (what you see on the screen is what you get on your printer).

The widths of the margins depend on the current font size, so if you want to change the font size, do so before making your drawing.

Fonts

With these commands, you set the font in which subsequent text will be drawn: Times, Helvetica, New Century Schoolbook, Palatino, or Courier.

You can mix the Symbol and IPA alphabets with the normal Roman alphabets and use sequences of backslash + digraph for special symbols (see also phonetic symbols).

For instance, you can get an ϵ by typing `\e`, or a β by typing `\ss`; you can get an € by typing `\ep`, or a  , which is a turned c, by typing `\ct`.

In a PostScript preview on the Indigo (double-click on a PostScript file), New Century Schoolbook will be replaced by Courier.

If you print to a PostScript printer, all fonts will be correct.

Styles

You can use all graphical text styles in the Picture window.

Links to this page

- [Margins](#)
- [Text...](#)
- [Viewport text...](#)

© *ppgb*, February 11, 2004

Formant: Draw tracks...

A command to draw the selected Formant objects to the Picture window.

Behaviour

Every formant value is drawn as one or two short line segments, connected, if possible, with a line segment of the corresponding formant values in the adjacent frames.

Arguments

From time (s), To time (s)

the time domain of the drawing. If *To time* is not greater than *From time*, the entire formant contour is drawn.

Maximum frequency (Hz)

the height of the y axis. For speech, 5000 Hz is a usual value.

Garnish

determines whether axes, numbers, and texts ("Time", "Formant frequency") will be drawn in the margins around the picture. Turn this button off if you prefer to garnish you picture by yourself with the Margins menu.

Usage

Unlike Formant: Speckle..., this procedure assumes that e.g. the second formant in frame i has something to do with the second formant in frame $i+1$. To make this assumption more plausible, use Formant: Track... first.

© ppgb, March 21, 1998

Matrix: Draw as squares...

A command to draw a Matrix object into the Picture window.

Arguments

Xmin, Xmax

the windowing domain in the x direction. Elements outside will not be drawn. *Autowindowing*: if ($Xmin \geq Xmax$), the entire x domain [x_{min}, x_{max}] of the Matrix is used.

Ymin, Ymax

the windowing domain in the y direction. Elements outside will not be drawn. *Autowindowing*: if ($Ymin \geq Ymax$), the entire y domain [y_{min}, y_{max}] of the Matrix is used.

Garnish

determines whether axes are drawn around the picture. Turn this button off if you prefer to garnish your picture by yourself with the Margins menu.

Behaviour

For every element of the Matrix inside the specified windowing domain, an opaque white or black rectangle is painted (white if the value of the element is positive, black if it is negative), surrounded by a thin black box. The *area* of the rectangle is proportional to the value of the element.

Trick

If you prefer the *sides* of the rectangle (instead of the area) to be proportional to the value of the element, you can use the formula "`self^2`" before drawing (see Matrix: Formula...).

© ppgb, March 19, 1998

Matrix: Paint cells...

A command to draw the contents of a Matrix to the Picture window.

Every cell of the matrix is drawn as a rectangle filled with a grey value between white (if the content of the cell is small) and black (if the content is large).

Links to this page

- [Formulas 1.7. Formulas for creation](#)
-

© *ppgb*, December 4, 2002

Phonetic symbols

To draw phonetic symbols in the Picture window or in the TextGridEditor, make sure that you have installed the SIL Doulos IPA 1989 font, e.g. from www.praat.org. You can then use backslash sequences as described in:

- Phonetic symbols: consonants
- Phonetic symbols: vowels
- Phonetic symbols: diacritics

Links to this page

- [Font menu](#)
 - [Special symbols](#)
-

© *ppgb*, February 11, 2004

Phonetic symbols: consonants

To draw phonetic symbols for consonants in the Picture window or in the TextGridEditor, make sure that you have installed the SIL Doulos IPA 1989 font, e.g. from www.praat.org. You can then use the backslash sequences in the following table.

[sorry, no pictures yet in the web version of this manual]

Other consonant symbols:

 \l~ (*l with tilde*): velarized *l*

 \hj (*heng with hooktop*): the Swedish rounded post-alveolar & velar fricative

How to remember the codes

For most of the codes, the first letter tells you the most similar letter of the English alphabet. The second letter can be *t* (*turned*), *c* (*capital* or *curled*), *s* (*script*), *-* (*barred*), *l* (*with leg*), *i* (*inverted*), or *j* (*left tail*). Some phonetic symbols are similar to Greek letters but have special phonetic (*f*) versions with serifs (, , ) or are otherwise slightly different (). The codes for  (*engma*),  (*eth*),  (*esh*), and  (*yogh*) are traditional alternative spellings. The retroflexes have a period in the second place, because an alternative traditional spelling is to write a dot under them. The code for  is an abbreviation for *fishhook*.

Links to this page

- [Phonetic symbols](#)

© ppgb, February 11, 2004

Phonetic symbols: diacritics

To draw phonetic diacritical symbols in the Picture window or in the TextGridEditor, make sure that you have installed the SIL Doulos IPA 1989 font, e.g. from www.praat.org. You can then use the backslash sequences in the following list.

In line:

[IMAGE] \:f the phonetic length sign
[IMAGE] \|f the phonetic stroke
t[IMAGE] t\cn (*corner*): unreleased plosive

Understrikes:

n[IMAGE] n|v (*strokeunder*): syllabic consonant
b[IMAGE] b\0v (*ringunder*): voiceless (e.g. lenis voiceless plosive, voiceless nasal or approximant)
o[IMAGE] o\Tv (*lowering*): lowered vowel; or turns a fricative into an approximant
o[IMAGE] o\T^ (*raising*): raised vowel; or turns an approximant into a fricative
e[IMAGE] e\~v (*minusunder*): backed
o[IMAGE] o\+v (*plusunder*): fronted
o[IMAGE] o\;v (*diareisunder*): breathy voice
o[IMAGE] o\~v (*tildeunder*): creaky voice
t[IMAGE] t\Nv (*bridgeunder*): dental (as opposed to alveolar)
e[IMAGE] e\3v (*halfringright*): slightly rounded
u[IMAGE] u\cv (*halfringleft*): slightly unrounded

Overstrikes:

[IMAGE] [IMAGE] \gf\0^ (*ringover*): voiceless
ɛ [IMAGE] \ep\'^ (*acuteover*): high tone
ɛ [IMAGE] \ep\'^ (*graveover*): low tone
ɛ [IMAGE] \ep\~^ (*tildeover*): nasalized
ɛ [IMAGE] \ep\v^ (*caronover*, *hac* [IMAGE] ek, *wedge*): rising tone
ɛ [IMAGE] \ep\^^ (*circumover*): falling tone
o [IMAGE] o\:^ (*diareisover*): centralized
k[IMAGE] p t[IMAGE] s k\lip (*ligature*): simultaneous articulation, or single segment

Links to this page

- [Phonetic symbols](#)
-

© *ppgb*, February 11, 2004

Phonetic symbols: vowels

To draw phonetic symbols for vowels in the Picture window or in the TextGridEditor, make sure that you have installed the SIL Doulos IPA 1989 font, e.g. from www.praat.org. You can then use the backslash sequences in the following table.

[sorry, no pictures yet in the web version of this manual]

Other vowel symbols are:

[IMAGE] \sɹ (*schwa with right hook*): rhotacized schwa

[IMAGE] \ɛf (*phonetic epsilon*, not recommended)

How to remember the codes

For most of the codes, the first letter tells you the most similar letter of the English alphabet. The second letter can be *t* (*turned*), *c* (*capital*), *s* (*script*), *r* (*reversed*), *-* (*barred*), or */* (*slashed*). One symbol (€) is a Greek letter. The codes for [IMAGE], [IMAGE], and [IMAGE] are abbreviations for *schwa*, *ram's horn*, and *horseshoe*.

Links to this page

- [Phonetic symbols](#)

© ppgb, February 11, 2004

Pitch: Draw...

A command for drawing the selected Pitch objects into the Picture window.

Arguments

From time (seconds), *To time* (seconds)

the time domain along the horizontal axis. If these are both zero, the time domain of the **Pitch** itself is taken (autowindowing).

Minimum frequency (Hz), *Maximum frequency* (Hz)

the frequency range along the vertical axis. *MaximumFrequency* must be greater than *minimumFrequency*.

Behaviour

In unvoiced frames, nothing will be drawn.

In voiced frames, the pitch frequency associated with the frame is thought to represent the time midpoint of the frame, but frequencies will be drawn at all time points in the frame, as follows:

- If two adjacent frames are both voiced, the frequency of the time points between the midpoints of the frames is linearly interpolated from both midpoints.
- If a voiced frame is adjacent to another voiced frame on one side, and to a voiceless frame on the other side, the frequencies in the half-frame on the unvoiced side will be linearly extrapolated from the midpoints of the two voiced frames involved.
- If a voiced frame is adjacent to two unvoiced frames, a horizontal line segment will be drawn at the frequency of the midpoint.

© ppgb, September 10, 1996

PointProcess: Draw...

A command to draw every selected PointProcess into the Picture window.

© *ppgb*, December 12, 2002

Click

One of the ways to control Editors.

How to click

1. Position the mouse above the object that you want to click.
2. Press and release the (left) mouse button.

See also Shift-click.

Usage in the Praat program

Clicking on an object is used for selecting this object while deselecting all previously selected objects; clicking is also used for moving a cursor hair.

Links to this page

- [ManipulationEditor](#)
- [PitchTierEditor](#)
- [PointEditor](#)
- [TextGridEditor](#)
- [Time selection](#)

© *ppgb*, September 13, 1996

Drag

Dragging is one of the ways to control Editors.

How to drag

1. Position the mouse above the object that you want to drag.
2. Press the (left) mouse button.
3. Keeping the mouse button pressed, move the mouse across the window. A shadow of the object will follow.
4. Release the mouse button when it is above the location where you want your object to be moved (the *drop site*). If the drop site makes any sense, the object will move there.

See also Shift-drag.

Usage in the Praat program

Dragging is used for manipulating the time and value of one or more marks, targets, or boundaries:

- ManipulationEditor
- TextGridEditor
- Time selection

© ppgb, September 13, 1996

Shift-drag

Shift-dragging is one of the ways to control Editors.

How to Shift-drag

1. Position the mouse above any of the objects that you want to drag (the objects were probably selected first).
2. Press a Shift key.
3. Press the (left) mouse button.
4. Keeping the mouse button pressed, move the mouse across the window. A shadow of the objects will follow.
5. Release the mouse button when it is above the location where you want your objects to be moved. If this *drop site* makes any sense, the objects will move there.

Usage in the Praat program

While plain dragging is used for moving objects that were selected first by clicking, **Shift-dragging** is used for manipulating the times and values of more marks, targets, or boundaries simultaneously:

- TextGridEditor
-

© ppgb, August 23, 1998

Spectrum: Get standard deviation...

A command to query the selected Spectrum object.

The standard deviation of a spectrum is the square root of the second central moment of this spectrum. See Spectrum: Get central moment....

Argument

Power

the quantity p in the formula for the centre of gravity and the second central moment. Common values are 2, 1, or $2/3$.

Interpretation

The standard deviation is a measure for how much the frequencies in a spectrum can deviate from the *centre of gravity*. For a sine wave, the standard deviation is zero. For a white noise, the standard deviation is the Nyquist frequency divided by $\sqrt{12}$.

Related measures

- Spectrum: Get centre of gravity...
- Spectrum: Get central moment...
- Spectrum: Get skewness...
- Spectrum: Get kurtosis...

© ppgb, March 23, 2002

Spectrum: Get skewness...

A command to query the selected Spectrum object.

The (normalized) skewness of a spectrum is the third central moment of this spectrum, divided by the 1.5 power of the second central moment. See Spectrum: Get central moment...

Argument

Power

the quantity p in the formula for the centre of gravity and the second and third central moment.
Common values are 2, 1, or $2/3$.

Interpretation

The skewness is a measure for how much the shape of the spectrum below the *centre of gravity* is different from the shape above the mean frequency. For a white noise, the skewness is zero.

Related measures

- Spectrum: Get centre of gravity...
- Spectrum: Get central moment...
- Spectrum: Get standard deviation...
- Spectrum: Get kurtosis...

© ppgb, March 23, 2002

Spectrum: Get kurtosis...

A command to query the selected Spectrum object.

The (normalized) kurtosis of a spectrum is the fourth central moment of this spectrum, divided by the square of the second central moment, minus 3. See Spectrum: Get central moment...

Argument

Power

the quantity p in the formula for the centre of gravity and the second and fourth central moment. Common values are 2, 1, or $2/3$.

Interpretation

The kurtosis is a measure for how much the shape of the spectrum around the *centre of gravity* is different from a Gaussian shape. For a white noise, the kurtosis is $\sqrt{6}/5$.

Related measures

- Spectrum: Get centre of gravity...
- Spectrum: Get central moment...
- Spectrum: Get standard deviation...
- Spectrum: Get skewness...

© ppgb, March 23, 2002

Spectrum: Get central moment...

A command to query the selected Spectrum object.

If the complex spectrum is given by $S(f)$, the n th central spectral moment is given by

$$\int_0^{\infty} (f - f_c)^n |S(f)|^p df$$

divided by the "energy"

$$\int_0^{\infty} |S(f)|^p df$$

In this formula, f_c is the spectral centre of gravity (see Spectrum: Get centre of gravity...). Thus, the n th central moment is the average of $(f - f_c)^n$ over the entire frequency domain, weighted by $|S(f)|^p$. For $p = 2$, the weighting is done by the power spectrum, and for $p = 1$, the weighting is done by the absolute spectrum. A value of $p = 2/3$ has been seen as well.

Arguments

Moment

the number n in the formulas above. A number of 3 gives you the third central spectral moment. It is not impossible to ask for fractional moments, e.g. $n = 1.5$.

Power

the quantity p in the formula above. Common values are 2, 1, or 2/3.

Usage

For $n = 1$, the central moment should be zero, since the centre of gravity f_c is computed with the same p . For $n = 2$, you get the variance of the frequencies in the spectrum; the standard deviation of the frequency is the square root of this. For $n = 3$, you get the non-normalized spectral skewness; to normalize it, you can divide by the 1.5 power of the second moment. For $n = 4$, you get the non-normalized spectral kurtosis; to normalize it, you can divide by the square of the second moment and subtract 3. Praat can directly give you the quantities mentioned here:

- Spectrum: Get centre of gravity...
- Spectrum: Get standard deviation...
- Spectrum: Get skewness...
- Spectrum: Get kurtosis...

© *ppgb*, March 23, 2002

Formula...

See Matrix: Formula...

Links to this page

- Dissimilarity
- Formulas
- Spectrum

© *ppgb*, March 19, 1998

Spectrum: To Ltas (1-to-1)

A command for converting each selected Spectrum object into an Ltas object without loss of frequency resolution.

Algorithm

Each band b_i in the Ltas is computed from a single frequency sample s_i in the Spectrum as follows:

$$b_i = 2 ((\text{Re}(s_i))^2 + (\text{Im}(s_i))^2) / 4.0 \cdot 10^{-10}$$

If the original Spectrum is expressible in Pa / Hz (sound pressure in air), the Ltas values are in "dB/Hz" relative to the auditory threshold at 1000 Hz ($2 \cdot 10^{-5}$ Pa).

Links to this page

- [Ltas: Get frequency of maximum...](#)
- [Ltas: Get frequency of minimum...](#)
- [Ltas: Get maximum...](#)
- [Ltas: Get minimum...](#)

© ppgb, March 27, 1998

LPC: To Spectrum (slice)...

You can choose this command after selecting 1 or more LPC objects.

Settings

Time (seconds)

the time at which the Spectrum must be calculated.

Minimum frequency resolution (Hz)

successive frequencies in the Spectrum will be maximally this distance apart.

Bandwidth reduction (Hz)

formants with small bandwidths show up very well as peaks in the spectrum because the poles lie close to the contour along which the spectrum is computed (the unit circle in the z-plane). Peak enhancement can be realized by computing the spectrum in the z-plane along a contour of radius $r = \exp(-\pi \cdot \text{BandwidthReduction} / \text{samplingFrequency})$. This technique is also called off-axis spectrum computation.

De-emphasis frequency (Hz)

Performs de-emphasis when frequency is in the interval (0, Nyquist frequency)

Algorithm

The Spectrum at time t will be calculated from the *nearest* LPC_Frame according to:

$$\text{Spectrum}(f) = \sqrt{\text{gain} \cdot T / df} / (1 + \sum_{k=1..numberOfCoefficients} a_k z^{-k}),$$

where T is the sampling period and $z = \exp(-2\pi i f T)$ and df is the distance in Hz between two successive components in the Spectrum.

1. Allocate a large enough buffer[1..*nfft*] to perform an fft analysis.
2. Make the first value of the buffer 1 and copy the prediction coefficients \mathbf{a} into the buffer. This results in buffer values: $(1, a_1, \dots, a_{\text{numberOfCoefficients}}, 0, \dots, 0)$.
3. If *deEmphasisFrequency* is in the interval (0, nyquistFrequency) then "multiply" the buffer with $(1 - b z^{-1})$, where $b = \exp(-\pi \text{deEmphasisFrequency} T)$. This results in buffer values: $(1, a_1 - b, a_2 - b \cdot a_1, \dots, a_{\text{numberOfCoefficients}} - b \cdot a_{\text{numberOfCoefficients}-1}, -b \cdot a_{\text{numberOfCoefficients}}, 0, \dots, 0)$. Note that the number of values in the buffer that differ from 0 has increased by one.
4. If *bandwidthReduction* > 0 then multiply corresponding values in the buffer by g^{i-1} where $g = \exp(2\pi \text{bandwidthReduction} T / \text{nfft})$, and i is the position index in the buffer. i runs from 1 to $\text{numberOfCoefficients} + 1 + t$, where t equals 1 when de-emphasis was performed, else 0.
5. Calculate the fft spectrum of the buffer with the coefficients. This results in complex amplitudes $(a_j, b_j), j=1..nfft/2+1$.
6. Calculate the LPC Spectrum by taking the inverse of the fft spectrum, i.e., each complex amplitude becomes $(a_j, b_j)^{-1} = (a_j - b_j) / (a_j^2 + b_j^2)$
7. Multiply all values with the scale factor $\sqrt{\text{gain} \cdot T / df}$.

Links to this page

- [LPC: To Spectrogram...](#)
-

© *djmw*, April 7, 2004

Polynomial: To Spectrum...

A command to compute the spectrum of the selected Polynomial objects.

Arguments

Nyquist frequency (Hz)

defines the highest frequency in the spectrum. The lowest frequency of the spectrum will be 0 Hz.

Number of frequencies

defines the number of frequencies in the spectrum.

Algorithm

We calculate the spectrum by evaluating the polynomial at regularly spaced points z_k on the upper half of a circle with radius $r = 1$ in the complex plane. The upperhalf of the unit circle, where $k \cdot \varphi$ is in the interval $[0, \pi]$, will be mapped to frequencies $[0, \text{Nyquist frequency}]$ in the spectrum.

The complex values z_k ($k=1..\text{numberOfFrequencies}$) are defined as:

$$z_k = r e^{i k \varphi} \quad , \text{ where,}$$

$$\varphi = \pi / (\text{numberOfFrequencies} - 1) \text{ and } r = 1.$$

© djmw, June 16, 1999

Spectra: Multiply

Returns a new Spectrum object that is the product of the two selected Spectrum objects.

© *djmw*, October 23, 2003

Spectrum: Conjugate

Reverses the sign of the complex part of the selected Spectrum object(s).

© *djmw*, October 23, 2003

Spectrum: Formula...

A command for changing the data in all selected Spectrum objects.

See the Formulas tutorial for examples and explanations.

© *ppgb*, December 6, 2002

Frequency selection

The way to select a frequency domain in the SpectrumEditor. This works completely analogously to the time selection in other editors.

© *ppgb*, April 2, 2001

Pitch: To PointProcess

A command that uses a Pitch object to generate a PointProcess.

Purpose

to interpret an acoustic periodicity contour as the frequency of an underlying point process (such as the sequence of glottal closures in vocal-fold vibration).

Algorithm

1. A PitchTier is created with Pitch: To PitchTier.
2. The algorithm of PitchTier: To PointProcess generates points along the entire time domain of the PitchTier.
3. The PitchTier is removed (it never appeared in the List of Objects).
4. The voiced/unvoiced information in the Pitch is used to remove all points that lie within voiceless frames.

© *ppgb*, September 17, 1996

Pitch: To PitchTier

A command that converts a Pitch object into a PitchTier object.

Algorithm

The **PitchTier** object will contain as many points as there were voiced frames in the **Pitch**.

The *time* of each point is the time associated with the centre of the corresponding *frame* of the **Pitch** contour. The *frequency* of the point is the pitch frequency associated in this frame with the current path through the candidates.

Links to this page

- [Manipulation](#)
- [Pitch: To PointProcess](#)

© ppgb, September 15, 1996

Pitch & TextTier: To PitchTier...

A command that creates a PitchTier object from one selected Pitch and one selected TextTier object.

Purpose

to return the frequencies in the Pitch contour at the times specified by the TextTier.

Argument

Check voicing (standard: on)

determines whether, if the time of a mark is not within a voiced frame, you will get a message like "No periodicity at time *xxx*.", and no PitchTier is created. If this button is off, the resulting pitch frequency will be 0.0 Hz.

Normal behaviour

For all the times of the marks in the TextTier, a pitch frequency is computed from the information in the Pitch, by linear interpolation.

All the resulting time-frequency pairs are put in a new PitchTier object.

The time domain of the resulting PitchTier is a union of the domains of the original Pitch and TextTier functions.

© *ppgb*, September 16, 2003

FAQ: Pitch analysis

Question: what algorithm is used for pitch analysis?

Answer: see Sound: To Pitch (ac)... The 1993 article is downloadable from <http://www.fon.hum.uva.nl/paul/>

Question: why do I get different results for the maximum pitch if...?

If you select a Sound and choose Sound: To Pitch..., the time step will usually be 0.01 seconds. The resulting Pitch object will have values for times that are 0.01 seconds apart. If you then click Info or choose *Get maximum pitch* from the Query submenu, the result is based on those time points. By contrast, if you choose *Get maximum pitch* from the *Pitch* menu in the SoundEditor window, the result will be based on the visible points, of which there tend to be a hundred in the visible window. These different time spacings will lead to slightly different pitch contours.

If you choose *Move cursor to maximum pitch*, then choose *Get pitch* from the *Pitch* menu, the result will be different again. This is because *Get maximum pitch* can do a parabolic interpolation around the maximum, whereas *Get pitch*, not realizing that the cursor is at a maximum, does a stupid linear interpolation, which tends to lead to lower values.

Links to this page

- [FAQ \(Frequently Asked Questions\)](#)

© ppgb, December 19, 2002

Get frame number from time...

A command that becomes available in the **Query** menu if you select a sound-analysis object that is a function of time and that is evenly sampled in time (Pitch, Formant, Intensity, Harmonicity).

The Info window will tell you the frame number belonging to the time that you specify. The result is presented as a real number.

Setting

Time (seconds)

the time for which you want to know the frame number.

Example

If the Pitch object has a time step of 10 ms, and the first frame is centred around 18 ms, the frame number associated with a time of 0.1 seconds is 9.2.

Scripting

You can use this command to put the nearest frame centre into a script variable:

```
select Pitch hallo
frame = Get frame from time... 0.1
nearestFrame = round (frame)
```

In this case, the value will not be written into the Info window. To round down or up, use

```
leftFrame = floor (frame)
rightFrame = ceiling (frame)
```

Algorithm

the result is

$$1 + (time - t_1) / \Delta t$$

where t_1 is the time associated with the centre of the first frame, and Δt is the time step.

Details for hackers

If you select one of the above objects and click Inspect, you can see how the relation between frame numbers and times is stored in the object: t_1 is the **x1** attribute, and Δt is the **dx** attribute.

© *ppgb*, May 5, 2004

Get number of frames

A command that becomes available in the **Query** menu if you select a sound-analysis object that is a function of time and that is evenly sampled in time (Pitch, Formant, Intensity, Harmonicity).

The Info window will tell you the total number of time frames in the object.

Details for hackers

If you select one of the above objects and click Inspect, you can see how the number of frames is stored in the object: it is the **nx** attribute.

© *ppgb*, April 20, 2004

Get time from frame number...

A command that becomes available in the **Query** menu if you select a sound-analysis object that is a function of time and that is evenly sampled in time (Pitch, Formant, Intensity, Harmonicity).

The Info window will tell you the time associated with the frame number that you specify.

Setting

Frame number

the frame number whose time is sought.

Algorithm

the result is

$$t_1 + (\text{frame_number} - 1) \cdot \Delta t$$

where t_1 is the time associated with the centre of the first frame, and Δt is the time step.

Details for hackers

If you select one of the above objects and click Inspect, you can see how the relation between frame numbers and times is stored in the object: t_1 is the **x1** attribute, and Δt is the **dx** attribute.

© ppgb, April 20, 2004

Get time step

A command that becomes available in the **Query** menu if you select a sound-analysis object that is a function of time and that is evenly sampled in time (Pitch, Formant, Intensity, Harmonicity).

The Info window will tell you the time difference between consecutive frames, e.g. the time difference between consecutive formant circles in the sound editor window.

Details for hackers

If you select one of the above objects and click Inspect, you can see how the time step is stored in the object: it is the **dx** attribute.

© *ppgb*, April 20, 2004

Pitch: Interpolate

A command that converts every selected Pitch object.

© *ppgb, August 11, 1999*

Pitch: Smooth...

A command that converts every selected Pitch object.

© *ppgb*, August 11, 1999

PitchTier: Get mean (curve)...

A query to the selected PitchTier object.

Return value

the mean of the curve within a specified time window.

Attributes

From time (s), *To time* (s)

the time window. Values outside this window are ignored. If *To time* is not greater than *From time*, the entire time domain of the tier is considered.

Algorithm

The curve consists of a sequence of line segments. The contribution of the line segment from (t_1, f_1) to (t_2, f_2) to the area under the curve is

$$1/2 (f_1 + f_2) (t_2 - t_1)$$

The mean is the sum of these values divided by *toTime* - *fromTime*.

For a PitchTier that was created from a Pitch object, this command gives the same result as **Get mean...** for the original Pitch object (but remember that the median, as available for Pitch objects, is more robust).

To get the mean in the entire curve, i.e. weighted by the durations of the line pieces, Use PitchTier: Get mean (points)... instead.

© ppgb, August 21, 2001

PitchTier: Get standard deviation (points)...

A query to the selected PitchTier object.

Return value

the standard deviation in the points within a specified time window.

Attributes

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the tier is considered.

For a PitchTier that was created from a Pitch object, this command gives the same result as **Get standard deviation....** for the original Pitch object (but remember that variation measures based on quantiles, as available for Pitch objects, are more robust).

To get the standard deviation in the entire curve, i.e. weighted by the durations of the line pieces, Use PitchTier: Get standard deviation (curve)... instead.

© ppgb, August 21, 2001

Query menu

One of the menus in most editors.

Links to this page

- [Get first formant](#)
- [Get pitch](#)
- [Get second formant](#)

© *ppgb*, April 17, 2001

Categories: Difference

A command to compute the difference between two selected Categories objects.

Behaviour

Each element in the first object is compared with the corresponding object in the second object according to its compare method. The number of different *categories* will be shown in the Info window.

© djmw, September 18, 1996

undefined

When you give a query command for a numeric value, Praat sometimes writes the numeric value **--undefined--** into the Info window (two hyphens at both sides of the word). This happens if the value you ask for is not defined, as in the following examples:

- You select a Sound with a finishing time of 1.0 seconds and ask for the minimum point in the wave form between 1.5 and 2.0 seconds (with the query command **Get minimum...**).
- You ask for a pitch value in a voiceless part of the sound (select a **Pitch**, then choose **Get value at time...**).
- You type into the Calculator the following formula: 10^{400} .

Usage in a script

In a Praat script, this value is simply represented as "undefined". You use it to test whether a query command returned a valid number:

```
select Pitch hallo
meanPitch = Get mean... 0.1 0.2 Hertz Parabolic
if meanPitch = undefined
  # Take some exceptional action.
else
  # Take the normal action.
endif
```

Details for hackers

In text files, this value is written as **--undefined--**. In binary files, it is written as a big-endian IEEE positive infinity. In memory, it is the ANSI-C constant `HUGE_VAL`, which equals infinity on IEEE machines.

Links to this page

- Formant: Get bandwidth at time...
- Formant: Get time of maximum...
- Formant: Get time of minimum...
- Formant: Get value at time...
- Formulas 3. Constants
- Get high index from time...
- Get low index from time...
- Get nearest index from time...
- Harmonicity: Get mean...
- PointProcess: Get interval...
- PointProcess: Get jitter (ddp)...

- Sound: Get nearest zero crossing...
 - Sound: Get standard deviation...
 - Sound: Get value at sample number...
-

© *ppgb*, April 14, 2004

Script for listing F0 statistics

"I need to split the wave into 50 msec sections, and then for each of those sections get the F0 statistics. That is for each 50 msec section of speech I want to get the average F0, min, max, and st dev."

First you create the complete pitch contour, i.e., you select the Sound and choose To Pitch.... You can then use the commands from the Query submenu in a loop:

```
startingTime = Get starting time
finishingTime = Get finishing time
numberOfTimeSteps = (finishingTime - startingTime) / 0.05
echo tmin tmax mean fmin fmax stdev
for step to numberOfTimeSteps
  tmin = startingTime + (step - 1) * 0.05
  tmax = tmin + 0.05
  mean = Get mean... tmin tmax Hertz
  minimum = Get minimum... tmin tmax Hertz Parabolic
  maximum = Get maximum... tmin tmax Hertz Parabolic
  stdev = Get standard deviation... tmin tmax Hertz
  printline 'tmin:6' 'tmax:6' 'mean:2'
  ... 'minimum:2' 'maximum:2' 'stdev:2'
endfor
```

Notes

One should not cut the sound up into pieces of 50 ms and then do **To Pitch...** on each of them, because Praat will not compute F0 values in the first or last 20 ms (or so) of each piece. This is because the analysis requires a window of 40 ms (or so) for every pitch frame. Instead, one typically does the analysis on the whole sound, then queries the resulting large Pitch object. In that way, the information loss of windowing only affects the two 20 ms edges of the whole sound.

The example writes lines to the Info window. If you want to write to a file instead, you start with something like

```
filedelete ~/results/out.txt
```

and add lines in the following way:

```
fileappend ~/results/out.txt 'tmin:6' 'tmax:6' 'mean:2'
... 'minimum:2' 'maximum:2' 'stdev:2' 'newline$'
```

Links to this page

- [Scripting examples](#)
-

© *ppgb*, October 14, 2002

Script for listing time\--F0\--intensity

I want a list of pitch and intensity values at the same times.

Since Sound: To Pitch... and Sound: To Intensity... do not give values at the same times, you create separate pitch and intensity contours with high time resolution, then interpolate. In the following example, you get pitch and intensity values at steps of 0.01 seconds by interpolating curves that have a time resolution of 0.001 seconds.

```
sound = selected ("Sound")
tmin = Get starting time
tmax = Get finishing time
To Pitch... 0.001 75 300
Rename... pitch
select sound
To Intensity... 75 0.001
Rename... intensity
echo Here are the results:
for i to (tmax-tmin)/0.01
  time = tmin + i * 0.01
  select Pitch pitch
  pitch = Get value at time... time Hertz Linear
  select Intensity intensity
  intensity = Get value at time... time Cubic
  printline 'time:2' 'pitch:3' 'intensity:3'
endfor
```

Links to this page

- [Scripting examples](#)

© ppgb, February 22, 2004

Childers (1978)

Modern spectrum analysis, IEEE Press.

The Burg algorithm for linear prediction coefficients is described on pages 252-255.

Links to this page

- [Sound: To Formant \(burg\)...](#)
-

© *ppgb*, May 15, 2003

Press et al. (1992)

W.H. Press, S.A. Teukolsky, W.T. Vetterling & B.P. Flannery (1992), *Numerical Recipes in C: the art of scientific computing*, Second Edition, Cambridge University Press.

Links to this page

- [FFNet & Pattern & Categories: Learn...](#)
- [LPC: To Formant](#)
- [Polynomial: To Roots](#)
- [Sound: To Formant \(burg\)...](#)
- [Vector peak interpolation](#)

© djmw, January 14, 1998

FAQ: Formant analysis

Problem: I get different formant values if I choose to analyse 3 formants than if I choose to analyse 4 formants.

Solution: the "number of formants" in formant analysis determines the number of peaks with which the *entire* spectrum is modelled. For an average female voice, you should choose to analyse 5 formants in the region up to 5500 Hz, even if you are interested only in the first three formants.

Problem: I often get only 1 formant in a region where I see clearly 2 formants in the spectrogram.

This occurs mainly in back vowels (F1 and F2 close together) for male voices, if the "maximum formant" is set to the standard of 5500 Hz, which is appropriate for female voices. Set the "maximum formant" down to 5000 Hz. No, Praat comes without a guarantee: the formant analysis is based on LPC, and this comes with several assumptions as to what a speech spectrum is like.

Question: what algorithm is used for formant analysis?

Answer: see Sound: To Formant (burg)....

Links to this page

- [FAQ \(Frequently Asked Questions\)](#)

© ppgb, September 16, 2003

Formant: Track...

A command to extract a specified number of formant tracks from each selected Formant object. The tracks represent the cheapest paths through the measured formant values in consecutive frames.

How to use

In order to be capable of producing three tracks (i.e. F1, F2, and F3), there must be at least three formant candidates in *every* frame of the Formant object. The typical use of this command, therefore, is to analyse five formants with Sound: To Formant (burg)... and then use the tracking command to extract three tracks.

When to use, when not

This command only makes sense if the whole of the formant contour makes sense. For speech, formant contours make sense only for vowels and the like. During some consonants, the Formant object may have fewer than three formant values, and trying to create three tracks through them will fail. You will typically use this command for the contours in diphthongs, if at all.

Arguments

To be able to interpret the arguments, you should know that the aim of the procedure is to minimize the sum of the *costs* associated with the three tracks.

Number of tracks

the number of formant tracks that the procedure must find. If this number is 3, the procedure will try to find tracks for F1, F2, and F3; if the Formant object contains a frame with less than three formants, the tracking procedure will fail.

Reference F1 (Hertz)

the preferred value near which the first track wants to be. For average (i.e. adult female) speakers, this value will be around the average F1 for vowels of female speakers, i.e. 550 Hz.

Reference F2 (Hertz)

the preferred value near which the second track wants to be. A good value will be around the average F2 for vowels of female speakers, i.e. 1650 Hz.

Reference F3 (Hertz)

the preferred value near which the third track wants to be. A good value will be around the average F3 for vowels of female speakers, i.e. 2750 Hz. This argument will be ignored if you choose to have fewer than three tracks, i.e., if you are only interested in F1 and F2.

Reference F4 (Hertz)

the preferred value near which the fourth track wants to be. A good value may be around 3850 Hz, but you will usually not want to track F4, because traditional formant lore tends to ignore it (however inappropriate this may be for the vowel [i]), and because Formant objects often contain not more than three formant values in some frames. So you will not usually specify a higher *Number of tracks* than 3, and in that case, this argument will be ignored.

Reference F5 (Hertz)

the preferred value near which the five track wants to be. In the unlikely case that you want five tracks, a good value may be around 4950 Hz.

Frequency cost (per kiloHertz)

the local cost of having a formant value in your track that deviates from the reference value. For instance, if a candidate (i.e. any formant in a frame of the Formant object) has a formant frequency of 800 Hz, and the *Frequency cost* is 1.0/kHz, the cost of putting this formant in the first track is 0.250, because the distance to the reference F1 of 550 Hz is 250 Hz. The cost of putting the formant in the second track would be 0.850 (= (1.650 kHz - 0.600 kHz) · 1.0/kHz), so we see that the procedure locally favours the inclusion of the 800 Hz candidate into the F1 track. But the next two cost factors may override this local preference.

Bandwidth cost

the local cost of having a bandwidth, relative to the formant frequency. For instance, if a candidate has a formant frequency of 400 Hz and a bandwidth of 80 Hz, and the *Bandwidth cost* is 1.0, the cost of having this formant in any track is (80/400) · 1.0 = 0.200. So we see that the procedure locally favours the inclusion of candidates with low relative bandwidths.

Transition cost (per octave)

the cost of having two different consecutive formant values in a track. For instance, if a proposed track through the candidates has two consecutive formant values of 300 Hz and 424 Hz, and the *Transition cost* is 1.0/octave, the cost of having this large frequency jump is (0.5 octave) · (1.0/octave) = 0.500.

Algorithm

This command uses a Viterbi algorithm with multiple planes. For instance, if the selected Formant object contains up to five formants per frame, and you request three tracks, the Viterbi algorithm will have to choose between ten candidates (the number of combinations of three out of five) for each frame.

The formula for the cost of e.g. track 3, with proposed values F_{2i} ($i = 1..N$, where N is the number of frames) is:

$$\begin{aligned} & \sum_{i=1..N} \text{frequencyCost} \cdot [\text{IMAGE}] F_{3i} - \text{reference}F3 [\text{IMAGE}] / 1000 + \\ & + \sum_{i=1..N} \text{bandWidthCost} \cdot B_{3i} / F_{3i} + \\ & + \sum_{i=1..N-1} \text{transitionCost} \cdot [\text{IMAGE}] \log_2 (F_{3i} / F_{3,i+1}) [\text{IMAGE}] \end{aligned}$$

Analogous formulas compute the cost of track 1 and track 2. The procedure will assign those candidates to the three tracks that minimize the sum of three track costs.

Links to this page

- [Formant: Draw tracks...](#)
-

© *ppgb*, March 8, 2002

Formant: Get bandwidth at time...

A query to the selected Formant object for the bandwidth of the specified formant at the specified time.

Arguments

Formant number

the ordinal number of the formant, counting up from 0 Hz. Specify "2" for F_2 etc.

Time (s)

the time at which to evaluate the bandwidth.

Units

the units of the result (Hertz or Bark).

Return value

the estimated bandwidth in Hertz or Bark. If *Time* is not within half a frame width of any frame centre, or If the specified *Formant number* is greater than the number of formants in the frame, the return value is undefined; otherwise, the formant is considered to belong to the frame whose centre is nearest to the specified time.

Algorithm

If possible (i.e. if the adjacent frame has enough formants), a linear interpolation is performed between the centre of the frame and the centre of the adjacent frame. With Bark units, the Hertz-to-Bark transformation is performed on the two frequencies $F \pm B$ (after interpolation), and the result is the difference between these two values

© ppgb, October 16, 1999

Formant: Get maximum...

A query to ask the selected Formant object for the maximum value of the specified formant.

Return value

the maximum, in Hertz or Bark.

Arguments

Formant number

the ordinal number of the formant, counting up from 0 Hz. Specify "2" for F_2 etc.

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the formant contour is considered.

Units

the units of the result (Hertz or Bark).

Interpolation

the interpolation method (none or parabolic). See vector peak interpolation.

© ppgb, October 16, 1999

Formant: Get mean...

A query to ask the selected Formant object for the mean value of the specified formant.

Return value

the mean, in Hertz or Bark.

Arguments

Formant number

the ordinal number of the formant, counting up from 0 Hz. Specify "2" for F_2 etc.

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the formant contour is considered.

Units

the units of the result (Hertz or Bark).

Scripting

You can use this command to put the mean into a script variable:

```
select Formant hallo
mean = Get mean... 2 0 0 Hertz
```

In this case, the value will not be written into the Info window.

© ppgb, October 16, 1999

Formant: Get minimum...

A query to ask the selected Formant object for the minimum value of the specified formant.

Return value

the minimum, in Hertz or Bark.

Arguments

Formant number

the ordinal number of the formant, counting up from 0 Hz. Specify "2" for F_2 etc.

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the formant contour is considered.

Units

the units of the result (Hertz or Bark).

Interpolation

the interpolation method (none or parabolic). See vector peak interpolation.

© ppgb, October 16, 1999

Formant: Get number of formants

A query to ask the selected Formant object for the number of formants in a specified frame.

Argument

Frame number

the frame number whose formant count is sought.

Return value

the number of formants.

© ppgb, October 16, 1999

Formant: Get quantile...

A query to ask the selected Formant object for an estimation of the specified quantile of the distribution that underlies the attested values of the specified formant.

Return value

the quantile, in Hertz or Bark.

Arguments

Formant number

the ordinal number of the formant, counting up from 0 Hz. Specify "2" for F_2 etc.

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the formant contour is considered.

Units

the units of the result (Hertz or Bark).

Quantile

the fraction (between 0 and 1) of the values of the underlying distribution expected to lie below the result. For instance, if *Quantile* is 0.10, the algorithm estimates the formant value below which 10% of all formant values are expected to lie. To get an estimate of the *median* of the underlying distribution, specify a quantile of 0.50.

Algorithm

First, the available formant values within the selected time domain are collected in an array. This array is then sorted and the quantile algorithm is performed. With Bark units, the Hertz-to-Bark transformation is performed before the quantile algorithm.

© ppgb, October 16, 1999

Formant: Get standard deviation

A query to ask the selected Formant object for the standard deviation of the attested values of the specified formant within a specified time domain.

Return value

the standard deviation, in Hertz or Bark.

Arguments

Formant number

the ordinal number of the formant, counting up from 0 Hz. Specify "2" for F_2 etc.

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the formant contour is considered.

Units

the units of the result (Hertz or Bark).

© ppgb, October 16, 1999

Formant: Get time of maximum...

A query to ask the selected Formant object for the time associated with the maximum value of a specified formant within a specified time domain.

Argument

Formant number

the ordinal number of the formant, counting up from 0 Hz. Specify "2" for F_2 etc.

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored, except for interpolation. If *To time* is not greater than *From time*, the entire time domain of the formant contour is considered.

Units

the units of the result (Hertz or Bark).

Interpolation

the interpolation method (None or Parabolic). See vector peak interpolation.

Return value

the time expressed in seconds. If no relevant formants are found, the value is undefined.

© ppgb, October 16, 1999

Formant: Get time of minimum...

A query to ask the selected Formant object for the time associated with the minimum value of a specified formant within a specified time domain.

Argument

Formant number

the ordinal number of the formant, counting up from 0 Hz. Specify "2" for F_2 etc.

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored, except for interpolation. If *To time* is not greater than *From time*, the entire time domain of the formant contour is considered.

Units

the units of the result (Hertz or Bark).

Interpolation

the interpolation method (None or Parabolic). See vector peak interpolation.

Return value

the time expressed in seconds. If no relevant formants are found, the value is undefined.

© ppgb, October 16, 1999

Formant: Get value at time...

A query to ask the selected Formant object for the frequency of the specified formant at the specified time.

Arguments

Formant number

the ordinal number of the formant, counting up from 0 Hz. Specify "2" for F_2 etc.

Time (s)

the time at which to evaluate the formant frequency.

Units

the units of the result (Hertz or Bark).

Interpolation

the interpolation method, see vector value interpolation. Always Linear.

Return value

the bandwidth in Hertz or Bark. If *Time* is not within half a frame width of any frame centre, or if the specified *Formant number* is greater than the number of formants in the frame, the value is undefined; otherwise, the formant is considered to belong to the frame whose centre is nearest to the specified time.

Algorithm

If possible (i.e. if the adjacent frame has enough formants), a linear interpolation is performed between the centre of the frame and the centre of the adjacent frame. With Bark units, the Hertz-to-Bark transformation is performed before interpolation.

© ppgb, October 16, 1999

Intensity & TextTier: To IntensityTier...

A command to copy information from an Intensity, at times specified by a TextTier, to points on an IntensityTier.

Behaviour

For all the times of the points in the TextTier, an intensity is computed from the information in the Intensity object, by linear interpolation.

© *ppgb*, March 21, 1997

Intensity: Get maximum...

A query to the selected Intensity object.

Return value

the maximum value within the specified time domain, expressed in dB.

Arguments

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Intensity object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Parabolic because of the usual nonlinearity (logarithm) in the computation of intensity; sinc interpolation would be too stiff and may give unexpected results.

© ppgb, September 16, 2003

Intensity: Get mean...

A query to the selected Intensity object.

Return value

the mean (in dB) of the intensity values of the frames within a specified time domain.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Intensity is considered.

Algorithm

The mean intensity between the times t_1 and t_2 is defined as

$$1/(t_2 - t_1) \int_{t_1}^{t_2} dt x(t)$$

where $x(t)$ is the intensity as a function of time, in dB. For our discrete Intensity object, this mean is approximated by

$$1/n \sum_{i=m..m+n-1} x_i$$

where n is the number of frame centres between t_1 and t_2 .

© ppgb, October 16, 1999

Intensity: Get minimum...

A query to the selected Intensity object.

Return value

the minimum value within a specified time domain, expressed in dB.

Arguments

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Intensity object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Parabolic because of the usual nonlinearity (logarithm) in the computation of intensity; sinc interpolation would be too stiff and may give unexpected results.

© ppgb, September 16, 2003

Intensity: Get standard deviation...

A query to the selected Intensity object.

Return value

the standard deviation (in dB) of the intensity values of the frames within a specified time domain.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Intensity is considered.

Algorithm

The standard deviation between the times t_1 and t_2 is defined as

$$1/(t_2 - t_1) \int_{t_1}^{t_2} dt (x(t) - \mu)^2$$

where $x(t)$ is the intensity (in dB) as a function of time, and μ its mean. For our discrete Intensity object, the standard deviation is approximated by

$$1/(n-1) \sum_{i=m..m+n-1} (x_i - \mu)^2$$

where n is the number of frame centres between t_1 and t_2 . Note the "minus 1".

© ppgb, October 16, 1999

Intensity: Get time of maximum...

A query to the selected Intensity object.

Return value

the time (in seconds) associated with the maximum intensity within a specified time domain.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Intensity object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Parabolic because of the usual nonlinearity (logarithm) in the computation of intensity; sinc interpolation would be too stiff and may give unexpected results.

© ppgb, September 16, 2003

Intensity: Get time of minimum...

A query to the selected Intensity object.

Return value

the time (in seconds) associated with the minimum intensity within a specified time domain.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Intensity object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Parabolic because of the usual nonlinearity (logarithm) in the computation of intensity; sinc interpolation would be too stiff and may give unexpected results.

© ppgb, September 16, 2003

Intensity: Get value at time...

A query to the selected Intensity object.

Return value

the intensity (in dB) at a specified time. If *time* is outside the frames of the Intensity, the result is 0.

Arguments

Time (s)

the time at which the value is to be evaluated.

Interpolation

the interpolation method, see vector value interpolation. The standard is Cubic because of the usual nonlinearity (logarithm) in the computation of intensity; sinc interpolation would be too stiff and may give unexpected results.

© ppgb, September 16, 2003

Intensity: Get value in frame...

A query to the selected Intensity object.

Argument

Frame number

the frame whose value is to be looked up.

Return value

the intensity value (in dB) in the specified frame. If the index is less than 1 or greater than the number of frames, the result is 0; otherwise, it is $z[1][frame\ number]$.

© ppgb, October 16, 1999

Intensity: To IntensityTier

A command to convert each selected Intensity object to an IntensityTier.

Behaviour

Every sample in the Intensity object is copied to a point on the IntensityTier.

Postconditions

Equal time domains:

• *result.xmin == intensity.xmin*

• *result.xmax == intensity.xmax*

Equal number of points:

• *result.points.size == intensity.nx*

For all points $i = 1 \dots \text{intensity.nx}$:

Explicit times:

• *result.points.item [i].time == intensity.x1 + (i - 1) * intensity.dx*

Equal number of points:

• *result.points.item [i].value == intensity.z [1] [i]*

© ppgb, March 21, 1997

PointProcess: To TextGrid...

A command to create an empty TextGrid from every selected PointProcess.

The only information in the PointProcess that is used, is its starting and finishing times.

Arguments

Tier names

a list of the names of the tiers that you want to create, separated by spaces.

Point tiers

a list of the names of the tiers that you want to be *point tiers*; the rest of the tiers will be *interval tiers*.

Example

If *Tier names* is "a b c", and *Point tiers* is "b", the resulting TextGrid object will contain an interval tier named "a", a point tier named "b", and another interval tier named "c".

© ppgb, January 13, 1998

TextTier(s): To TextGrid

A command to merge all selected TextTier objects into a new TextGrid object.

You can also merge with IntervalTier objects:
TextTier(s) & IntervalTier(s): To TextGrid

© *ppgb*, March 17, 1997

IntervalTier(s): To TextGrid

A command to merge all selected IntervalTier objects into a new TextGrid object.

You can also merge with TextTier objects:

TextTier(s) & IntervalTier(s): To TextGrid

© *ppgb*, March 17, 1997

TextTier(s) & IntervalTier(s): To TextGrid

A command to merge all selected TextTier and IntervalTier objects into a new TextGrid object.

Links to this page

- [IntervalTier\(s\): To TextGrid](#)
- [TextTier\(s\): To TextGrid](#)

© *ppgb*, March 17, 1997

TextGrid & TextTier: Append

A command to append the selected TextTier to the selected TextGrid.

© *ppgb*, March 17, 1997

TextGrid & IntervalTier: Append

A command to append the selected IntervalTier to the selected TextGrid.

© *ppgb*, March 17, 1997

TextGrid & TextGrid: Merge

A command to merge two selected TextGrid objects into a new TextGrid.

© *ppgb*, March 17, 1997

Label & Sound: To TextGrid

A command to convert an old-style **Label** object, which you have probably read from an old file, into a new-style TextGrid object.

The Sound is needed to supply the TextGrid with a finite time domain, which old-style Label objects lacked.

© *ppgb*, March 17, 1997

TextGrid: Extend time...

Extends the domain of the selected TextGrid object.

Arguments

Extend domain by

defines the amount of time by which the domain will be extended.

At

defines whether starting times or finishing times will be modified.

Behaviour

We add an extra (empty) interval into each *interval tier*. This is necessary to keep original intervals intact. According to the value of the second argument, the new interval will be added at the beginning or at the end of the tier.

For *point tiers* only the domain will be changed.

© djmw, July 2, 2002

Manipulation: Extract duration tier

A command to extract a copy of the duration information in each selected Manipulation object into a new DurationTier object.

© *ppgb*, March 30, 2001

Manipulation: Extract pitch tier

A command to extract a copy of the pitch information in each selected Manipulation object into a new PitchTier object.

© *ppgb*, March 30, 2001

Manipulation: Extract pulses

A command to extract a copy of the vocal-pulse information in each selected Manipulation object into a new PointProcess object.

© *ppgb*, March 30, 2001

Manipulation: Get resynthesis (PSOLA)

A command to extract the sound from each selected Manipulation object, resynthesized with the PSOLA method.

© *ppgb*, March 30, 2001

Manipulation: Play (PSOLA)

A command to play each selected Manipulation object, resynthesized with the PSOLA method.

© *ppgb*, March 30, 2001

Manipulation: Replace duration tier

You can replace the duration tier that you see in your Manipulation object with a separate DurationTier object, for instance one that you extracted from another Manipulation or one that you created with Create DurationTier....

To do this, select your Manipulation object together with the DurationTier object and click **Replace duration tier**.

© *ppgb*, February 16, 2003

Manipulation: Replace pitch tier

You can replace the pitch tier that you see in your Manipulation object with a separate PitchTier object, for instance one that you extracted from another Manipulation or one that you created with Create PitchTier....

To do this, select your Manipulation object together with the PitchTier object and click **Replace pitch tier**.

© ppgb, February 16, 2003

Manipulation: Replace pulses

A command to replace the vocal-pulse information in the selected Manipulation object with the selected PointProcess object.

© *ppgb*, March 30, 2001

PointProcess: Up to PitchTier...

A command to promote every selected PointProcess to a PitchTier.

Argument

Frequency (Hz)

the pitch frequency that will be associated with every point.

Behaviour

The times of all the points are trivially copied, and so is the time domain. The pitch information will be the same for every point.

© ppgb, March 29, 1997

PointProcess: To PitchTier...

A command to compute a PitchTier from a PointProcess.

Argument

Maximum interval (s)

the maximum duration of a period; intervals longer than this are considered voiceless.

Algorithm

A pitch point is constructed between each consecutive pair of points in the **PointProcess**, if these are more than *maximumInterval* apart. The associated pitch value will be the inverse of the duration of the interval between the two points.

© *ppgb*, April 2, 1997

PitchTier: Down to PointProcess

A command to degrade every selected PitchTier to a PointProcess.

Behaviour

The times of all the pitch points are trivially copied, and so is the time domain. The pitch information is lost.

© *ppgb*, April 10, 2001

Get low index from time...

A query to ask the selected tier object (DurationTier, FormantTier, IntensityTier, PitchTier, TextTier) which point is nearest to, but no later than, the specified time.

Argument

Time (s)

the time from which you want to get the point index.

Return value

This query returns the index of the point with the highest time less than or equal to *time*. It is undefined if there are no points. It is 0 (offleft) if the specified time is less than the time of the first point.

© ppgb, February 16, 2003

Get high index from time...

A query to ask the selected tier object (DurationTier, FormantTier, IntensityTier, PitchTier, TextTier) which point is nearest to, but no earlier than, the specified time.

Argument

Time (s)

the time from which you want to get the point index.

Return value

This query returns the index of the point with the lowest time greater than or equal to *time*. It is undefined if there are no points. It is the number of points plus 1 (offright) if the specified time is greater than the time of the last point.

© ppgb, February 16, 2003

Get nearest index from time...

A query to ask the selected tier object (DurationTier, FormantTier, IntensityTier, PitchTier, TextTier) which point is nearest to the specified time.

Argument

Time (s)

the time near which you want to get the point index.

Return value

This query returns the index of the point with the highest time less than or equal to *time*. It is undefined if there are no points.

© ppgb, February 16, 2003

Get area...

A query to the selected tier object (PitchTier, IntensityTier, DurationTier).

Return value

the area under the curve.

Attributes

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the tier is considered.

Algorithm

The curve consists of a sequence of line segments. The contribution of the line segment from (t_1, f_1) to (t_2, f_2) to the area is

$$1/2 (f_1 + f_2) (t_2 - t_1)$$

© ppgb, February 16, 2003

PitchTier: Get mean (points)...

A query to the selected PitchTier object.

Return value

the mean of the points within a specified time window.

Attributes

From time (s), *To time* (s)

the time window. Values outside this window are ignored. If *To time* is not greater than *From time*, the entire time domain of the tier is considered.

To get the mean in the entire curve, i.e. weighted by the durations of the line pieces, Use PitchTier: Get mean (curve)... instead.

© ppgb, August 21, 2001

PitchTier: Get standard deviation (curve)...

A query to the selected PitchTier object.

Return value

the standard deviation in the curve within a specified time window.

Attributes

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the tier is considered.

Algorithm

The curve consists of a sequence of line segments. The contribution of the line segment from (t_1, f_1) to (t_2, f_2) to the variance-multiplied-by-time is

$$\left[\frac{1}{4} (f_1 + f_2)^2 + \frac{1}{12} (f_1 - f_2)^2 \right] (t_2 - t_1)$$

The standard deviation is the square root of: the sum of these values divided by *toTime* - *fromTime*.

To get the standard deviation in the points only, i.e. not weighted by the durations of the line pieces, Use PitchTier: Get standard deviation (points)... instead.

© ppgb, August 21, 2001

DurationTier: Add point...

A command to add a point to each selected DurationTier. For an example, see Create DurationTier...

Arguments

Time (s)

the time at which a point is to be added.

Relative duration

the relative duration value of the requested new point.

Behaviour

The tier is modified so that it contains the new point. If a point at the specified time was already present in the tier, nothing happens.

© ppgb, February 16, 2003

DurationTier: Get target duration...

A query to the selected DurationTier for the target duration of a specified time range.

Arguments

From time (s), *To time* (s)

the start and end of the time range. If *fromTime* or *toTime* is outside the time domain of the Duration object, there will be .

Return value

the target duration in seconds.

© ppgb, October 16, 1999

DurationTierEditor

One of the editors in the Praat program, for viewing and editing a DurationTier object. To create a DurationTierEditor window, select a DurationTier and click Edit.

© *ppgb*, March 30, 2001

IntensityTier: Add point...

A command to add a point to each selected IntensityTier.

Arguments

Time (s)

the time at which a point is to be added.

Intensity (dB)

the intensity value of the requested new point.

Behaviour

The tier is modified so that it contains the new point. If a point at the specified time was already present in the tier, nothing happens.

Links to this page

- [Create IntensityTier...](#)
-

© *ppgb*, April 10, 2001

PointProcess: Up to IntensityTier...

A command to promote every selected PointProcess to an IntensityTier.

Argument

Intensity (dB)

the intensity that will be associated with every point.

Behaviour

The times of all the points are trivially copied, and so is the time domain. The intensity information will be the same for every point.

© ppgb, March 29, 1997

IntensityTierEditor

One of the editors in the Praat program, for viewing and editing an IntensityTier object. To create a IntensityTierEditor window, select an IntensityTier and click Edit.

© *ppgb*, March 30, 2001

IntensityTier: Down to PointProcess

A command to degrade every selected IntensityTier to a PointProcess.

Behaviour

The times of all the points are trivially copied, and so is the time domain. The intensity information is lost.

© *ppgb*, April 10, 2001

PointProcess: Get jitter (ddp)...

A query to the selected PointProcess object.

Return value

the periodic jitter, which is defined as the relative mean absolute third-order difference of the point process (= the second-order difference of the interval process):

$$jitter = \sum_{i=2}^{N-1} |2T_i - T_{i-1} - T_{i+1}| / \sum_{i=2}^{N-1} T_i$$

where T_i is the i th interval and N is the number of intervals. If no sequences of three intervals can be found whose durations are between *Shortest period* and *Longest period*, the result is undefined.

Arguments

Shortest period (seconds)

the shortest possible interval that will be considered. For intervals T_i shorter than this, the $(i-1)$ st, i th, and $(i+1)$ st terms in the formula are taken as zero. This argument will normally be very small, say 0.1 ms.

Longest period (seconds)

the shortest possible interval that will be considered. For intervals T_i longer than this, the $(i-1)$ st, i th, and $(i+1)$ st terms in the formula are taken as zero. For example, if the minimum frequency of periodicity is 50 Hz, set this argument to 20 milliseconds; intervals longer than that will be considered voiceless.

Usage

The periodic jitter can be used as a measure of voice quality. See Voice 2. Jitter.

© ppgb, May 21, 2003

PointProcess: Get jitter (local)...

A query to the selected PointProcess object. See Voice 2. Jitter.

© *ppgb*, May 21, 2003

PointProcess: Get jitter (local, absolute)...

A query to the selected PointProcess object. See Voice 2. Jitter.

© *ppgb*, May 21, 2003

PointProcess: Get jitter (ppq5)...

A query to the selected PointProcess object. See Voice 2. Jitter.

© *ppgb*, May 21, 2003

PointProcess: Get jitter (rap)...

A query to the selected PointProcess object. See Voice 2. Jitter.

© *ppgb*, May 21, 2003

Boersma (1993)

Paul Boersma (1993): "Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound." *Proceedings of the Institute of Phonetic Sciences* **17**: 97-110. University of Amsterdam.

Can be downloaded as a PDF file from <http://fon.hum.uva.nl/paul/>

Links to this page

- Sound: To Harmonicity (ac)...
- Sound: To Pitch (ac)...
- Voice 5. Comparison with other programs

© ppgb, March 12, 2003

Deliyski (1993)

Dimitar D. Deliyski: "Acoustic model and evaluation of pathological voice production." *Proceedings Eurospeech '93*, Vol. 3, 1969-1972.

Links to this page

- [Voice 5. Comparison with other programs](#)
-

© *ppgb*, March 12, 2003

Anderson (1978)

N. Anderson (1978), "On the calculation of filter coefficients for maximum entropy spectral analysis", in Childers, *Modern Spectrum Analysis*, IEEE Press, 252-255.

Links to this page

- [Sound: To LPC \(burg\)...](#)
-

© *djmw*, July 1, 2003

LPC: To LFCC...

You can choose this command after selecting 1 or more LPC objects.

Behaviour

The transformation from a-coefficients to cepstral coefficients as described in Markel & Gray (1976).

Settings

Number of coefficients

the desired number of cepstral coefficients.

Links to this page

- [LFCC](#)
-

© *djmw*, April 7, 2004

LPC: To Polynomial (slice)...

A command that creates a Polynomial object from each selected LPC object.

Settings

Time (s)

defines the LPC frame whose coefficients will be selected.

Behaviour

The linear prediction coefficients $a_{1..n}$ of the selected LPC frame will be copied to polynomial coefficients $c_{1..n+1}$ as follows:

$$c_i = a_{n-i+1},$$

$$c_{n+1} = 1$$

Links to this page

- [LPC: Draw poles...](#)
- [Polynomial](#)

© *djmw*, April 7, 2004

LPC: Draw gain...

You can choose this command after selecting 1 or more LPC objects.

Settings

From time (seconds), To time (seconds)
the time domain along the x -axis.

Minimum gain, Maximum gain
the range for the y -axis.

Garnish
determines whether to draw a bounding box and axis labels.

Behaviour

Gain will be drawn as a function of time (gain also equals the prediction error energy).

© *djmw*, April 7, 2004

LPC: Draw poles...

You can choose this command after selecting 1 or more LPC objects.

Settings

Time

the time of the nearest frame.

Behaviour

The roots of the linear prediction polynomial, constructed from the coefficients of the analysis frame, will be drawn in the complex plane.

© *djmw*, April 7, 2004

LPC: To VocalTract (slice)...

You can choose this command after selecting 1 or more LPC objects.

Settings

Time

the time of the nearest frame.

Length

the length of the vocal tract.

Length according to Wakita

the length of the vocal tract is calculated according to the algorithm as described in Wakita (1977).

Behaviour

A new VocalTract area function is calculated from the prediction coefficients in the frame.

© djmw, April 7, 2004

Aliasing

Aliasing (Du. *vouwvervorming*) is the phenomenon of the ambiguity of a sampled signal.

Example

With a sampling frequency of 10 kHz, a sine wave with a frequency of 3 kHz receives the same representation as a sine wave with a frequency of 7 kHz, 13 kHz, or 17 kHz, and so on. If the sampled signal is meant to represent a continuous spectral range starting at 0 Hz (which is the most common case for speech recordings), all these tones are likely to be interpreted as 3 kHz tones after sampling.

To remedy this unwanted situation, the signal is usually low-pass filtered with a cut-off frequency just below 5 kHz, prior to sampling.

Links to this page

- [Create Sound from gamma-tone...](#)
- [Nyquist frequency](#)
- [sampling frequency](#)

© *ppgb*, March 31, 2004

Vector peak interpolation

An algorithm for finding a maximum or a minimum in a sampled signal.

Overview

The signal is described with the sequence y_i , $i = 1 \dots n$, where n is the number of samples. Each sample i is associated with an x value (typically, time) given by $x_i = x_1 + (i - 1) dx$, where dx is the sample period.

The maximum is looked for in two kinds of locations:

1. At the left and right edge, i.e. at $i = 1$ and at $i = n$.
2. At or *near* all local maxima, i.e. at or *near* those i that satisfy $y_{i-1} < y_i \leq y_{i+1}$.

The greatest of the following values, therefore, will be the maximum:

1. y_1 .
2. The local maxima, which are at or *near* y_i , where $y_{i-1} < y_i \leq y_{i+1}$.
3. y_n .

We will now see what *near* means. The precision of the result depends on the *interpolation method* of this algorithm.

1. Lowest precision: round to sample

If the interpolation method is None, the local maxima are *at* the samples m that satisfy $y_{m-1} < y_m \leq y_{m+1}$. Thus, their x values are at $x_m = x_1 + (m - 1) dx$, and their y values are y_m .

This kind of precision is appropriate for an unordered sequence of values y_i : the result is simply the greatest available value.

2. Middle precision: parabolic interpolation

If the interpolation method is Parabolic, the algorithm uses one point on each side of every local maximum y_m to estimate the location and value of the local maximum. Because a Taylor expansion shows that any smooth curve can be approximated as a parabola in the vicinity of any local maximum, the location x_{max} and value y_{max} can be found with the following procedure:

$$dy \equiv 1/2 (y_{m+1} - y_{m-1})$$

$$d^2y \equiv 2 y_m - y_{m-1} - y_{m+1}$$

$$m' \equiv m + dy/d^2y$$

$$x_{max} = x_1 + (m' - 1) dx$$

$$y_{max} = y_m + 1/2 dy^2 / d^2y$$

This kind of precision is appropriate if y is considered a smooth function of x , as in:

- Formant: Get minimum...
- Formant: Get time of minimum...
- Formant: Get maximum...
- Formant: Get time of maximum...
- Intensity: Get minimum...
- Intensity: Get time of minimum...
- Intensity: Get maximum...
- Intensity: Get time of maximum...

3. Higher precision: cubic interpolation

If the interpolation method is Cubic, the interpolation is performed over four points (see vector value interpolation). The results are similar to those of the parabolic interpolation method, but you can use it (or sinc interpolation) if you want the result of a command like **Get maximum...** to be equal to the result of a sequence of commands like **Get time of maximum...** and **Get value at time...**

4. Highest precision: sinc interpolation

If the interpolation method is Sinc70 or Sinc700, the algorithm assumes that the signal is a sum of sinc functions, so that a number of points (namely, 70 or 700) on each side of the initial guess m must be taken into account (see vector value interpolation). The algorithm finds the maximum of this continuous function by Brent's method (see Press et al. (1992)).

This method is appropriate for signals that result from sampling a continuous signal after it has been low-pass filtered at the Nyquist frequency. See:

- Sound: Get minimum...
- Sound: Get time of minimum...
- Sound: Get maximum...
- Sound: Get time of maximum...
- Sound: Get absolute extremum...

Links to this page

- [Harmonicity: Get maximum...](#)
 - [Harmonicity: Get minimum...](#)
 - [Harmonicity: Get time of maximum...](#)
 - [Harmonicity: Get time of minimum...](#)
 - [Ltas: Get frequency of maximum...](#)
 - [Ltas: Get frequency of minimum...](#)
 - [Ltas: Get maximum...](#)
 - [Ltas: Get minimum...](#)
-

© *ppgb*, April 10, 2001

Vector value interpolation

An algorithm for estimating the value of a sampled signal at a specified location.

Overview

The signal is described with the sequence $y_i, i = 1 \dots n$, where n is the number of samples. Each sample i is associated with an x location (typically, time) given by $x_i = x_1 + (i - 1) dx$, where dx is the sample period, so that the real-valued sample number associated with a given time x is

$$s = (x - x_1) / dx + 1$$

If the resulting s is an integer number, the y value must be y_s . Otherwise, the estimated y value $y(s)$ must be interpolated from nearby values of y . The precision of the result depends on the *interpolation method* of this algorithm.

1. Lowest precision: round to sample

If the interpolation method is Nearest, we take the value of the nearest point:

$$near \equiv \text{round}(s)$$

$$y(s) \approx y_{near}$$

2. Middle precision: linear interpolation

If you know or assume that the function that underlies your points is continuous, the "rounding" interpolation would be poor, because the rounded value would abruptly change at the centres between the sample points.

For a linear interpolation, therefore, we use the attested values on both sides (*left* and *right*) of s :

$$s_l \equiv \text{floor}(s) ; s_r \equiv s_l + 1$$

$$y(s) \approx y_l + (s - s_l) \cdot (y_r - y_l)$$

where **floor** (x) computes the greatest integer not greater than x . This interpolation is continuous.

3. Higher precision: cubic interpolation

If you know or assume that the function that underlies your points is *smooth*, i.e. its derivative is defined for every x , linear interpolation would probably be poor, because the derivative of the interpolated function would abruptly change at every sample point.

The next higher interpolation (Cubic), therefore, is differentiable at sample points. To enforce this, we define the derivatives y'_l and y'_r at the left and right sample points on the basis of *their* immediate neighbours (i.e., the algorithm needs four sample points), perhaps by a parabolic interpolation through these three points. A parabolic interpolation has the advantage that the extrema will be computed correctly if the underlying function can be approximated by a parabola near its extremes (see vector peak interpolation).

Because the derivative of a parabolic function is a linear function of x , the derivatives at the left and right sample points are simply estimated as

$$y'_l \approx (y_r - y_{l-1}) / 2 ; y'_r \approx (y_{r+1} - y_l) / 2$$

Now that we know y_l, y_r, y'_l , and y'_r , we can fit these values with a third-degree (*cubic*) polynomial:

$$As_l^3 + Bs_l^2 + Cs_l + D = y_l$$

$$As_r^3 + Bs_r^2 + Cs_r + D = y_r$$

$$3As_l^2 + 2Bs_l + C = y'_l$$

$$3As_r^2 + 2Bs_r + C = y'_r$$

If we shift the x axis to the left sample point, we can reduce the four equations to

$$D = y_l$$

$$A + B + C + D = y_r$$

$$C = y'_l$$

$$3A + 2B + C = y'_r$$

so that the interpolated value $y(s)$ at any point s between s_l and s_r is estimated as

$$(y'_r + y'_l - 2y_r + 2y_l) \varphi_l^3 + (3y_r - 3y_l - 2y'_l - y'_r) \varphi_l^2 + y'_l \varphi_l + y_l$$

where $\varphi_l \equiv s - s_l$. Some rearrangement gives

$$y(s) \approx y_l \varphi_r + y_r \varphi_l + \varphi_l \varphi_r [1/2 (y'_r - y'_l) + (\varphi_l - 1/2) (y'_l + y'_r - 2(y_r - y_l))]$$

where $\varphi_r \equiv 1 - \varphi_l$. From this formula we see:

1. The first two terms define the linear interpolation.
2. If the underlying function is linear, so that y'_l equals y'_r and both equal $y_r - y_l$, the higher-degree terms are zero.
3. If $y'_l + y'_r$ equals $2(y_r - y_l)$, the third-degree term is zero, so that the interpolated function is parabolic.
4. At the left and right points, one of the φ is 0 and the other is 1, so that at these boundary points, y is computed with exact precision.

4. Highest precision: sinc interpolation

If the interpolation method is Sinc70 or Sinc700, the algorithm assumes that the signal is a sum of sinc functions, so that a number of points (the *interpolation depth*: 70 or 700) on each side of s must be taken into account.

Because the interpolation depth must be finite, the sum of sinc functions is multiplied by a Hanning window:

$$s_l \equiv \text{floor}(s); s_r \equiv s_l + 1$$

$$\varphi_l \equiv s - s_l; \varphi_r \equiv 1 - \varphi_l$$

$$y(s) \approx \sum_{i=1 \dots N} y_{r-i} \text{sinc}(\pi(\varphi_l + i - 1)) (1/2 + 1/2 \cos(\pi(\varphi_l + i - 1)/(\varphi_l + N))) + \sum_{i=1 \dots N} y_{l+i} \text{sinc}(\pi(\varphi_r + i - 1)) (1/2 + 1/2 \cos(\pi(\varphi_r + i - 1)/(\varphi_r + N)))$$

where the sinc function is defined as

$$\text{sinc}(0) \equiv 1; \text{sinc}(x) \equiv \sin x / x \text{ for } x \neq 0$$

If s is less than the interpolation depth or greater than $n + 1$ minus the interpolation depth, the depth is reduced accordingly.

This method is appropriate for signals that result from sampling a continuous signal after it has been low-pass filtered at the Nyquist frequency. See:

Sound: Get value at time...

Links to this page

- Formant: Get value at time...
- Harmonicity: Get value at time...
- Intensity: Get value at time...
- Ltas: Get value at frequency...

© *ppgb*, January 4, 1998

FormantTier: Down to PointProcess

A command to degrade every selected FormantTier to a PointProcess.

Behaviour

The times of all the formant points are trivially copied, and so is the time domain. The formant information is lost.

© *ppgb*, April 10, 2001

TextTier: Down to PointProcess

A command to degrade every selected TextTier to a PointProcess.

Behaviour

The times of all the text points are trivially copied, and so is the time domain. The text information is lost.

© *ppgb*, April 10, 2001

PointProcess: Get low index...

A query to the selected PointProcess object.

Return value

the index of the nearest point before or at the specified time, or 0 if the point process contains no points or the specified time is before the first point.

Argument

Time (seconds)

the time from which a point is looked for.

© *ppgb*, December 12, 2002

PointProcess: Get high index...

A query to the selected PointProcess object.

Return value

the index of the nearest point at or after the specified time, 0 if the point process contains no points, or a number higher than the number of points if the specified time is after the last point.

Argument

Time (seconds)

the time from which a point is looked for.

© ppgb, December 12, 2002

PointProcess: Get nearest index...

A query to the selected PointProcess object.

Return value

the index of the point nearest to the specified time, or 0 if the point process contains no points.

Argument

Time (seconds)

the time around which a point is looked for.

© ppgb, December 12, 2002

PointProcess: Get interval...

A query to the selected PointProcess object.

Return value

the duration of the interval around a specified time. if the point process contains no points or if the specified time falls before the first point or not before the last point, the value is undefined. Otherwise, the result is the distance between the nearest points to the left and to the right of the specified time. If the point process happens to contain a point at exactly the specified time, the duration of the interval following this point is returned.

Argument

Time (seconds)

the time around which a point is looked for.

© *ppgb*, December 12, 2002

PointProcesses: Union

A command to merge two selected PointProcess objects into one.

Behaviour

The resulting **PointProcess** will contain all the points of the two original point processes, sorted by time. Points that occur in both original point processes, will occur only once in the resulting point process.

The time domain of the resulting point process is the union of the time domains of the original point processes.

© *ppgb*, December 12, 2002

PointProcesses: Intersection

A command to merge two selected PointProcess objects into one.

Behaviour

The resulting **PointProcess** will contain only those points that occur in both original point processes.

The time domain of the resulting point process is the intersection of the time domains of the original point processes.

© *ppgb*, December 12, 2002

PointProcesses: Difference

A command to compute the difference of two selected PointProcess objects.

Behaviour

The resulting **PointProcess** will contain only those points of the first selected original point process that do not occur in the second.

The time domain of the resulting point process is equal to the time domain of the first original point process.

© *ppgb*, December 12, 2002

PointProcess: Add point...

A command to add a point to each selected PointProcess.

Argument

Time (s)

the time at which a point is to be added.

Behaviour

The point process is modified so that it contains the new point. If a point at the specified time was already present in the point process, nothing happens.

© *ppgb*, April 10, 2001

PointProcess: Remove point...

A command to remove a point from every selected PointProcess.

Arguments

Index

the index of the point that is to be removed.

Behaviour

Does nothing if *index* is less than 1 or greater than the number of points *nt* in the point process. Otherwise, one point is removed (e.g., if *index* is 3, the third point is removed), and the other points stay the same.

© ppgb, December 12, 2002

PointProcess: Remove point near...

A command to remove a point from every selected PointProcess.

Arguments

Time (seconds)

the time around which a point is to be removed.

Behaviour

Does nothing if there are no points in the point process. Otherwise, the point nearest to *time* is removed, and the other points stay the same.

© ppgb, December 12, 2002

PointProcess: Remove points...

A command to remove a range of points from every selected PointProcess.

Arguments

From index (≥ 1)

the first index of the range of points that are to be removed.

To index

the last index of the range of points that are to be removed.

Behaviour

All points that originally fell in the range [*fromIndex*, *toIndex*] are removed, and the other points stay the same.

© ppgb, December 12, 2002

PointProcess: Up to TextTier...

A command to promote every selected PointProcess to a TextTier.

Argument

Text

the text that will be placed in every point.

Behaviour

The times of all the points are trivially copied, and so is the time domain. The text information will be the same for every point.

© *ppgb*, March 29, 1997

Boersma (1997a)

Paul Boersma (1997a): "The elements of Functional Phonology." Ms. University of Amsterdam.

Available as Rutgers Optimality Archive **173**, <http://rucss.rutgers.edu/roa.html>

Superseded by chapters 1 and 7 through 13 of Boersma (1998).

Links to this page

- [Create tongue-root grammar...](#)
- [OTGrammar_tongueRoot](#)

© *ppgb*, December 19, 1998

Boersma (1997b)

Paul Boersma (1997b): "How we learn variation, optionality, and probability." *Proceedings of the Institute of Phonetic Sciences* **21**: 43-58. University of Amsterdam.

Equals chapter 15 of Boersma (1998).

A less correct version (demoting and promoting a single pair of constraints, instead of them all) is available as Rutgers Optimality Archive **221**, <http://rucss.rutgers.edu/roa.html>

Links to this page

- OT learning 1. Kinds of OT grammars
- OTAnyGrammar examples

© ppgb, December 19, 1998

Boersma (2000)

Paul Boersma (2000): "Learning a grammar in Functional Phonology." In Joost Dekkers, Frank van der Leeuw, & Jeroen van de Weijer (eds.): *Phonology, Syntax, and Acquisition in Optimality Theory*. Oxford University Press.

An extended version is chapter 14 of Boersma (1998).

Links to this page

- [Create tongue-root grammar...](#)
- [OT learning 1. Kinds of OT grammars](#)
- [OTAnyGrammar & 2 Strings: Learn \(GLA\)...](#)
- [OTAnyGrammar examples](#)
- [OTAnyGrammar: Learn one \(GLA\)...](#)
- [OTGrammar & 2 Strings: Learn...](#)
- [OTGrammar: Learn one...](#)
- [OTGrammar_tongueRoot](#)

© ppgb, October 27, 2000

OTGrammar & 2 Strings: Learn...

Causes the selected OTGrammar object to process a number of input/output pairs according to the Gradual Learning Algorithm by Boersma (1998) and Boersma (2000). See OT learning 4. Learning an ordinal grammar and OT learning 5. Learning a stochastic grammar.

© *ppgb*, November 20, 2001

OTGrammar: Learn one...

Causes every selected OTGrammar object to process one input/output pair according to the Gradual Learning Algorithm by Boersma (1998) and Boersma (2000). See OT learning 4. Learning an ordinal grammar and OT learning 5. Learning a stochastic grammar.

© *ppgb*, November 20, 2001

phonToDifferenceLimens

A routine for converting sensation level in phons into intensity difference limen level, the inverse of differenceLimensToPhon.

Formula

$$\text{phonToDifferenceLimens}(\text{phon}) = 30 \cdot ((61/60)^{\text{phon}} - 1)$$

Derivation

In first approximation, humans can detect an intensity difference of 1 phon, i.e. if two sounds that differ only in intensity are played a short time after each other, people can generally detect their intensity difference if it is greater than 1 phon.

But the sensitivity is somewhat better for louder sounds. According to Jesteadt, Wier & Green (1977), the relative difference limen of intensity is given by

$$\text{DLI} = \Delta I / I = 0.463 \cdot (I / I_0)^{-0.072}$$

In this formula, I is the intensity of the sound in Watt/m^2 , I_0 is the intensity of the auditory threshold (i.e. $10^{-12} \text{ Watt/m}^2$ at 1000 Hz), and ΔI is the just noticeable difference.

Boersma (1998: 109) calculates a difference-limen scale from this. Given an intensity I , the number of difference limens above threshold is

$$\int_{I_0}^I dx \Delta I(x) = (1 / 0.463) \int_{I_0}^I dx I_0^{-0.072} x^{0.072-1}$$

$$= (1 / (0.463 \cdot 0.072)) ((I/I_0)^{0.072} - 1)$$

The sensation level in phon is defined as

$$\text{SL} = 10 \log_{10} (I/I_0)$$

so that the number of difference limens above threshold is

$$(1 / (0.463 \cdot 0.072)) (10^{(0.072/10)(10 \log(I/I_0))} - 1) = 30 \cdot (1.0167^{\text{SL}} - 1)$$

Links to this page

- [Formulas 4. Mathematical functions](#)
-

© *ppgb*, December 15, 2002

Prince & Smolensky (1993)

Alan Prince & Paul Smolensky (1993): *Optimality Theory: Constraint Interaction in Generative Grammar*. Rutgers University Center for Cognitive Science Technical Report 2.

Links to this page

- [Optimality Theory](#)
- [OT learning 1. Kinds of OT grammars](#)
- [OT learning 3.2. Data from another grammar](#)
- [OTAnyGrammar examples](#)

© ppgb, October 21, 1997



An abbreviation for Optimality Theory.

Links to this page

- [OT learning 1. Kinds of OT grammars](#)
-

© *ppgb*, November 5, 2002

Tesar & Smolensky (1998)

Bruce Tesar & Paul Smolensky (1998): "Learnability in Optimality Theory." *Linguistic Inquiry* **29**: 229-268.

The first version of the constraint-demotion algorithm appeared in:

Bruce Tesar & Paul Smolensky (1993): "The learnability of Optimality Theory: an algorithm and some basic complexity results", ms. Department of Computer Science & Institute of Cognitive Science, University of Colorado at Boulder. Available as Rutgers Optimality Archive **2**, <http://ruccs.rutgers.edu/roa.html>

The Error-Driven Constraint Demotion algorithm can be found in:

Bruce Tesar & Paul Smolensky (1996): "Learnability in Optimality Theory (long version)". Technical Report **96-3**, Department of Cognitive Science, Johns Hopkins University, Baltimore. Available as Rutgers Optimality Archive **156**, <http://ruccs.rutgers.edu/roa.html>

Links to this page

- OT learning 1. Kinds of OT grammars
- OT learning 4. Learning an ordinal grammar
- OT learning 6. Shortcut to OT learning
- OT learning 7. Learning from overt forms
- OTAnyGrammar & 2 Strings: Learn (T&S)
- OTAnyGrammar examples
- OTAnyGrammar: Learn one (T&S)...
- Robust Interpretive Parsing

© ppgb, October 19, 1999

OTGrammar: Generate inputs...

A command to create a Strings object from a selected OTGrammar.

A practical grammar-specific implementation of the *richness of the base*: the inputs are drawn at random with equal probabilities from the inputs associated with the tableaux. For an example, see OT learning 3.2. Data from another grammar.

Argument

Number of trials

the number of times a string will be drawn from the possible inputs to the grammar.

© ppgb, December 30, 1998

OTGrammar: Input to output...

A command to ask the selected OTGrammar object to evaluate the candidates associated with a specified input form.

See OT learning 2.8. Asking for one output for tutorial information.

Arguments

Input form

the input form whose surface form you want to know. If this string is not in the list of the possible inputs of the selected OTGrammar, you will get an error message.

Noise (standard value: 2.0)

the standard deviation of the noise added to the ranking value of every constraint during evaluation. See OT learning 2.4. Evaluation.

© ppgb, September 16, 2003

OTGrammar: Input to outputs...

A command to ask the selected OTGrammar object to evaluate a number of times the candidates associated with a specified input form. The result is a Distributions object.

See OT learning 2.9. Output distributions for tutorial information and examples.

Arguments

Trials (standard value: 1000)

the number of evaluations that you want to perform.

Noise (standard value: 2.0)

the standard deviation of the noise added to the ranking value of every constraint during the evaluations. See OT learning 2.4. Evaluation.

Input form

the input form whose surface forms you want to measure. If this string is not in the list of the possible inputs of the selected OTGrammar, you will get an error message.

© ppgb, September 16, 2003

OTGrammar: To output Distributions...

A command to ask the selected OTGrammar object to evaluate a number of times the candidates associated with every input form. The result is a Distributions object. See OT learning 2.9. Output distributions.

Links to this page

- [OT learning 5. Learning a stochastic grammar](#)
-

© *ppgb*, December 30, 1998

OTGrammar & Strings: Inputs to outputs...

An action that creates a Strings object from a selected OTGrammar and a selected Strings.

The selected Strings object is considered as a list of inputs to the OTGrammar grammar.

Arguments

Noise

the standard deviation of the noise that will be temporarily added to the ranking value at each evaluation.

The resulting Strings object will contain the output string of the grammar for each of the input strings.

See OT learning 3.2. Data from another grammar.

© ppgb, December 30, 1998

Robust Interpretive Parsing

The mapping from overt forms to surface forms in the acquisition model by Tesar & Smolensky (1998).

In PRAAT, you can do robust interpretive parsing on any OTGrammar object. See OT learning 7. Learning from overt forms.

© *ppgb*, November 5, 2002

Strings: To Distributions

A command to analyse each selected Strings object into a Distributions object.

The resulting **Distributions** will collect the occurrences of every string in the **Strings** object, and put the number of occurrences in its first and only column.

Example

We start from the following **Strings**:

```
6 (number of strings)
"hallo"
"dag allemaal"
"hallo"
"tot morgen"
"hallo"
"tot morgen"
```

This will give us the following **Distributions**:

```
1 (number of columns) "" (no column name)
"hallo" 3
"dag allemaal" 1
"tot morgen" 2
```

Links to this page

- [OT learning 2.9. Output distributions](#)
- [OTAnyGrammar examples](#)

© ppgb, October 25, 1997

PairDistribution: To Strings...

A command to generate a number of string pairs from the selected PairDistribution object. This command will create two aligned Strings objects of equal size.

Arguments

Number (standard: 1000)

the number of the strings in either resulting Strings object.

Name of first Strings (standard: "input")

the name of the resulting Strings object associated with the first string of each pair.

Name of second Strings (standard: "output")

the name of the resulting Strings object associated with the second string of each pair.

Example

Suppose the PairDistribution contains the following:

```
4 pairs
"at+ma" "atma" 100
"at+ma" "apma" 0
"an+pa" "anpa" 20
"an+pa" "ampa" 80
```

The resulting Strings object "input" may then contain:

```
at+ma, an+pa, an+pa, at+ma, at+ma, an+pa, an+pa, an+pa, an+pa, at+ma, ...
```

The Strings object "output" may then contain:

```
atma, ampa, ampa, atma, atma, ampa, anpa, ampa, ampa, atma, ...
```

Links to this page

- [OT learning 3.1. Data from a pair distribution](#)

© ppgb, September 16, 2003

Archangeli & Pulleyblank (1994)

Diana Archangeli & Douglas Pulleyblank (1994): *Grounded Phonology*. Cambridge, Mass.: MIT Press.

Links to this page

- [Create tongue-root grammar...](#)
- [OT learning 3.2. Data from another grammar](#)
- [OTGrammar_tongueRoot](#)

© ppgb, October 21, 1997

Boersma & Hayes (2001)

Paul Boersma & Bruce Hayes (2001): "Empirical tests of the Gradual Learning Algorithm." *Linguistic Inquiry* **32**: 45-86.

Links to this page

- [OT learning 6. Shortcut to OT learning](#)
-

© ppgb, May 11, 2002

McCarthy & Prince (1995)

John J. McCarthy & Alan Prince (1995): Faithfulness and reduplicative identity. In Jill Beckman, Laura Walsh Dickey & Suzanne Urbanczyk (eds.), *Papers in Optimality Theory. University of Massachusetts Occasional Papers* 18. Amherst, Mass.: Graduate Linguistic Student Association. pp. 249–384. [Rutgers Optimality Archive 60, <http://roa.rutgers.edu>]

Links to this page

- OT learning 7. Learning from overt forms
-

© ppgb, November 5, 2002

Distributions: To Strings...

A command to create a Strings object from every selected Distributions object.

Arguments

Column number

the column (in the **Distributions** object) that contains the distribution that you are interested in. Often the **Distributions** object will only contain a single distribution, so this argument will often be 1. If the **Distributions** object contains nine distributions, specify any number between 1 and 9.

Number of strings

the number of times a string will be drawn from the chosen distribution. This is the number of strings that the resulting Strings object is going to contain.

Behaviour

Every string in the resulting **Strings** object will be a row label of the **Distributions** object. The number in each row at the specified column will be considered the relative frequency of occurrence of that row.

Example. Suppose we have the following **Distributions**:

```
File type = "ooTextFile"
Object class = "Distributions"
2 (number of columns)
    "English" "French" (column labels)
3 (number of rows)
"the" 108 1.5
"a" 58.1 33
"pour" 0.7 15.5
```

If you set *Column* to 1 and *Number of strings* to 1000, you will get a Strings object with approximately 647 occurrences of "the", 348 occurrences of "a", and 4 occurrences of "pour". If you had set *Column* to 2 ("French"), you would have gotten about 30 times "the", 660 times "a", and 310 times "pour". The actual numbers will vary because the choice of a string will not depend on previous choices.

Links to this page

- OT learning 7. Learning from overt forms
- OTAnyGrammar examples

© ppgb, October 22, 1997

OTAnyGrammar: Generate one input

A command to write one randomly generated input string from a selected OTAnyGrammar to the Info window.

A one-string version of OTAnyGrammar: Generate inputs....

Links to this page

- [OTAnyGrammar examples](#)

© *ppgb*, October 23, 1997

OTAnyGrammar: Sort...

A command to sort an OTAnyGrammar by the disharmonies (effective rankings) of its constraints. The disharmonies are computed anew from the ranking values of the constraints and a global ranking spreading.

Argument

Noise

a noise factor that is used to compute the disharmony of each constraint C_i according to

$$\text{disharmony}_i = \text{ranking}_i + (\text{Noise}) \cdot \mathbf{z}$$

where \mathbf{z} is a Gaussian random variable with zero mean and unit variance.

Behaviour

The disharmonies are computed anew from the ranking values of the constraints and a global ranking spreading.

If two constraints have equal disharmonies (which can easily occur if the ranking spreading is zero), the result will randomly vary each time you choose **Sort...**

Example

1. Choose Create tongue-root method grammar... and specify a nine-constraint grammar with Wolof ranking.
2. Click **Edit**. You see that the constraints *[rtr / mid] and *[atr / mid] have equal disharmonies.
2. Choose **Sort...** several times (with a *ranking spreading* of zero) and see the equally-ranked constraints swap places.

Links to this page

- [OTAnyGrammar examples](#)
-

© ppgb, December 24, 1997

OTAnyGrammar: Input to output...

A command to write an output string, as computed from an input string by a selected OTAnyGrammar, to the Info window.

A one-string version of OTAnyGrammar & Strings: Inputs to outputs...

Links to this page

- [OTAnyGrammar examples](#)

© *ppgb*, October 23, 1997

OTAnyGrammar & Strings: Inputs to outputs...

An action that creates a Strings object from a selected OTAnyGrammar (or subclass thereof) and a selected Strings.

The selected **Strings** object is considered as a list of inputs to the **OTAnyGrammar**.

The resulting **Strings** object will contain the output string of the grammar for each of the input strings.

Links to this page

- [OTAnyGrammar examples](#)
- [OTAnyGrammar: Input to output...](#)

© *ppgb*, October 23, 1997

Create tongue-root method grammar...

A command for creating an OTGrammar_tongueRoot object.

For the meanings of the constraints, see OTGrammar_tongueRoot.

Links to this page

- [OTAnyGrammar examples](#)
- [OTAnyGrammar: Sort...](#)

© *ppgb*, December 23, 1998

OTAnyGrammar & Strings: Learn output (T&S)

- OTAnyGrammar examples
-

© *ppgb*, October 24, 1997

OTAnyGrammar & Strings: Learn output (GLA)...

- OTAnyGrammar examples
-

© *ppgb*, October 24, 1997

Hayes & MacEachern (1998)

Bruce P. Hayes & Margaret MacEachern (1998): "Quatrain form in English folk verse", *Language* **74**: 473-507.

Links to this page

- [OTAnyGrammar examples](#)
-

© ppgb, December 19, 1998

Discriminant & SSCP: Project

A command to project the selected SSCP object on the eigenspace defined by the selected Discriminant object.

Further details can be found in *Eigen & SSCP: Project*

© *djmw*, March 13, 2002

Discriminant: Extract pooled within-groups SSCP

Extract the pooled within-group SSCP from the selected Discriminant object.

© *djmw*, March 14, 2002

Discriminant: Extract within-group SSCP...

Extract the SSCP for group *index* from the selected Discriminant object.

© *djmw*, March 14, 2002

Eigen & SSCP: Project

A command to project the SSCP object onto the eigenspace of the Eigen object.

Behaviour

Transform the SSCP object as if it was calculated in a coordinate system given by the eigenvectors of the Eigen object. This can be done as follows:

$$S_t = E' S E, \text{ where}$$

where E' is the transpose of the matrix with eigenvectors E , S is the square matrix with sums of squares and crossproducts, and S_t the newly created square matrix. The dimension of S_t may be smaller than the dimension of S .

Links to this page

- [Discriminant & SSCP: Project](#)
- [PCA & Covariance: Project](#)
- [PCA & SSCP: Project](#)

© *djmw*, March 28, 2002

PCA & SSCP: Project

A command to project the SSCP object onto the eigenspace of the PCA object.

Further details can be found in *Eigen & SSCP: Project*.

© *djmw*, February 25, 2004

SSCP & TableOfReal: Extract quantile range...

Extract those rows from the selected TableOfReal object whose Mahalanobis distance, with respect to the selected SSCP object, are within the quantile range.

© *djmw*, February 25, 2004

SSCP: Draw sigma ellipse...

A command to draw for the selected SSCP an ellipse that covers a part of the multivariate data.

Arguments

Number of sigmas
determines the data coverage.

© *djmw, February 22, 1999*

SSCP: Get confidence ellipse area...

A command to query the selected SSCP object for the area of a confidence ellipse.

Algorithm

The algorithm proceeds as follows:

1. The four array elements in the SSCP-matrix that correspond to the chosen dimensions are copied into a two-dimensional matrix S (symmetric of course).
2. The eigenvalues of S are determined, call them s_1 and s_2 .
3. The lengths l_1 and l_2 of the two axes of the ellipse can be obtained as (see for example Johnson (1998), page 410):

$$l_i = scaleFactor \cdot \sqrt{s_i},$$

where

$$scaleFactor = \sqrt{f \cdot p \cdot (n - 1) / (n \cdot (n - p))},$$

in which $f = \text{invFisherQ}(1 - confidenceLevel, p, n - p)$, where p is the `numberOfRows` from the SSCP object and n the `numberOfObservations`.

4. The area of the ellipse will be $\pi \cdot l_1 \cdot l_2$.

Links to this page

- Discriminant: Get confidence ellipse area...

© djmw, May 25, 2000

SSCP: Get fraction variation...

A command to ask the selected SSCP object for the fraction of the total variation that is accounted for by the selected dimension(s).

Further details can be found in Covariance: Get fraction variance....

© *djmw*, February 10, 2004

SSCP: To CCA...

A command that creates a canonical correlation object from the selected SSCP object.

Arguments

Dimension of dependent variate (ny)

defines a partition of the square $n \times n$ SSCP matrix S into the parts S_{yy} of dimension $ny \times ny$, S_{xx} of dimension $nx \times nx$, and the parts S_{yx} and S_{xy} of dimensions $nx \times ny$ and $ny \times nx$, respectively.

Algorithm

The partition for the square SSCP-matrix is as follows:

[sorry, no pictures yet in the web version of this manual]

The canonical correlation equations we have to solve are:

$$(1) (S_{yx} S_{xx}^{-1} S_{yx}' - \lambda S_{yy})\mathbf{y} = \mathbf{0}$$

$$(2) (S_{yx}' S_{yy}^{-1} S_{yx} - \lambda S_{xx})\mathbf{x} = \mathbf{0}$$

where S_{yy} [$ny \times ny$] and S_{xx} [$nx \times nx$] are symmetric, positive definite matrices belonging to the dependent and the independent variables, respectively.

These two equations are not independent and we will show that both equations have the same eigenvalues and that the eigenvectors \mathbf{x} for equation (2) can be obtained from the eigenvectors \mathbf{y} of equation (1).

We can solve equation (1) in several ways, however, the numerically stablest algorithm is probably by performing first a Cholesky decomposition of S_{xx} and S_{yy} , followed by a generalized singular value decomposition. The algorithm goes as follows:

The Cholesky decompositions ("square roots") of S_{yy} and S_{xx} are:

$$S_{yy} = \mathbf{U}' \mathbf{U} \text{ and } S_{xx} = \mathbf{H}' \mathbf{H},$$

where \mathbf{U} and \mathbf{H} are upper triangular matrices. From these decompositions, the inverse for S_{xx}^{-1} is easily computed. Let \mathbf{K} be the inverse of \mathbf{H} , then we can write:

$$S_{xx}^{-1} = \mathbf{K} \mathbf{K}'.$$

We next substitute in equation (1) and rewrite as:

$$((\mathbf{K}'\mathbf{S}_{yx}')' (\mathbf{K}'\mathbf{S}_{yx}') - \lambda \mathbf{U}' \mathbf{U})\mathbf{x} = 0$$

This equation can be solved for eigenvalues and eigenvectors by the generalized singular value decomposition because it is of the form $\mathbf{A}'\mathbf{A} - \lambda \mathbf{B}'\mathbf{B}$.

Now, given the solution for equation (1) we can find the solution for equation (2) by first multiplying (1) from the left with $\mathbf{S}_{yx}'\mathbf{S}_{yy}^{-1}$, resulting in:

$$(\mathbf{S}_{yx}'\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}\mathbf{S}_{xx}^{-1}\mathbf{S}_{yx}' - \lambda \mathbf{S}_{yx}') \mathbf{y} = 0$$

Now we split of the term $\mathbf{S}_{xx}^{-1}\mathbf{S}_{yx}'$ and obtain:

$$(\mathbf{S}_{yx}'\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx} - \lambda \mathbf{S}_{xx}) \mathbf{S}_{xx}^{-1}\mathbf{S}_{yx}' \mathbf{y} = 0$$

This equation is like equation (2) and it has therefor the same eigenvalues and eigenvectors. (We also proved this fact in the algorithmic section of TableOfReal: To CCA....)

The eigenvectors \mathbf{x} is now

$$\mathbf{x} = \mathbf{S}_{xx}^{-1}\mathbf{S}_{yx}' \mathbf{y}.$$

© djmw, November 3, 2003

SSCP: To Covariance...

A command that creates a Covariance object from each selected SSCP object.

Arguments

Number of constraints

determines the factor by which each entry in the SSCP-matrix is scaled to obtain the Covariance matrix.

Details

The relation between the numbers c_{ij} in the covariance matrix and the numbers s_{ij} in the originating SSCP matrix is:

$$c_{ij} = s_{ij} / (\text{numberOfObservations} - \text{numberOfConstraints})$$

Normally *numberOfConstraints* will equal 1. However, when the originating SSCP was the result of summing g SSCP objects, as is, for example, the case when you obtained the total within-groups SSCP from the individual group SSCP's, *numberOfConstraints* will equal g .

© djmw, May 24, 1999

TableOfReal: To SSCP...

Calculates Sums of Squares and Cross Products (SSCP) from the selected TableOfReal.

Algorithm

The sums of squares and cross products s_{ij} between the elements of columns i and j are calculated as:

$$s_{ij} = \sum_k (x_{ki} - \text{mean}_i)(x_{kj} - \text{mean}_j),$$

where x_{mn} is the element m in column n and mean_n is the mean of column n .

© djmw, February 18, 1999

Covariance & TableOfReal: Extract quantile range...

Extract those rows from the selected TableOfReal object whose Mahalanobis distance, with respect to the selected Covariance object, are within the quantile range.

© *djmw*, February 25, 2004

Covariance: Difference

You can choose this command after selecting two objects of type Covariance.

We test the hypothesis that the samples that gave rise to the two covariance matrices \mathbf{M}_1 and \mathbf{M}_2 , were drawn from the same distribution. The test statistic is L' which is distributed as a χ^2 variate with $p(p+1)/2$ degrees of freedom.

$$L' = L \cdot (1 - (2p + 1 - 2 / (p + 1)) / (6 \cdot (N - 1))),$$

where,

$$L = (N - 1) \cdot (\ln \text{determinant}(\mathbf{M}_1) - \ln \text{determinant}(\mathbf{M}_2)) + \text{trace}(\mathbf{M}_2 \cdot \mathbf{M}_1^{-1}) - p,$$

p is dimension of covariance matrix and N is the number of observations underlying the covariance matrix.

For more details on this test, see e.g. page 292 of Morrison (1990).

© djmw, December 22, 1998

Covariance: Get fraction variance...

A command to ask the selected Covariance object for the fraction of the total variance that is accounted for by the selected dimension(s).

Arguments

From dimension, To dimension

define the range of components. By choosing both numbers equal, you get the fraction of the variance "explained" by that dimension.

Details

The total variance is the sum of the diagonal elements of the covariance matrix **C**, i.e., its trace. The fraction is defined as:

$$\sum_{i=from..to} C_{ii} / \sum_{i=1..numberOfRows} C_{ii}$$

Links to this page

- [PCA & TableOfReal: Get fraction variance...](#)
- [SSCP: Get fraction variation...](#)

© djmw, April 7, 2004

Covariance: Get significance of means difference...

Gets the level of significance for the *difference* of two means from the selected Covariance object being different from a hypothesized value.

Arguments

Index1, Index2

the positions of the two elements of the means vector whose difference is compared to the hypothesized difference.

Value

the hypothesized difference (μ).

Paired samples

determines whether we treat the two means as being dependent.

Equal variances

determines whether the distribution of the difference of the means is a Student t-distribution (see below).

Behaviour

This is Student's t-test for the significance of a difference of means. The test statistic is:

$$t = (\text{mean}_1 - \text{mean}_2 - \mu) \sqrt{N / s^2} \text{ with } ndf \text{ degrees of freedom.}$$

In the formula above mean_1 and mean_2 are the elements of the means vector, μ is the hypothesized difference and N is the number of observations. The value that we use for the (combined) variance s^2 is:

$$s^2 = \text{var}_1 + \text{var}_2 - 2 * \text{covar}_{12},$$

when the samples are *paired*, and

$$s^2 = \text{var}_1 + \text{var}_2$$

when they are not.

The var_1 and var_2 are the variance components for mean_1 and mean_2 , respectively, and covar_{12} is their covariance. When we have *paired samples* we assume that the two variances are not independent and their covariance is subtracted, otherwise their covariance is not taken into account. Degrees of freedom parameter ndf usually equals $2(N-1)$.

If the two variances are significantly different, the statistic t above is only *approximately* distributed as Student's t with degrees of freedom equal to:

$$ndf = (N-1) \cdot (var_1 + var_2)^2 / (var_1^2 + var_2^2).$$

The returned probability p will be the *two-sided* probability

$$p = 2 * \text{studentQ}(t, ndf)$$

A low probability p means that the difference is significant.

Links to this page

- T-test
-

© djmw, April 7, 2004

Covariance: Get significance of one mean...

Gets the level of significance for one mean from the selected Covariance object being different from a hypothesized mean.

Arguments

Index

the position of the element in the means vector (centroid) that you want to test.

Value

the hypothesized mean μ (see below).

Behaviour

This is the standard test on means when the variance is unknown. The test statistic is

$$t = (mean - \mu) \sqrt{N / s^2},$$

which has the Student t distribution with $ndf = N-1$ degrees of freedom.

In the formulas above, *mean* is the element of the mean vector at position *index*, μ is the hypothesized mean, N is the number of observations, s^2 is the variance at position [*index*][*index*] in the covariance matrix.

The returned probability p is the *two-sided* probability

$$p = 2 * \text{studentQ}(t, ndf)$$

A low probability p means that the difference is significant.

Links to this page

- T-test

Covariance: Get significance of one variance...

Gets the probability for one variance from the selected Covariance object being different from a hypothesized variance.

Arguments

Index

the position of the variance element.

Hypothesized variance

the hypothesized variance σ^2

Behaviour

The test statistic

$$\chi^2 = (N-1)s^2 / \sigma^2,$$

is distributed as a chi-squared variate with $ndf = N-1$ degrees of freedom.

The returned probability p will be

$$p = \text{chiSquareQ}(\chi^2, ndf)$$

© djmw, April 7, 2004

Covariance: Get significance of variance ratio...

Gets the probability for the ratio of two variances from the selected Covariance object being different from a hypothesized ratio.

Arguments

Index1, index2

determine the variances

Hypothesized ratio

the hypothesized ratio F

Behaviour

The test statistic

$$f = s_1^2 / s_2^2 / \text{ratio}$$

is distributed as Fisher's F distribution with $ndf_1 = N-1$ and $ndf_2 = N-1$ degrees of freedom for the numerator and denominator terms, respectively.

The returned probability p will be the *two-sided* probability

$$p = 2 * \text{fisherQ}(f, ndf_1, ndf_2)$$

If $s_2^2 > s_1^2$ we use $1/f$ to determine the probability.

© djmw, April 7, 2004

PCA & Covariance: Project

A command to project the Covariance object onto the eigenspace of the PCA object.

Further details can be found in Eigen & SSCP: Project.

Links to this page

- [PCA & TableOfReal: Get fraction variance...](#)

© *djmw*, February 25, 2004

TableOfReal: To Covariance

A command that creates a Covariance object from every selected TableOfReal object. The covariances are calculated between columns.

Algorithm

The covariance coefficients s_{ij} between the elements of columns i and j are defined as:

$$s_{ij} = \sum_k (x_{ki} - \text{mean}_i)(x_{kj} - \text{mean}_j) / (\text{numberOfObservations} - \text{numberOfConstraints}),$$

where x_{ki} is the element k in column i , mean_i is the mean of column i , $\text{numberOfObservations}$ equals the number of rows in the table, and $\text{numberOfConstraints}$ equals 1.

The actual calculation goes as follows

1. Centralize each column (subtract the mean).
2. Get its singular value decomposition $\mathbf{U} \mathbf{\Sigma} \mathbf{V}'$.
3. Form $\mathbf{S} = \mathbf{V} \mathbf{\Sigma} \mathbf{V}'$.
4. Divide all elements in \mathbf{S} by $(\text{numberOfObservations} - 1)$.

Links to this page

- [PCA & TableOfReal: Get fraction variance...](#)
- [T-test](#)

© djmw, January 17, 2002

CCA & Correlation: To TableOfReal (loadings)

Determine from the selected CCA and Correlation objects the correlations of the canonical variables with the original variables. These correlations are called *canonical factor loadings*, or also *structure correlation coefficients*.

© djmw, May 25, 2002

TableOfReal: To Correlation

A command that creates a (*Pearson*) Correlation object from every selected TableOfReal object. The correlations are calculated between columns.

Algorithm

The linear correlation coefficient r_{ij} (also called the *product moment correlation coefficient* or *Pearson's correlation coefficient*) between the elements of columns i and j is calculated as:

$$r_{ij} = \frac{\sum_k (x_{ki} - \text{mean}_i)(x_{kj} - \text{mean}_j)}{(\sqrt{\sum_k (x_{ki} - \text{mean}_i)^2}) \sqrt{\sum_k (x_{kj} - \text{mean}_j)^2}},$$

where x_{mn} is the element m in column n , and mean_n is the mean of column n .

© djmw, January 5, 2002

TableOfReal: To Correlation (rank)

A command that creates a (*Spearman rank-order*) Correlation object from every selected TableOfReal object. The correlations are calculated between columns.

Algorithm

The Spearman rank-order correlation coefficient r_{ij} between the elements of columns i and j is calculated as the linear correlation of the ranks:

$$r_{ij} = \sum_k (R_{ki} - Rmean_i) (R_{kj} - Rmean_j) / (\sqrt{\sum_k (R_{ki} - Rmean_i)^2} \sqrt{\sum_k (R_{kj} - Rmean_j)^2}),$$

where R_{mn} is the rank of element m in column n , and $Rmean_n$ is the mean of the ranks in column n .

© djmw, January 5, 2002

Eigen: Draw eigenvector...

A command to draw an eigenvector from the selected Eigen.

Arguments

Eigenvector number

determines the eigenvector to be drawn.

Component loadings

when on, the eigenvector is multiplied with the square root of the corresponding eigenvalue. (For PCA-analysis this means that you will draw the so called *component loading vector*. You will be able to compare quantitatively the elements in different component loading vectors because the i -th element in the j -th component loading vector gives the covariance between the i -th original variable and the j -th principal component.)

Element range

determine the first and last element of the vector that must be drawn.

Minimum, Maximum

determine the lower and upper bounds of the plot (choosing Maximum < Minimum will draw the *inverted* eigenvector).

Mark size, Mark string

determine size and type of the marks that will be drawn.

Garnish

determines whether a bounding box and margins will be drawn.

© djmw, April 7, 2004

PCA & Configuration: To TableOfReal (reconstruct)

A command to reconstruct a TableOfReal from the selected Configuration and PCA.

The TableOfReal is reconstructed from the eigenvectors of the PCA and elements of the Configuration are the weight factors:

$$t_i = \sum_k c_{ik} e_k,$$

where t_i is the i -th row in the resulting TableOfReal object, c_{ik} is the element at row i and column k in the Configuration object and e_k the k -th eigenvector from the PCA object.

Links to this page

- [PCA: To TableOfReal \(reconstruct 1\)...](#)

© *djmw*, January 8, 2003

PCA & TableOfReal: Get fraction variance...

A command to query the selected PCA and TableOfReal object for the explained fraction of the variance if the TableOfReal object were projected onto the PCA space.

Algorithm

1. The TableOfReal is converted to a Covariance object.
 2. The Covariance object is projected on the PCA eigenspace and the newly obtained projected Covariance object is queried for the fraction variance.
-

© *djmw*, March 24, 2004

PCA: Get eigenvalue...

A command to query the selected PCA for the i^{th} eigenvalue.

© *djmw*, February 25, 2004

PCA: Get eigenvector element...

A command to query the selected PCA for the j^{th} element of the i^{th} eigenvector.

© *djmw*, February 25, 2004

Read Matrix from raw text file...

A command to read a Matrix object from a file on disk.

File format

The file should contain each row of the matrix on a separate line. Within each row, the elements must be separated by spaces or tabs.

For instance, the following text file will be read as a Matrix with three rows and four columns:

```
0.19 3 245 123
18e-6 -3e18 0 0.0
1.5 2.5 3.5 4.5
```

The resulting Matrix will have the same domain and sampling as Matrices created with **Create simple Matrix...** In the above example, this means that the Matrix will have $x_{min} = 0.5$, $x_{max} = 4.5$, $n_x = 4$, $dx = 1.0$, $x_1 = 1.0$, $y_{min} = 0.5$, $y_{max} = 3.5$, $n_y = 3$, $dy = 1.0$, $y_1 = 1.0$.

Links to this page

- [TableOfReal](#)

© ppgb, March 22, 1998

CCA & TableOfReal: To TableOfReal (loadings)

Determine from the selected CCA and TableOfReal objects the correlations of the canonical variables with the original variables. These correlations are called *canonical factor loadings*, or also *structure correlation coefficients*.

© djmw, May 25, 2002

CCA & TableOfReal: To TableOfReal (scores)...

Determines the scores on the dependent and the independent canonical variates from the selected CCA and TableOfReal objects.

Arguments

Number of canonical correlations

determines the dimension, i.e., the number of elements of the resulting canonical score vectors. The newly created table will have twice this number of columns because we have calculated score vectors for the dependent and the independent variates.

Behaviour

The scores on the dependent set are determined as $\mathbf{T}_y \mathbf{Y}$, where \mathbf{T}_y is the dependent part in the table and \mathbf{Y} is a matrix with *numberOfCanonicalCorrelations* eigenvectors for the dependent variate.

The scores for the independent variates are then determined in an analogous way as $\mathbf{T}_x \mathbf{X}$.

The scores for the dependent data will be in the lower numbered columns, the scores for the independent part will be in the higher numbered columns of the newly created object.

Links to this page

- [Canonical correlation analysis](#)

© djmw, April 7, 2004

Confusion: To TableOfReal (marginals)

A new TableOfReal object is created from the selected Confusion object with one extra row and column.

The first element of the extra row will contain the sum of the confusions in the the first *column*, the first element of the extra column will contain the sum of the confusions in the the first *row*, etc... The bottom-right element will contain the sum of all confusions.

© *djmw*, October 31, 2001

Eigen & TableOfReal: Project...

A command to project the rows of the TableOfReal object onto the eigenspace of the Eigen object.

Arguments

Number of dimensions,
defines the dimension, i.e., the number of columns, of the resulting object.

Algorithm

Project each row of the TableOfReal on the coordinate system given by the eigenvectors of the Eigen object. This can be done as follows:

$$y_{ij} = \sum_{k=1..numberOfColumns} e_{jk} x_{ik}, \text{ where}$$

e_{jk} is the k -th element of the j -th eigenvector, x_{ik} is the k -th element of the i -th row and y_{ij} is the j -th element at the i -th row of the matrix part of the resulting object.

Links to this page

- [Discriminant & TableOfReal: To Configuration...](#)
- [PCA & TableOfReal: To Configuration...](#)

© djmw, April 7, 2004

FFNet: Extract weights...

Extract all the weights, from all the units in the specified layer of the selected FFNet, to a TableOfReal.

Arguments

Layer number
determines the layer.

Behaviour

The weights will be arranged in the TableOfReal as follows:

The table columns will be indexed by the unit numbers in the selected layer, while the rows will be indexed by the unit numbers from the previous layer. There will be one extra row to accommodate the bias weights. The rows and columns are labelled with layer number and unit number as " L_i-j ", where i is the layer number and j the unit number from that layer. The layer number for the rows is one less than the layer number in the columns. The last row is labelled as "Bias".

© *djmw*, April 22, 2004

PCA: To TableOfReal (reconstruct 1)...

A command to reconstruct a single data item. The result is stored as a TableOfReal with only one row.

Arguments

Coefficients

the weight for the eigenvectors.

The algorithm is explained in PCA & Configuration: To TableOfReal (reconstruct).

© djmw, January 8, 2003

TableOfReal: Centre columns

A command that centres the columns in the selected TableOfReal objects.

Algorithm

The new values in the table, x'_{ij} , will be:

$$x'_{ij} = x_{ij} - x_{.j},$$

where

$$x_{.j} = \sum_{i=1..numberOfRows} x_{ij} / numberOfRows,$$

the average of column j .

© djmw, April 22, 1998

TableOfReal: Centre rows

A command that centres the rows in the selected TableOfReal objects.

Algorithm

The new values in the table, x'_{ij} , will be:

$$x'_{ij} = x_{ij} - x_{i.},$$

where

$$x_{i.} = \sum_{j=1..numberOfColumns} x_{ij} / numberOfColumns,$$

the average of row i .

© djmw, April 22, 1998

TableOfReal: Change column labels...

Changes the column labels of the selected TableOfReal object according to the specification in the search and replace fields.

Both search and replace fields may contain Regular expressions. The *Replace limit* parameter limits the number of replaces that may occur within each label.

© *djmw*, August 22, 2001

TableOfReal: Change row labels...

Changes the row labels of the selected TableOfReal object according to the specification in the search and replace fields.

Both search and replace fields may contain Regular expressions. The *Replace limit* parameter limits the number of replaces that may occur within each label.

© *djmw*, August 22, 2001

TableOfReal: Draw biplot...

A command to draw a biplot for each column in the selected TableOfReal object.

Arguments

Xmin, Xmax, Ymin, Ymax

determine the drawing boundaries.

Split factor

determines the weighing of the row and column structure (see below).

Behaviour

1. Get the singular value decomposition $\mathbf{U} \mathbf{\Sigma} \mathbf{V}'$ of the table.
2. Calculate weighing factors λ for row and columns

$$\lambda_{r,1} = \sigma_1^{splitFactor}$$

$$\lambda_{c,1} = \sigma_1^{1-splitFactor}$$

$$\lambda_{r,2} = \sigma_2^{splitFactor}$$

$$\lambda_{c,2} = \sigma_2^{1-splitFactor}$$

where σ_1 and σ_2 are the first and the second singular values

3. For the rows (i from 1..*numberOfRows*) form:

$$xr_i = U_{i1} \lambda_{r,1}$$

$$yr_i = U_{i2} \lambda_{r,2}$$

4. For the columns (i from 1..*numberOfColumns*) form:

$$xc_i = V_{i1} \lambda_{c,1}$$

$$yc_i = V_{i2} \lambda_{c,2}$$

5. Plot the points (x_{r_i}, y_{r_i}) and (x_{c_i}, y_{c_i}) in the same figure with the corresponding row and column labels.

© *djmw*, June 3, 2002

TableOfReal: Draw rows as histogram...

A command to draw a histogram from the rows in the selected TableOfReal object.

The histogram will consist of *groups* of bars. The number of groups will be determined by the number of selected columns from the table, while the number of bars within each group will be determined from the number of selected rows.

Arguments

Row numbers and *Column range*,

determine the part of the table that you want to draw. The column range determines the number of bars that you want to draw for each row selected by the *Row numbers* argument.

Ymin and *Ymax*

the drawing boundaries.

The following arguments are all relative to the width of a bar in the histogram.

Horizontal offset

the offset from the left and right margin.

Distance between bar groups

the distance between each group, i.e., the distance between the right side of the last bar in a group to the left side of the first bar in the next group.

Distance between bars

the distance between the bars in a group.

Grey values

the grey values of the bars in a group.

Bar positioning

If you want to put the labels yourself you will need the following information.

The width of a bar is determined as follows:

$$width = 1 / (nc \cdot nr + 2 \cdot hoffset + (nc - 1) \cdot intergroup + nc \cdot (nr - 1) \cdot interbar),$$

where *nc* is the number of columns (groups) to draw, *nr* is the number of rows to draw (the number of bars within a group), *hoffset* is the horizontal offset, *intergroup* the distance between each group and *interbar* the distance between the bars within a group.

The spacing between the bars drawn from a row:

$$dx = (intergroup + nr + (nr - 1) \cdot interbar) * width$$

The first bar for the k -th row starts at:

$$x1 = \text{offset} \cdot \text{width} + (i - 1) \cdot (1 + \text{interbar}) \cdot \text{width}$$

© djmw, June 19, 2003

TableOfReal: Get table norm

A command that returns the norm of the selected TableOfReal object.

Algorithm

Returns: $\text{sqrt}(\sum_{i=1..numberOfRows} \sum_{j=1..numberOfColumns} x_{ij}^2)$.

© djmw, April 22, 1998

TableOfReal: Normalize columns...

A command that normalizes the columns in the selected TableOfReal objects.

Argument

Norm

determines the sum of the squared elements in each column after normalization.

Algorithm

All elements x_{ij} in each column $j=1..numberOfColumns$ will be multiplied by $\text{sqrt}(norm / \sum_{i=1..numberOfRows} x_{ij}^2)$.

© *djmw*, April 22, 1998

TableOfReal: Normalize rows...

A command that normalizes the rows in the selected TableOfReal objects.

Argument

Norm

determines the sum of the squared elements in each row after normalization.

Algorithm

All elements x_{ij} in each row $i=1..numberOfRows$ will be multiplied by $\text{sqrt}(norm / \sum_{j=1..numberOfColumns} x_{ij}^2)$.

© *djmw*, April 22, 1998

TableOfReal: Normalize table...

A command that normalizes the elements in the selected TableOfReal objects.

Argument

Norm

determines the sum of the squared elements after normalization.

Algorithm

All elements x_{ij} will be multiplied by $\text{sqrt}(norm / \sum_{i=1..numberOfRows} \sum_{j=1..numberOfColumns} x_{ij}^2)$.

© djmw, April 22, 1998

TableOfReal: Select columns where row...

Copy columns from the selected TableOfReal object to a new TableOfReal object.

Arguments

Columns

defines the indices of the columns to be selected. Ranges can be defined with a colon ":". Columns will be selected in the specified order.

Row condition

specifies a condition for the selection of rows. If the condition evaluates as *true* for a particular row, the selected elements in this row will be copied. See Matrix: Formula... for the kind of expressions that can be used here.

Examples

```
Select columns where row... "1 2 3" 1
Select columns where row... "1 : 3" 1
```

Two alternative expressions to copy the first three columns to a new table with the same number of rows.

```
Select columns where row... "3 : 1" 1
```

Copy the first three columns to a new table with the same number of rows. The new table will have the 3 columns reversed.

```
Select columns where row... "1:6 9:11" self[row,8]>0
```

Copy the first six columns and columns 9, 10, and 11 to a new table. Copy only elements from rows where the element in column 8 is greater than zero.

© djmw, May 2, 2002

TableOfReal: To CCA...

A command that creates a CCA object from the selected TableOfReal object.

Arguments

Dimension of dependent variate (ny)

defines the partition of the table into the two parts whose correlations will be determined. The first ny columns must be the dependent part, the rest of the columns will be interpreted as the independent part (nx columns). In general nx must be larger than or equal to ny .

Behaviour

Calculates canonical correlations between the *dependent* and the *independent* parts of the table. The corresponding canonical coefficients are also determined.

Algorithm

The canonical correlation equations for two data sets \mathbf{T}_y [$n \times p$] and \mathbf{T}_x [$n \times q$] are:

$$(1) (\mathbf{S}_{yx} \mathbf{S}_{xx}^{-1} \mathbf{S}_{yx}' - \lambda \mathbf{S}_{yy}) \mathbf{y} = \mathbf{0}$$

$$(2) (\mathbf{S}_{yx}' \mathbf{S}_{yy}^{-1} \mathbf{S}_{yx} - \lambda \mathbf{S}_{xx}) \mathbf{x} = \mathbf{0}$$

where \mathbf{S}_{yy} [$p \times p$] and \mathbf{S}_{xx} [$q \times q$] are the covariance matrices of data sets \mathbf{T}_y and \mathbf{T}_x , respectively, \mathbf{S}_{yx} [$p \times q$] is the matrix of covariances between data sets \mathbf{T}_y and \mathbf{T}_x , and the vectors \mathbf{y} and \mathbf{x} are the *canonical weights* or the *canonical function coefficients* for the dependent and the independent data, respectively. In terms of the (dependent) data set \mathbf{T}_y and the (independent) data set \mathbf{T}_x , these covariances can be written as:

$$\mathbf{S}_{yy} = \mathbf{T}_y' \mathbf{T}_y, \mathbf{S}_{yx} = \mathbf{T}_y' \mathbf{T}_x \text{ and } \mathbf{S}_{xx} = \mathbf{T}_x' \mathbf{T}_x.$$

The following singular value decompositions

$$\mathbf{T}_y = \mathbf{U}_y \mathbf{D}_y \mathbf{V}_y' \text{ and } \mathbf{T}_x = \mathbf{U}_x \mathbf{D}_x \mathbf{V}_x'$$

transform equation (1) above into:

$$(3) (\mathbf{V}_y \mathbf{D}_y \mathbf{U}_y' \mathbf{U}_x \mathbf{U}_x' \mathbf{U}_y \mathbf{D}_y \mathbf{V}_y' - \lambda \mathbf{V}_y \mathbf{D}_y \mathbf{D}_y \mathbf{V}_y') \mathbf{y} = \mathbf{0}$$

where we used the fact that:

$$\mathbf{S}_{xx}^{-1} = \mathbf{V}_x \mathbf{D}_x^{-2} \mathbf{V}_x'$$

Equation (3) can be simplified by multiplication from the left by $\mathbf{D}_y^{-1} \mathbf{V}_y'$ to:

$$(4) ((\mathbf{U}_x' \mathbf{U}_y)' (\mathbf{U}_x' \mathbf{U}_y) - \lambda \mathbf{I}) \mathbf{D}_y \mathbf{V}_y' \mathbf{y} = \mathbf{0}$$

This equation can, finally, be solved by a substitution of the s.v.d of $\mathbf{U}_x' \mathbf{U}_y = \mathbf{U} \mathbf{D} \mathbf{V}'$ into (4). This results in

$$(5) (\mathbf{D}^2 - \lambda \mathbf{I}) \mathbf{V}' \mathbf{D}_y \mathbf{V}_y' \mathbf{y} = \mathbf{0}$$

In an analogous way we can reduce eigenequation (2) to:

$$(6) (\mathbf{D}^2 - \lambda \mathbf{I}) \mathbf{U}' \mathbf{D}_x \mathbf{V}_x' \mathbf{x} = \mathbf{0}$$

From (5) and (6) we deduce that the eigenvalues in both equations are equal to the squared singular values of the product matrix $\mathbf{U}_x' \mathbf{U}_y$. These singular values are also called *canonical correlation coefficients*. The eigenvectors \mathbf{y} and \mathbf{x} can be obtained from the columns of the following matrices \mathbf{Y} and \mathbf{X} :

$$\mathbf{Y} = \mathbf{V}_y \mathbf{D}_y^{-1} \mathbf{V}$$

$$\mathbf{X} = \mathbf{V}_x \mathbf{D}_x^{-1} \mathbf{U}$$

For example, when the vector \mathbf{y} equals the first column of \mathbf{Y} and the vector \mathbf{x} equals the first column of \mathbf{X} , then the vectors $\mathbf{u} = \mathbf{T}_y \mathbf{y}$ and $\mathbf{v} = \mathbf{T}_x \mathbf{x}$ are the linear combinations from \mathbf{T}_y and \mathbf{T}_x that have maximum correlation. Their correlation coefficient equals the first canonical correlation coefficient.

Links to this page

- Canonical correlation analysis
- SSCP: To CCA...

© djmw, April 24, 2002

TableOfReal: To Pattern and Categories...

Extracts a Pattern and a Categories from the selected TableOfReal.

The selected rows and columns are copied into the Pattern and the corresponding row labels are copied into a Categories.

© *djmw*, April 29, 2004

Morrison (1990)

D.F. Morrison (1990), *Multivariate Statistical Methods*, McGraw-Hill, New York.

Links to this page

- [Covariance: Difference](#)
- [PCA: Get equality of eigenvalues...](#)

© djmw, January 23, 1998

Create Configuration...

A command to create a Configuration with the specified number of points and number of dimensions. The location of the points will be determined by the formula (see Formulas for more information about possible formulas).

© *djmw*, April 13, 1998

Dissimilarity: To Configuration (i-spline mds)...

A command that creates a Configuration object from a Dissimilarity object.

Dissimilarities δ_{ij} and disparities d'_{ij} will be related by a spline function:

$$d'_{ij} = \sum_{k=1..(numberOfInteriorKnots+order)} \text{spline}_k(\text{knots}, \text{order}, \delta_{ij}),$$

where $\text{spline}_k(\cdot)$ is the value of the k^{th} I-spline of order $order$ and knot sequence $knot$ evaluated at δ_{ij} .

Settings

Number of dimensions

determines the dimensionality of the configuration.

Number of interior knots

determines the number of segment boundaries. Each interior knot is the boundary between two segments. The splines in each segment will be joined as continuously as possible.

Order of I-spline

The order of the polynomial basis of the I-spline.

Finding the optimal Configuration involves a minimization process:

Tolerance

When successive values for the stress differ by less than *Tolerance*, the minimization process stops.

Maximum number of iterations

Minimization stops after this number of iterations has been reached.

Number of repetitions

If chosen larger than 1, the minimization process will be repeated, each time with another random start configuration. The configuration that results in minimum stress, will be saved.

Hints

If *numberOfInteriorKnots* is zero, polynomial regression will be performed. Therefore, the combination *numberOfInteriorKnots* = 0 and *order* = 1 also gives interval scaling (in fact, it is the implementation in this program).

In the limit when *order* = 0 and *numberOfInteriorKnots* = *numberOfDissimilarities*, monotone regression is performed.

Links to this page

- [Dissimilarity & Configuration & Weight: To Configuration...](#)
 - [Dissimilarity & Weight: To Configuration...](#)
 - [Dissimilarity & Weight: To Configuration...](#)
-

© *djmw*, April 7, 2004

Dissimilarity: To Configuration (interval mds)...

A command that creates a Configuration object from a Dissimilarity object.

The disparities d'_{ij} will be obtained from dissimilarities δ_{ij} according to:

$$d'_{ij} = a + b \cdot \delta_{ij}$$

Links to this page

- [Dissimilarity & Configuration & Weight: To Configuration...](#)
- [Dissimilarity & Weight: To Configuration...](#)
- [Dissimilarity & Weight: To Configuration...](#)

© djmw, January 5, 1998

Dissimilarity: To Configuration (ratio mds)...

A command that creates a Configuration object from a Dissimilarity object.

The disparities d'_{ij} will be obtained from dissimilarities δ_{ij} according to:

$$d'_{ij} = b \cdot \delta_{ij}$$

Links to this page

- [Dissimilarity & Configuration & Weight: To Configuration...](#)
- [Dissimilarity & Weight: To Configuration...](#)
- [Dissimilarity & Weight: To Configuration...](#)

© djmw, January 5, 1998

Dissimilarity: To Configuration (absolute mds)...

A command that creates a Configuration object from a Dissimilarity object.

The disparities d'_{ij} will be obtained from dissimilarities δ_{ij} according to:

$$d'_{ij} = \delta_{ij}$$

Links to this page

- [Dissimilarity & Configuration & Weight: To Configuration...](#)
- [Dissimilarity & Weight: To Configuration...](#)
- [Dissimilarity & Weight: To Configuration...](#)

© *djmw*, January 5, 1998

Dissimilarity & Weight: To Configuration...

A command that creates a Configuration object from a Dissimilarity object. With the selected Weight object the influence of each dissimilarity on stress can be influenced.

Settings

May be different and depend on the representation function, i.e. the scale of measurement.

- Dissimilarity: To Configuration (monotone mds)...
- Dissimilarity: To Configuration (i-spline mds)...
- Dissimilarity: To Configuration (interval mds)...
- Dissimilarity: To Configuration (ratio mds)...
- Dissimilarity: To Configuration (absolute mds)...

© *djmw*, April 7, 2004

Dissimilarity & Configuration: To Configuration (monotone mds)...

A command that creates a Configuration object from a Dissimilarity object. The selected Configuration object serves as a starting configuration for the minimization process.

© *djmw*, January 19, 1998

Dissimilarity & Configuration: To Configuration (i-spline mds)...

A command that creates a Configuration object from a Dissimilarity object. The selected Configuration object serves as a starting configuration for the minimization process.

© *djmw*, January 19, 1998

Dissimilarity & Configuration: To Configuration (interval mds)...

A command that creates a Configuration object from a Dissimilarity object. The selected Configuration object serves as a starting configuration for the minimization process.

© *djmw*, January 19, 1998

Dissimilarity & Configuration: To Configuration (ratio mds)...

A command that creates a Configuration object from a Dissimilarity object. The selected Configuration object serves as a starting configuration for the minimization process.

© *djmw*, January 19, 1998

Dissimilarity & Configuration: To Configuration (absolute mds)...

A command that creates a Configuration object from a Dissimilarity object. The selected Configuration object serves as a starting configuration for the minimization process.

© *djmw*, January 19, 1998

Configuration: To Configuration (varimax)...

A command that rotates the selected Configuration object to a new Configuration object whose coordinates have maximum *squared* variance.

Settings

Normalize rows

when selected, the distances of all points to the origin will be made equal before iteration starts. We remember these scale factors and restore the original distances after the iteration process has stopped.

Quartimax

when selected, the sum of fourth powers, normalized or raw, will be maximized.

Maximum number of iterations

sets a limit to the number of iterations. One iteration consists of *numberOfDimensions* · (*numberOfDimensions*-1)/2 planar rotations of all pairs of dimensions.

Tolerance

also determines when the iteration stops. This happens if $|v_i - v_{i+1}| < \textit{Tolerance} \cdot v_i$, where v_i is the squared variance for the i^{th} iteration.

The iteration process stops when either the *maximum number of iterations* is reached or the *tolerance* criterion is met, whichever one is first.

Algorithm

The Varimax rotation procedure was first proposed by Kaiser (1958). Given a *numberOfPoints* × *numberOfDimensions* configuration **A**, the procedure tries to find an orthonormal rotation matrix **T** such that the sum of variances of the columns of **B*****B** is a maximum, where **B** = **A****T** and * is the element wise (Hadamard) product of matrices. A direct solution for the optimal **T** is not available, except for the case when *numberOfDimensions* equals two. Kaiser suggested an iterative algorithm based on planar rotations, i.e., alternate rotations of all pairs of columns of **A**.

However, this procedure is not without problems: the varimax function may have stationary points that are not even local maxima. We have incorporated an algorithm of Ten Berge (1995) that prevents this unpleasant situation from happening.

© djmw, April 7, 2004

Configuration & AffineTransform: To Configuration

A command that transforms the selected Configuration to a new Configuration object according to the specifications in the selected AffineTransform object.

© *djmw*, October 8, 2001

Configuration & Procrustus: To Configuration

A command that transforms the selected Configuration to a new Configuration object according to the specifications in the selected Procrustus object.

© *djmw*, October 8, 2001

Configuration: Randomize

Changes all coordinates of the points x_i in the Configuration according to:

$$x_{ij} = \text{randomUniform}(-1, 1)$$

© *djmw, December 1, 1997*

Configuration: Rotate (pc)

Rotates the Configuration to principal directions. The principal directions correspond to the principal components.

© *djmw*, December 1, 1997

Configuration: Rotate...

Rotates the Configuration in a plane around the origin.

Settings

Dimension 1, Dimension 2

the dimensions that span the plane.

Angle

the clockwise rotation angle in degrees w.r.t. the direction vector of the lowest of the two dimensions.

© *djmw, April 7, 2004*

Configuration: Invert dimension...

Inverts one dimension of a Configuration.

Settings

Dimension

the dimensions that has to be inverted.

Behaviour

For all points $i=1..numberOfPoints$: if $j == dimension$ then $x_{ij} = -x_{ij}$.

© djmw, April 7, 2004

Configuration: Normalize...

Normalizes the selected Configuration.

Settings

Sum of squares (default: 0.0)

The desired value for the variance.

Each dimension separately

When on, the sum of squares in each dimension (column) will be scaled to *sumOfSquares*. When off, the sum of squares of all the matrix elements will equal *sumOfSquares*.

With the default value (0.0) for *sumOfSquares*, and *eachDimensionSeparately* chosen, an INDSCAL-like normalization is applied: the sum of squares for each column is scaled to equal 1.0. When *eachDimensionSeparately* is off, a Kruskal-like normalization is applied: the sum of squares of the whole matrix is scaled equal to *numberOfRows*.

Behaviour

Before the normalization will be applied, however, we first translate the centre of the configuration to the origin by subtracting the mean for each dimension. The sum of squares than equals variance.

© djmw, April 7, 2004

Configuration: Centralize

Makes the centre of the selected Configuration equal to the origin.

© *djmw*, April 13, 1998

Configuration: To Configuration (procrustus)

A command that transforms the second selected Configuration object to match the first selected Configuration object as closely as possible. This problem of fitting one configuration (testee) to another (target) as closely as possible is called the Procrustus problem. We use a special procrustus transform algorithm that does not mutilate or distort the testee configuration.

Both Configuration objects must have the same dimensions.

© *djmw*, December 19, 1997

Configuration: To Distance

A command that computes a Distance object for each selected Configuration.

Algorithm

The distance d_{ij} between objects i and j is calculated as:

$$d_{ij} = d_{ji} = (\sum_{k=1..numberOfDimensions} |x_{ik} - x_{jk}|^2)^{1/2}$$

© djmw, December 7, 1997

Configuration: To Similarity (cc)

A command that create one Similarity object from the selected Configuration objects.

In the Similarity object entry s_{ij} equals the congruence coefficient for the i -th and j -th selected Configuration object.

All Configuration objects must have the same number of points and the same dimensions.

© *djmw*, January 30, 1998

Configurations: To AffineTransform (congruence)...

A command that creates an AffineTransform object from two selected Configuration objects.

We calculate the affine transform that transforms the second selected Configuration object to match the first selected Configuration object as closely as possible. The degree of proportionality is the congruence between corresponding dimensions.

Settings

Maximum number of iterations

sets a limit to the number of iterations.

Tolerance

also determines when the iteration stops. This happens if $|f(\mathbf{T}_i) - f(\mathbf{T}_{i+1})| < \textit{Tolerance} \cdot f(\mathbf{T}_i)$, where $f(\mathbf{T}_i)$ is the sum of the congruences for the i^{th} iteration (see below).

The iteration process stops when either the *maximum number of iterations* is reached or the *tolerance* criterion is met, whichever one is first.

Algorithm

Sometimes the criterion used in a procrustus transform is too restrictive for comparing two configurations. This criterion is only zero when the positions in the rotated configuration (\mathbf{AT}) equal the positions in the other configuration (\mathbf{B}). Brokken (1983) proposed an algorithm to maximize instead the sum of congruences between corresponding dimensions of \mathbf{AT} and \mathbf{B} . Specifically he proposed to maximize

$$f(\mathbf{T}) = \sum_{i=1..numberOfDimensions} \mathbf{t}'_i \mathbf{A}' \mathbf{b}_i / ((\mathbf{t}'_i \mathbf{A}' \mathbf{A} \mathbf{t}_i)^{1/2} (\mathbf{b}'_i \mathbf{b}_i)^{1/2}),$$

where \mathbf{t}'_i and \mathbf{b}'_i are the i^{th} column of \mathbf{T} and \mathbf{B} , respectively. A direct solution for \mathbf{T} is not available, it can only be obtained by an iterative procedure. The implemented algorithm is from Kiers & Groenen (1996) and shows excellent convergence properties.

© djmw, April 7, 2004

Configurations: To Procrustus

A command that creates a Procrustus object from two selected Configuration objects.

We calculate the procrustus transform that transforms the second selected Configuration object to match the first selected Configuration object as closely as possible.

© *djmw*, October 8, 2001

congruence coefficient

The *congruence coefficient* is a measure of similarity between two Configurations.

The congruence coefficient $c(\mathbf{X}, \mathbf{Y})$ for the configurations \mathbf{X} and \mathbf{Y} is defined as:

$$c(X, Y) = \sum_{i < j} w_{ij} d_{ij}(\mathbf{X}) d_{ij}(\mathbf{Y}) / ([\sum_{i < j} w_{ij} d_{ij}^2(\mathbf{X})]^{1/2} [\sum_{i < j} w_{ij} d_{ij}^2(\mathbf{Y})]^{1/2}),$$

where $d_{ij}(\mathbf{X})$ is the distance between the points i and j in configuration \mathbf{X} and w_{ij} are nonnegative weights (default: $w_{ij} = 1$).

Since distances are nonnegative, the congruence coefficient has a value between 0 and 1.

The *congruence coefficient* is a better measure of the similarity between configurations than the *correlation coefficient* of the distances. Borg & Groenen (1997) give a simple example where things go wrong with correlation coefficients: two configurations \mathbf{X} and \mathbf{Y} with three points each, have distances $d_{12}(\mathbf{X}) = 1$, $d_{13}(\mathbf{X}) = 2$, $d_{23}(\mathbf{X}) = 3$ and $d_{12}(\mathbf{Y}) = 2$, $d_{13}(\mathbf{Y}) = 3$, $d_{23}(\mathbf{Y}) = 4$. These distances have a correlation coefficient of 1. However, in \mathbf{X} the three points lie on a straight line and in \mathbf{Y} the points form a triangle. This unwanted situation occurs because in the calculation of the correlation coefficient the mean is subtracted from the distances and the resulting values are no longer distances (they may become negative). In calculating the correlation between the distances we should not subtract the mean. In fact, the congruence coefficient is exactly this correlation coefficient calculated with respect to the origin and not with respect to the centroid position (the "mean").

For further information on how well one number can assess the similarity between two configurations see Borg & Groenen (1997) section 19.7.

Links to this page

- Configuration: To Similarity (cc)

© djmw, April 7, 2004

ContingencyTable: To Configuration (ca)...

A command that creates a Configuration object from the selected ContingencyTable object by means of Correspondence analysis.

Settings

Number of dimensions

The dimensionality of the Configuration.

Scaling of the final configuration

determines whether row points are in the centre of gravity of column points, or, column points are in the centre of gravity of row points, or, whether rows and columns are treated symmetrically.

Algorithm

1. We start with the following transformation of the entries f_{ij} :

$$h_{ij} = f_{ij} / \sqrt{(f_{i+}f_{+j})} - \sqrt{(f_{i+}f_{+j})} / N,$$

where h_{ij} is the entry for a cell in the matrix \mathbf{H} with transformed data, f_{i+} is the total count for row i , f_{+j} is the total count for column j and N is the grand total. This can be written in matrix form as:

$$\mathbf{H} = \mathbf{R}^{-1/2} \mathbf{F} \mathbf{C}^{-1/2} - \mathbf{R}^{1/2} \mathbf{u} \mathbf{u}' \mathbf{C}^{1/2} / N,$$

where \mathbf{R} and \mathbf{C} are diagonal matrices with the row and column totals, respectively and \mathbf{u} a column vector with all elements equal to 1.

2. Next the singular value decomposition of matrix \mathbf{H} is performed:

$$\mathbf{H} = \mathbf{K} \mathbf{A} \mathbf{L}',$$

where $\mathbf{K}'\mathbf{K} = \mathbf{I}$, $\mathbf{L}'\mathbf{L} = \mathbf{I}$, and \mathbf{A} is a diagonal matrix with singular values.

3. Now the row (\mathbf{X}) and column points (\mathbf{Y}) can be determined. Three normalizations are possible:

o Scale row points in the centre of gravity of column points

$$\mathbf{X} = \sqrt{N} \mathbf{R}^{-1/2} \mathbf{K} \mathbf{A}$$

$$\mathbf{Y} = \sqrt{N} \mathbf{C}^{-1/2} \mathbf{L}$$

o Scale column points in the centre of gravity of row points

$$\mathbf{X} = \sqrt{N} \mathbf{R}^{-1/2} \mathbf{K}$$

$$\mathbf{Y} = \sqrt{N} \mathbf{C}^{-1/2} \mathbf{L} \mathbf{A}$$

- Treat row points and column points symmetrically

$$\mathbf{X} = \sqrt{N} \mathbf{R}^{-1/2} \mathbf{K} \mathbf{A}^{-1/2}$$

$$\mathbf{Y} = \sqrt{N} \mathbf{C}^{-1/2} \mathbf{L} \mathbf{A}^{-1/2}$$

For more details see Gifi (1990), chapter 8.

© *djmw*, April 7, 2004

Dissimilarity & Configuration & Weight: Get stress...

A command that calculates the stress between distances d_{ij} derived from the selected Configuration object and disparities d'_{ij} derived from the selected Dissimilarity object. With the selected Weight object the evaluation of the influence of each dissimilarity on stress can be influenced.

Settings

Normalized stress, Kruskal's stress-1, Kruskal's stress-2 or Raw stress

Behaviour

Except for *absolute mds*, we use stress formula's that are independent of the scale of the Configuration (see stress): you would have got the same stress value if you had pre-multiplied the selected Configuration with any number greater than zero.

© djmw, April 7, 2004

Dissimilarity & Configuration & Weight: To Configuration...

A command that creates a Configuration object from a Dissimilarity object. With the selected Weight object the influence of each dissimilarity on stress can be influenced. The selected Configuration object serves as a starting configuration for the minimization process.

Settings

- Dissimilarity: To Configuration (monotone mds)...
- Dissimilarity: To Configuration (i-spline mds)...
- Dissimilarity: To Configuration (interval mds)...
- Dissimilarity: To Configuration (ratio mds)...
- Dissimilarity: To Configuration (absolute mds)...

© *djmw*, April 7, 2004

Dissimilarity & Configuration: Draw regression (absolute mds)...

Draws a scatterplot of the dissimilarities δ_{ij} from the selected Dissimilarity object versus disparities d'_{ij} obtained from the "regression" of distances d_{ij} from Configuration on the dissimilarities δ_{ij} .

$$d'_{ij} = \delta_{ij}$$

Settings

Minimum proximity, Maximum proximity

minimum and maximum values for the proximities (horizontal axis).

Minimum distance, Maximum distance

minimum and maximum values for the distances (vertical axis).

Mark size (mm), Mark string

size and kind of the marks in the plot.

garnish

when on, draws a bounding box with decoration.

© djmw, April 7, 2004

Dissimilarity & Configuration: Draw regression (i-spline mds)...

Draws a scatterplot of the dissimilarities δ_{ij} from the selected Dissimilarity versus disparities d'_{ij} obtained from the regression of distances d_{ij} from Configuration on the spline transformed dissimilarities δ_{ij} .

Settings

Number of interior knots

determines the number of segments.

Order of I-spline

The order of the polynomial basis of the I-spline.

Minimum proximity, Maximum proximity

minimum and maximum values for the proximities (horizontal axis).

Minimum distance, Maximum distance

minimum and maximum values for the distances (vertical axis).

Mark size (mm), Mark string

size and kind of the marks in the plot.

garnish

when on, draws a bounding box with decoration.

© djmw, April 7, 2004

Dissimilarity & Configuration: Draw regression (interval mds)...

Draws a scatterplot of the dissimilarities δ_{ij} from the selected Dissimilarity versus disparities d'_{ij} obtained from the regression of distances d_{ij} from Configuration on the dissimilarities δ_{ij} .

$$d'_{ij} = a + b \cdot \delta_{ij},$$

where the values of a and b are determined by regression.

Settings

Minimum proximity, Maximum proximity

minimum and maximum values for the proximities (horizontal axis).

Minimum distance, Maximum distance

minimum and maximum values for the distances (vertical axis).

Mark size (mm), Mark string

size and kind of the marks in the plot.

garnish

when on, draws a bounding box with decoration.

© djmw, April 7, 2004

Dissimilarity & Configuration: Draw regression (ratio mds)...

Draws a scatterplot of the dissimilarities δ_{ij} from the selected Dissimilarity versus disparities d'_{ij} obtained from the "regression" of distances d_{ij} from Configuration on the dissimilarities δ_{ij} .

$$d'_{ij} = b \cdot \delta_{ij},$$

where the value of b is determined by regression.

Settings

Minimum proximity, Maximum proximity

minimum and maximum values for the proximities (horizontal axis).

Minimum distance, Maximum distance

minimum and maximum values for the distances (vertical axis).

Mark size (mm), Mark string

size and kind of the marks in the plot.

garnish

when on, draws a bounding box with decoration.

© djmw, April 7, 2004

Dissimilarity & Configuration: Get stress (absolute mds)...

A command to obtain the stress value for the selected Dissimilarity and Configuration object.

Behaviour

Stress formula's are **dependent** of the scale of the Configuration: you will get **another** stress value if you had pre-multiplied the selected Configuration with any number greater than zero.

© *djmw*, January 19, 1998

Dissimilarity & Configuration: Get stress (i-spline mds)...

A command to obtain the stress value for the selected Dissimilarity and Configuration object.

Behaviour

We use stress formula's that are independent of the scale of the Configuration: you would have got the same stress value if you had pre-multiplied the selected Configuration with any number greater than zero.

© *djmw*, January 19, 1998

Dissimilarity & Configuration: Get stress (interval mds)...

A command to obtain the stress value for the selected Dissimilarity and Configuration object.

Behaviour

We use stress formula's that are independent of the scale of the Configuration: you would have got the same stress value if you had pre-multiplied the selected Configuration with any number greater than zero.

© *djmw*, January 19, 1998

Dissimilarity & Configuration: Get stress (ratio mds)...

A command to obtain the stress value for the selected Dissimilarity and Configuration object.

Behaviour

We use stress formula's that are independent of the scale of the Configuration: you would have got the same stress value if you had pre-multiplied the selected Configuration with any number greater than zero.

© *djmw*, January 19, 1998

Dissimilarity & Configuration: To Configuration (kruskal)...

A command to fit an optimal Configuration for the selected Dissimilarity object. The selected Configuration will be used as the starting configuration in the kruskal analysis.

© *djmw*, December 1, 1997

Dissimilarity: To Configuration (kruskal)...

A command that creates a Configuration object from a Dissimilarity object.

Settings

Number of dimensions (default: 2)

The dimensionality of the Configuration.

Distance metric (default: 2, i.e. Euclidean)

the general distance between points \mathbf{x}_i and \mathbf{x}_j ($i, j = 1..numberOfPoints$) is:

$$\left(\sum_{k=1..numberOfDimensions} |x_{ik} - x_{jk}|^{metric} \right)^{1/metric}$$

Sort distances

determines the handling of ties in the data. When off, whenever two or more dissimilarities are equal we do not care whether the fitted distances are equal or not. Consequently, no constraints are imposed on the fitted distances. When on, however, we impose the constraint that the fitted distances be equal whenever the dissimilarities are equal.

For the calculation of stress:

Formula1 (default)

$$stress = \sqrt{\sum (distance_k - fittedDistance_k)^2 / \sum distance_k^2}$$

Formula2

$$stress = \sqrt{\sum (distance_k - fittedDistance_k)^2 / \sum (distance_k - averageDistance)^2}$$

Note that values of stress 2 are generally more than double those of stress 1 for the same degree of fit.

Finding the optimal Configuration involves a minimization process:

Tolerance

When successive values for the stress differ less than *Tolerance* the minimization process stops.

Maximum number of iterations

Minimization stops after this number of iterations has been reached.

Number of repetitions

When chosen larger than 1, the minimization process will be repeated, each time with another random start configuration. The configuration that results in minimum stress will be saved.

Precautions

When there are few objects it is impossible to recover many dimensions. A rough rule of thumb is that there should be at least twice as many number of observations, i.e. the $numberOfPoints \cdot (numberOfPoints - 1) / 2$ (dis)similarities, than parameters to be estimated, i.e. the $numberOfPoints \cdot numberOfDimensions$ position coordinates. A practical guide is:

- for $numberOfDimensions = 1$ you need ≥ 5 objects
- for $numberOfDimensions = 2$ you need ≥ 9 objects
- for $numberOfDimensions = 3$ you need ≥ 13 objects

There is no feasible way to be certain that you have found the true global minimum. However, by using a great number of different random starting configurations to scale the same data it is often possible to obtain practical certainty. Although the procedure for obtaining an initial configuration is based on a *linear* relation between distance and (dis)similarity, it gives a very good approximation of the optimal **Configuration** and the **Minimizer** practically always finds the global minimum from it (I guess...). A way to find out is to try the *numberOfRepetitions* parameter which gives you the possibility to fit many times and each time start with another random initial configuration.

Algorithm

1. The Dissimilarity object is converted to a Distance object in the same way as in Dissimilarity: To Distance....)
2. From the Distance object an initial Configuration is found by first transforming the Distance object to a matrix with scalar products of distances and subsequently solving for the first $numberOfDimensions$ eigenvectors of this matrix.
3. A minimalization algorithm is started that tries to minimize a function. In this function:
 - o 3.1 We normalize the current Configuration from the minimizer
 - o 3.2 Calculate a new Distance object from the configuration
 - o 3.3 Do a monotone regression of this Distance on the Dissimilarity. This results in a new Distance object.
 - o 3.4 Calculate stress from this Distance and the Distance obtained from Dissimilarity.

The optimization process is controlled by a conjugate gradient minimization algorithm that tries to minimize the *stress* function. In Kruskal (1964), a steepest descent algorithm is used which is less efficient.

Distance & Configuration & Salience: Get VAF...

Calculates the "variance accounted for" from the selected collection of Distance objects, the selected Configuration and the selected **Salience**.

© *djmw*, December 1, 1997

Distance & Configuration & Saliency: To Configuration (indscal)...

A command that creates a new Configuration from the selected collection of Distance objects, the selected Configuration and the selected **Saliency**. The selected Configuration and Saliency serve as start values for the INDSCAL analysis.

© djmw, December 1, 1997

Distance & Configuration: Get VAF...

Calculates the "*variance accounted for*" from the selected collection of Distance objects and the selected Configuration. The optimal Saliency necessary for the calculation will be inferred from the selected Distance and Configuration objects.

© djmw, December 1, 1997

Distance & Configuration: To Configuration (indscale)...

Performs an INDSCALE analysis on the selected objects of type Distance and calculates a Configuration from them. Uses the selected Configuration object as the initial Configuration in the iteration process.

© *djmw*, December 1, 1997

Distance: To Configuration (indscale)...

Perform an INDSCAL analysis on the selected object(s) of type Distance that results in a Configuration and a Saliency object.

Links to this page

- [Create INDSCAL Carroll & Wish example...](#)
-

© *djmw*, November 24, 1997

Distance: To Configuration (ytl)...

A command that creates one Configuration and one Saliency object from a collection of one or more Distance objects.

This Configuration and Saliency object normally serve as starting points for an individual difference scaling such as an INDSCAL analysis.

The algorithm is described in Young, Takane & Lewyckyj (1978).

© *djmw*, November 24, 1997

Confusion: To Dissimilarity...

A command that creates a Dissimilarity from every selected Confusion.

Settings

Normalize

when on, normalize rows by dividing each row element by the row sum. In this way you correct for unequal stimulus numbers.

No symmetrization, Average, Houtgast

determine the symmetrization procedure. See Confusion: To Similarity...

Maximum dissimilarity

determines the maximum dissimilarity possible. When the default value, 0.0, is chosen, *maximumDissimilarity* is calculated as the maximum element in the Similarity object.

Algorithm

We first transform the Confusion to a Similarity. See Confusion: To Similarity...

To obtain dissimilarities from similarities we "reverse" the latter:

$$dissimilarity_{ij} = maximumDissimilarity - similarity_{ij}$$

© djmw, April 7, 2004

Dissimilarity: Get additive constant

A command that calculates the "additive constant" from the selected Dissimilarity.

Distances d_{ij} will be obtained from dissimilarities δ_{ij} according to:

$$distance_{ij} = dissimilarity_{ij} + additiveConstant$$

We use a procedure by Cailliez (1983) to solve the "additive constant problem", i.e. find the smallest *additiveConstant* such that all $distance_{ij}$ in the above equation have a Euclidean representation.

Links to this page

- [Dissimilarity: To Distance...](#)

© *djmw*, December 1, 1997

Confusion: To Dissimilarity (pdf)...

A command that creates a Dissimilarity from every selected Confusion.

Settings

Symmetrize first

when on, the confusion matrix is symmetrized before we calculate dissimilarities.

Maximum dissimilarity (units of sigma)

specifies the dissimilarity from confusion matrix elements that are zero.

Algorithm

1. Normalize rows by dividing each row element by the row sum (optional).
2. Symmetrize the matrix by averaging f_{ij} and f_{ji} .
3. Transformation of the confusion measure which is a sort of *similarity* measure to the *dissimilarity* measure.

Similarity and dissimilarity have an inverse relationship: the greater the similarity, the smaller the dissimilarity and vice versa. Both have a monotonic relationship with distance. The most simple way to transform the similarities f_{ij} into dissimilarities is:

$$dissimilarity_{ij} = maximumSimilarity - similarity_{ij}$$

For ordinal analyses like Kruskal this transformation is fine because only order relations are important in this analysis. However, for metrical analyses like INDSCAL this is not optimal. In INDSCAL, distance is a linear function of dissimilarity. This means that, with the transformation above, you ultimately fit an INDSCAL model in which the distance between object i and j will be linearly related to the confusion between i and j .

For the relation between confusion and dissimilarity, the model implemented here, makes the assumption that the amount of confusion between objects i and j is related to the amount that their probability density functions, pdf's, overlap. Because we do not know these pdf's we make the assumption that both are normal, have equal *sigma* and are one-dimensional. The parameter to be determined is the distance between the centres of both pdf's. According to formula 26.2.23 in Abramowitz & Stegun (1970), for each fraction f_{ij} , we have to find an x that solves:

$$f_{ij} = 1 / \sqrt{(2\pi)} \int_x^{\infty} e^{-t/2} dt$$

This x will be used as the dissimilarity between i and j . The relation between x and f_{ij} is monotonic. This means that the results for a Kruskal analysis will not change much. For INDSCAL, in general, you will note a significantly better fit.

Links to this page

- [Distance](#)
 - [INDSCAL analysis](#)
-

© *djmw*, April 7, 2004

Similarity: To Dissimilarity...

A command that creates a Dissimilarity from every selected Similarity.

Settings

Maximum dissimilarity

determines the maximum dissimilarity possible. When the default value, 0.0, is chosen *maximumDissimilarity* is calculated as the maximum element in the Similarity object.

Algorithm

To obtain dissimilarities we 'reverse' similarities:

$$dissimilarity_{ij} = maximumDissimilarity - similarity_{ij}$$

In this way the order of dissimilarities is the reverse of the order of the similarities.

© djmw, April 7, 2004

Distance: To ScalarProduct...

A command that creates a ScalarProduct for each selected Distance.

Settings

Make sum of squares equal 1.0

when selected, the elements in the resulting matrix part will be scaled such that the sum of all the squared elements in the matrix equals 1.0.

Algorithm

ScalarProduct entries b_{ij} are created from distances d_{ij} by double centering the matrix with elements $-1/2 d_{ij}^2$, i.e.,

$$b_{ij} = -1/2(d_{ij}^2 - d_{.j}^2 - d_{i.}^2 + d_{..}^2),$$

where the dot (\cdot) means averaging over that dimension.

© djmw, April 7, 2004

disparities

The numbers d'_{ij} that result from applying an admissible transformation f on the dissimilarities δ_{ij} , i.e., $d'_{ij} = f(\delta_{ij})$. Disparities have the same dimension as distances. Other names for disparities are *pseudo distances* and *target distances*.

Links to this page

- Dissimilarity & Configuration & Weight: Get stress...
- Dissimilarity & Configuration: Draw regression (absolute mds)...
- Dissimilarity & Configuration: Draw regression (i-spline mds)...
- Dissimilarity & Configuration: Draw regression (interval mds)...
- Dissimilarity & Configuration: Draw regression (monotone mds)...
- Dissimilarity & Configuration: Draw regression (ratio mds)...
- Dissimilarity: To Configuration (absolute mds)...
- Dissimilarity: To Configuration (i-spline mds)...
- Dissimilarity: To Configuration (interval mds)...
- Dissimilarity: To Configuration (monotone mds)...
- Dissimilarity: To Configuration (ratio mds)...
- MDS models
- stress

© djmw, January 11, 1998

spline

A spline function f is a piecewise polynomial function defined on an interval $[x_{min}, x_{max}]$ with specified continuity constraints, i.e., when the interval $[x_{min}, x_{max}]$ is subdivided by points ξ_i such that $x_{min} = \xi_1 < \dots < \xi_q = x_{max}$, then within each subinterval $[\xi_j, \xi_{j+1})$ the function is a polynomial P_j of specified order k .

A knot sequence $t = \{t_1, \dots, t_{n+k}\}$, where n is the number of free parameters that specify the spline function, is derived from the ξ_i by placing knots at the boundary values ξ_i according to the order of continuity at that boundary. The most common continuity characteristics imposed on f request that for adjacent polynomials the derivatives up to order $k-2$ match. For example, the knot sequence of a spline of order k for a partition of $[x_{min}, x_{max}]$ into three intervals ($q = 4$) will be $t_1 = \dots = t_k = x_{min} (= \xi_1)$, $t_{k+1} = \xi_2$, $t_{k+2} = \xi_3$, $t_{k+3} = \dots = t_{k+k+2} = x_{max} (= \xi_4)$. This is called a *simple* knot sequence, because all interior knots are simple. The number of free parameters n for this case obeys a simple formula:

$$n = \text{numberOfInteriorKnots} + \text{order}.$$

With suitable basis functions, for example, the M-spline family $M_i(x|k, t)$, $i=1..n$, we can write any spline f in the form:

$$f = \sum_{i=1..n} a_i M_i,$$

where the M_i are defined by the following recursive formulas:

$$M_i(x|1,t) = 1 / (t_{i+1} - t_i), t_i \leq x < t_{i+1}, 0 \text{ otherwise}$$

$$M_i(x|k,t) = k [(x-t_i)M_i(x|k-1,t) + (t_{i+k}-x)M_{i+1}(x|k-1,t)] / ((k-1)(t_{i+k}-t_i))$$

These M_i are localized because $M_i(x|k,t) > 0$ only when $t_i \leq x < t_{i+k}$ and zero otherwise. Also, we have $\int M_i(x)dx = 1$. Because of this localization a change in coefficient a_i will only effect f within this interval.

The following picture shows an M-spline of order 3 on the interval $[0, 1]$, with three interior knots at 0.3, 0.5 and 0.6.

[sorry, no pictures yet in the web version of this manual]

Because the M-splines are nonnegative, *monotone* splines can be derived from them by *integration*:

$$I_i(x|k,t) = \int_{x_{min}}^x M_i(u|k,t) du$$

Because each $M_i(x|k, t)$ is a piecewise polynomial of degree $k-1$, each I_i will be of degree k . Now we can write:

$$f = \sum_{i=1..n} b_i I_i(x|k,t)$$

We can use an M-spline of order $k+1$ with a simple knot sequence t , for which $t_j \leq x < t_{j+1}$, to put the I-spline of order k into a more convenient form:

$$I_i(x|k,t) = 0, i > j$$

$$I_i(x|k,t) = \sum_{m=i+1..j} (t_{m+k+1} - t_m) M_m(x|k+1,t)/(k+1), j-k \leq i \leq j$$

$$I_i(x|k,t) = 1, i < j-k$$

The following figure shows the I-splines that were derived from the M-splines above.

[sorry, no pictures yet in the web version of this manual]

These spline formula's were taken from Ramsay (1988) and the errors in his I-spline formulas were corrected.

Links to this page

- [Create ISpline...](#)
- [Create MSpline...](#)
- [Dissimilarity & Configuration: Draw regression \(i-spline mds\)...](#)
- [Dissimilarity: To Configuration \(i-spline mds\)...](#)
- [ISpline](#)
- [MDS models](#)
- [MSpline](#)

© djmw, January 12, 1998

Borg & Groenen (1997)

I. Borg & P. Groenen (1997), *Modern Multidimensional Scaling: Theory and Applications*, Springer.

Links to this page

- [congruence coefficient](#)
 - [MDS models](#)
 - [Procrustus](#)
 - [procrustus transform](#)
-

© djmw, December 19, 1997

Ramsay (1988)

J.O. Ramsay (1988), "Monotone Regression Splines in Action", *Statistical Science* **3**, 425-461.

Links to this page

- [MDS models](#)
- [spline](#)

© djmw, January 6, 1998

Green, Carmone & Smith (1989)

P. Green, F. Carmone, S. Smith (1989): *Multidimensional scaling: concepts and applications*, section 3. Allyn and Bacon.

Links to this page

- [Create letter R example...](#)
-

© djmw, December 1, 1997

De Leeuw (1977)

J. de Leeuw (1977), "Applications of convex analysis to multidimensional scaling", In J.R. Barra, F. Brodeau, G. Romier & B. van Cutsem (eds.), *Recent developments in statistics*, Amsterdam, The Netherlands: North-Holland, 133-145.

Links to this page

- [smacof](#)
- [stress](#)

© djmw, December 1, 1997

Kruskal (1964)

J.B. Kruskal (1964), "Nonmetric multidimensional scaling: a numerical method", *Psychometrika* **29**, 115-129.

Links to this page

- Dissimilarity: To Configuration (kruskal)...
- stress

© djmw, December 1, 1997

Takane, Young & de Leeuw (1976)

Y. Takane, F. Young, J. de Leeuw (1976), "Non-metric individual differences multidimensional scaling: an alternating least squares method with optimal scaling features", *Psychometrika* **42**, 7-67.

Links to this page

- [individual difference scaling](#)
-

© djmw, December 1, 1997

Carroll & Chang (1970)

J.D. Carroll & J.-J. Chang, (1970): "Analysis of Individual Differences in Multidimensional scaling via an N-way generalization of "Eckart-Young" Decomposition", *Psychometrika* **35**, 283-319.

Links to this page

- CANDECOMP
- individual difference scaling
- INDSCAL analysis

© djmw, December 1, 1997

CANDECOMP

An algorithm to solve the INDSCAL problem.

In the analysis of the INDSCAL three-way data matrix ($numberOfPoints \times numberOfDimensions \times numberOfSources$) we seek to minimize the function:

$$f(X, W_1, \dots, W_{numberOfSources}) = \sum_{i=1..numberOfSources} |S_i - XW_iX'|^2$$

where S_i is a known symmetric $numberOfPoints \times numberOfPoints$ matrix with scalar products of distances for source i , X is the unknown configuration $numberOfPoints \times numberOfDimensions$ matrix, X' its transpose, and, W_i is the diagonal $numberOfDimensions \times numberOfDimensions$ weight matrix for source i . The function above has no analytical solution for X and the W_i . It can be solved, however, by an iterative procedure which Carroll & Chang have christened CANDECOMP (CANonical DECOMPosition). This method minimizes, instead of the function given above, the following function:

$$g(X, Y, W_1, \dots, W_{numberOfSources}) = \sum_{i=1..numberOfSources} |S_i - XW_iY'|^2$$

where X and Y are both $numberOfPoints \times numberOfDimensions$ configuration matrices.

The algorithm proceeds as follows:

1. Initialize the W matrices and the configuration matrix X . This can for example be done according to a procedure given in Young, Takane & Lewyckyj (1978).
2. An alternating least squares minimization process is started as described that sequentially updates Y , X and W (Carroll & Chang (1970)):
 - 2.1. Solve for a new Y given X and the W_i
 - 2.2. Solve for a new X given the W_i and the new Y .
 - 2.3. Solve for the W_i given the new X and Y .

Evaluate the goodness-of-fit criterion and either repeat the minimization sequence (2.1-2.3) or continue.

3. Done: make Y equal to X and solve a last time for the W_i .

Note: during the minimization the following constraints are effective:

The configuration must be centered.

The sum of squared coordinates in the configuration space is one for each dimension, i.e., the configuration always has unit variance in each dimension.

Links to this page

- [INDSCAL analysis](#)
-

© *djmw*, December 1, 1997

Ten Berge, Kiers & Krijnen (1993)

J.M.F. ten Berge, H.A.L. Kiers & W.P. Krijnen (1993), "Computational Solutions for the Problem of Negative Saliences and Nonsymmetry in INDSCAL", *Journal of Classification* **10**, 115-124.

Links to this page

- [INDSCAL analysis](#)
-

© djmw, December 7, 1997

Golub & van Loan (1996)

G. Golub & C. van Loan (1996), *Matrix computations*, third edition, The Johns Hopkins University Press, London

Links to this page

- [INDSCAL analysis](#)
- [Matrix: Solve equation...](#)

© djmw, December 7, 1997

Carroll & Wish (1974)

J.D. Carroll & M. Wish, (1974): "Models and methods for three-way multidimensional scaling", in D.H. Krantz, R.C. Atkinson, R.D. Luce & P. Suppes (Eds.), *Contemporary developments in mathematical psychology: Vol. 2 Measurement, psychophysics, and neural information processing*, 283-319, New York, Academic Press.

Links to this page

- [Create INDSCAL Carroll & Wish example...](#)
-

© djmw, December 1, 1997

Create formant table (Pols & Van Nierop 1973)

A command to create a Table object filled with the frequencies and the levels of the first three formants from the 12 Dutch monophthong vowels as spoken in /h_t/ context by 50 male and 25 female speakers.

Table layout

The created table will contain 10 columns

Column 1, labeled as *Sex*

speaker sex: Either "m" or "f" (for *male* or *female*).

Column 2, labeled as *Speaker*

speaker id: a number from 1 to 75.

Column 3, labeled as *Vowel*

the vowel name. The following list gives the vowel in p_t context word together with its representation in this column: (*poet*, oe), (*paat*, aa), (*poot*, oo), (*pat*, a), (*peut*, eu), (*piet*, ie), (*puut*, uu), (*peet*, ee), (*put*, u), (*pet*, e), (*pot*, o), (*pit*, i).

Column 4, labeled as *IPA*

the IPA-notation for the vowels

Column 5, 6 and 7, labeled as *F1*, *F2* and *F3*

the frequencies in Hertz of the first three formants.

Column 8, 9 and 10, labeled as *L1*, *L2* and *L3*

the levels in decibel below overall SPL of the first three formants.

More details about these data and how they were measured can be found in Pols et al. (1973) and Van Nierop et al. (1973).

© djmw, June 20, 2002

generalized singular value decomposition

The *generalized singular value decomposition* (gsvd) of an $m \times n$ matrix \mathbf{A} and a $p \times n$ matrix \mathbf{B} is given by the pair of factorizations

$$\mathbf{A} = \mathbf{U} \Sigma_1 [\mathbf{0}, \mathbf{R}] \mathbf{Q}' \quad \text{and} \quad \mathbf{B} = \mathbf{V} \Sigma_2 [\mathbf{0}, \mathbf{R}] \mathbf{Q}'$$

The matrices in these factorizations have the following properties:

- \mathbf{U} [$m \times m$], \mathbf{V} [$p \times p$] and \mathbf{Q} [$n \times n$] are orthogonal matrices. In the reconstruction formula's above we maximally need only the first n columns of matrices \mathbf{U} and \mathbf{V} (when m and/or p are greater than n).
- \mathbf{R} [$r \times r$], is an upper triangular nonsingular matrix. r is the rank of $[\mathbf{A}', \mathbf{B}']'$ and $r \leq n$. The matrix $[\mathbf{0}, \mathbf{R}]$ is $r \times n$ and its first $n \times (n - r)$ part is a zero matrix.
- Σ_1 [$m \times r$] and Σ_2 [$p \times r$] are real, nonnegative and "diagonal".

In practice, the matrices Σ_1 and Σ_2 are never used. Instead a shorter representation with numbers α_i and β_i is used. These numbers obey $0 \leq \alpha_i \leq 1$ and $\alpha_i^2 + \beta_i^2 = 1$. The following relations exist:

$$\Sigma_1' \Sigma_1 + \Sigma_2' \Sigma_2 = \mathbf{I},$$

$$\Sigma_1' \Sigma_1 = \mathbf{diag} (\alpha_1^2, \dots, \alpha_r^2), \text{ and,}$$

$$\Sigma_2' \Sigma_2 = \mathbf{diag} (\beta_1^2, \dots, \beta_r^2).$$

The ratios α_i / β_i are called the *generalized singular values* of the pair \mathbf{A}, \mathbf{B} . Let l be the rank of \mathbf{B} and $k + l (= r)$ the rank of $[\mathbf{A}', \mathbf{B}']'$. Then the first k generalized singular values are infinite and the remaining l are finite. (When \mathbf{B} is of full rank then, of course, $k = 0$).

Special cases

• If \mathbf{B} is a square nonsingular matrix, the gsvd of \mathbf{A} and \mathbf{B} is equivalent to the singular value decomposition of $\mathbf{A} \mathbf{B}^{-1}$.

• The generalized eigenvalues and eigenvectors of $\mathbf{A}' \mathbf{A} - \lambda \mathbf{B}' \mathbf{B}$ can be expressed in terms of the gsvd. The columns of the matrix \mathbf{X} , constructed as

$$X = Q^* \begin{pmatrix} I & 0 \\ 0 & \text{inv}(R) \end{pmatrix},$$

form the eigenvectors. The important eigenvectors, of course, correspond to the positions where the l eigenvalues are not infinite.

Links to this page

- [SSCP: To CCA...](#)
 - [TableOfReal: To Discriminant](#)
-

© *djmw*, October 7, 1998

concentration ellipse

The percentage of bivariate normally distributed data covered by an ellipse whose axes have a length of $numberOfSigmas \cdot \sigma$ can be obtained by integration of the p.d.f. over an elliptical area. This results in the following equation as can be verified from equation 26.3.21 in Abramowitz & Stegun (1970):

$$percentage = (1 - \exp(-numberOfSigmas^2/2)) \cdot 100\%,$$

where the $numberOfSigmas$ is the radius of the "ellipse":

$$(x/\sigma_x)^2 + (y/\sigma_y)^2 = numberOfSigmas^2.$$

The $numberOfSigmas = 1$ ellipse covers 39.3%, the $numberOfSigmas = 2$ ellipse covers 86.5% and the $numberOfSigmas = 3$ ellipse covers 98.9% of the data.

Links to this page

- Discriminant: Draw sigma ellipses...
- Discriminant: Get concentration ellipse area...
- SSCP: Draw sigma ellipse...

© djmw, May 31, 2000

Discriminant & Pattern: To Categories...

A command to use the selected Discriminant to classify each pattern from the selected Pattern into a category.

Arguments as in Discriminant & TableOfReal: To ClassificationTable....

© *djmw*, April 22, 2004

Categories: To Confusion

A command to compute the Confusion matrix from two selected **Categories** objects.

Algorithm

A confusion matrix is constructed from both **Categories** objects in the following way: The first **Categories** object is considered the stimulus **Categories** and its unique (sorted) categories form the row indices of the confusion matrix, the unique (sorted) categories of the second object form the column indices of this matrix.

Next, each element in the first **Categories** object is compared with the corresponding object in the second object and the element in the confusion matrix addressed by this pair is incremented by 1.

Links to this page

- [Feedforward neural networks 2. Quick start](#)
- [Feedforward neural networks 3. FFNet versus discriminant classifier](#)

© *djmw*, September 18, 1996

Confusion: To Similarity...

A command that creates a Similarity from every selected Confusion.

Settings

Normalize

when on, normalize rows by dividing each row element by the row sum. In this way you correct for unequal stimulus numbers.

No symmetrization, Average, Houtgast

determine the symmetrization procedure.

Algorithm

The *Average* procedure averages:

$$similarity_{ij} = similarity_{ji} = (confusion_{ij} + confusion_{ji}) / 2$$

The *Houtgast* procedure as described in the paper by Klein, Plomp & Pols (1970), expresses similarity between stimuli i and j by the number of times that stimulus i and j have resulted in the same response, summated over all response categories.

We use the following formula to calculate the *Houtgast* dissimilarities:

$$similarity_{ij} = \sum_{k=1..numberOfColumns} \min(confusion_{ik}, confusion_{jk})$$

which is equivalent to the formula in the Klein et al. paper:

$$similarity_{ij} = \sum_{k=1..numberOfColumns} (confusion_{ik} + confusion_{jk} - |confusion_{ik} - confusion_{jk}|)$$

Links to this page

- [Confusion: To Dissimilarity...](#)
-

© djmw, April 7, 2004

Confusion: Condense...

Groups row and column labels of the selected Confusion object in order to reduce its dimension.

Arguments

Search

the pattern to match

Replace

the pattern that replaces the match(es)

Replace limit

limits the maximum number of times that a match/replace cycle may occur within each label.

Search and replace are

defines whether the search and replace strings are taken literally or as a regular expression.

Behaviour

First all row and column labels are changed according to the search and replace specification. Next all rows or columns that have the same labels are summed.

© djmw, April 7, 2004

Efron & Tibshirani (1993)

B. Efron & R.J. Tibshirani (1993), *An introduction to the bootstrap*, Chapman & Hall.

Links to this page

- [Bootstrap](#)
-

© *djmw*, November 3, 2003

CCA & TableOfReal: Predict...

- Canonical correlation analysis
-

© *djmw*, May 3, 2002

Weenink (2003)

D.J.M. Weenink (1999), "Canonical correlation analysis", *Proceedings of the Institute of Phonetic Sciences of the University of Amsterdam* **25**, 81-99.

Links to this page

- [Canonical correlation analysis](#)
-

© djmw, February 25, 2004

Discriminant: Get concentration ellipse area...

A command to query the Discriminant object for the area of the concentration ellipse of one of its groups.

Arguments

Number of sigmas

determines the data coverage.

Discriminant plane

When on, the selected *X* and *Y*-dimension will refer to the eigenvectors of the discriminant space, and, consequently, the area of the projection of the hyper ellipsoid onto the space spanned by these eigenvectors will be calculated. When off, the selected *X* and *Y*-dimension will refer to the original dimensions.

Algorithm

See SSCP: Get sigma ellipse area...

© *djmw*, April 7, 2004

Discriminant: Get confidence ellipse area...

A command to query the Discriminant object for the area of the confidence ellipse of one of its groups.

Arguments

Discriminant plane

When on, the selected *X* and *Y*-dimension will refer to the eigenvectors of the discriminant space, and, consequently, the area of the projection of the hyper ellipsoid onto the space spanned by these eigenvectors will be calculated. When off, the selected *X* and *Y*-dimension will refer to the original dimensions.

Algorithm

See SSCP: Get confidence ellipse area...

© *djmw*, April 7, 2004

Discriminant: Get contribution of component...

A command to ask the selected Discriminant for the contribution of the j^{th} discriminant function (component) to the total variance.

Details

The contribution is defined as:

$$\text{eigenvalue}[j] / \sum_{i=1..numberOfEigenvalues} \text{eigenvalue}[i]$$

© djmw, November 6, 1998

Discriminant: Get partial discrimination probability...

A command to test the selected Discriminant for the significance of discrimination afforded by the remaining $n-k$ eigenvectors after the acceptance of the first k eigenvectors.

Details

The test statistic is:

$\chi^2 = -(degreesOfFreedom - (numberOfGroups + dimension) / 2) \ln \Lambda'$, where

$degreesOfFreedom = (dimension - k)(numberOfGroups - k - 1)$, and,

$\Lambda' = \prod_{j=k+1..numberOfEigenvalues} 1 / (1 + eigenvalue[j])$

© djmw, November 2, 1998

Discriminant: Get Wilks' lambda...

A command to ask the selected Discriminant for the value of Wilks' lambda (a multivariate measure of group differences over several variables).

Arguments

From

the first eigenvalue number from which the value for lambda has to be calculated.

Details

Wilks' lambda is defined as:

$$\Lambda = \prod_{i=\text{from..numberOfEigenvalues}} 1 / (1 + \text{eigenvalue}[i])$$

Because lambda is a kind of *inverse* measure, values of lambda which are near zero denote high discrimination between groups.

© djmw, April 7, 2004

epoch

A term that is often used in the context of machine learning. An epoch is one complete presentation of the *data set to be learned* to a learning machine.

Learning machines like feedforward neural nets that use iterative algorithms often need many epochs during their learning phase.

A discriminant classifier is also a learning machine. However, in contrast with neural nets a discriminant classifier only needs one epoch to learn.

Links to this page

- [Feedforward neural networks 1.1. The learning phase](#)
-

© *djmw*, April 28, 2004

Feedforward neural networks

This tutorial describes the use of FFNet feedforward neural networks in PRAAT.

1. What is a feedforward neural network?

1.1 The learning phase

1.2 The classification phase

2. Quick start

3. FFNet versus discriminant classifier

4. Command overview

© *djmw*, May 11, 2004

Encapsulated PostScript

a kind of PostScript file that can easily be imported into word processors and drawing programs. In Praat, you can create an Encapsulated PostScript (EPS) file by choosing Write to EPS file....

Links to this page

- [PostScript settings...](#)
- [Read from Praat picture file...](#)
- [Write to Praat picture file...](#)

© *ppgb*, October 10, 2000

Clear history

A command in the Edit menu of the ScriptEditor for clearing the remembered history. See History mechanism.

Links to this page

- [Scripting 1. My first script](#)
 - [Scripting 2. Arguments to commands](#)
-

© *ppgb*, September 27, 2000

Paste history

A command in the Control menu for viewing the history in a ScriptEditor. See History mechanism.

Links to this page

- [Scripting 1. My first script](#)
 - [Scripting 2. Arguments to commands](#)
-

© *ppgb*, November 7, 1998

Buttons file

The file into which changes in the availability and visibility of commands in the fixed and dynamic menus are recorded.

The buttons file is written to disk when you leave Praat, and it is read again when you enter Praat the next time. It is a simple Praat script that you can read (but should not edit) with any text editor.

Adding buttons

To add a command to a fixed or dynamic menu, you typically use the ScriptEditor.

Removing buttons

To remove an added command from a fixed or dynamic menu, you typically use the ButtonEditor.

Hiding and showing buttons

To hide a built-in command from a fixed or dynamic menu, or to make a hidden command visible, you typically use the ButtonEditor.

Unix

If your home directory is /people/miep, the buttons file is /people/miep/.praat-dir/buttons. If the directory .praat-dir does not exist, it is created when you enter Praat. If you rename Praat, the name of the directory will also be different.

MacOS X

If you are Miep, the buttons file will be /Users/Miep/Library/Preferences/Praat Prefs/Buttons.

Classic Macintosh

The buttons file, which is called "Buttons", will be in a folder named "Praat Preferences" in the Preferences folder in your System Folder. If your hard disk is called "Hæl' sji'f", and you have a Dutch system, the complete path to your preferences file is:

```
Hæl' sji'f:Stysteemmap:Voorkeuren:Praat Preferences:Buttons
```

Windows

The buttons file may be C:\WINDOWS\Praat\Buttons.ini.

Links to this page

- [Add action command...](#)
 - [Add menu command...](#)
 - [Hidden commands](#)
 - [Initialization script](#)
-

© *ppgb*, December 4, 2002

FAQ: Scripts

Question: how do I do something to all the files in a directory?

Answer: look at Create Strings as file list....

Question: why doesn't the editor window react to my commands?

Your commands are probably something like:

```
Read from file... hello.wav
Edit
Zoom... 0.3 0.5
```

Answer: Praat doesn't know it has to send the **Zoom** command to the editor window called **Sound hello**. There could be several Sound editor windows on your screen. According to Scripting 7.1. Scripting an editor from a shell script, you will have to say this explicitly:

```
Read from file... hello.wav
Edit
editor Sound hello
Zoom... 0.3 0.5
```

Problem: a line like "Number = 1" does not work.

Solution: names of variables should start with a lower-case letter.

Question: why do names of variables have to start with a lower-case letter? I would like to do things like "F0 = Get mean pitch".

Answer: Praat scripts combine button commands with things that only occur in scripts. Button commands always start with a capital letter, e.g. "Play". Script command always start with lower case, e.g. "echo Hello". A minimal pair is "select", which simulates a mouse click in the object list, versus "Select...", which sets the selection in editor windows. Variable names that start with a capital letter would be rather ambiguous in assignments, as in "x = Get", where "Get" would be a variable, versus "x = Get mean", where "Get mean" is a button command. To prevent this, Praat enforces a rigorous lower-case/upper-case distinction.

Question: how do I convert a number into a string?

Answer: a\$ = "'a'"

Question: how do I convert a string into a number?

Answer: a = 'a\$'

Links to this page

- [FAQ \(Frequently Asked Questions\)](#)
-

© *ppgb*, February 22, 2004

Script for listing time\F0 pairs

I wish to have a list of time markers in one column and F0 in the other. I want to have Those times that have no voiced data should be represented as "." in the F0 column.

```
echo Time: Pitch:
numberOfFrames = Get number of frames
for iframe to numberOfFrames
    time = Get time from frame... iframe
    pitch = Get value in frame... iframe Hertz
    if pitch = undefined
        printline 'time:6' .
    else
        printline 'time:6' 'pitch:3'
    endif
endfor
```

If you want to see this in a text file, you can copy and paste from the Info window, or add a line like

```
fappendinfo out.txt
```

Links to this page

- [Scripting examples](#)

© *ppgb*, October 23, 2003

Script for creating a frequency sweep

"I have to find a formula for a sinewave which sweeps from 1 kHz to 12 kHz in 60 seconds while ramping the amplitude from 1 to 12 volts in the same amount of time."

The absolute amplitude in volts cannot be handled, of course, but linear crescendo is easy:

```
Create Sound... sweep 0 60 44100
... 0.05 * (1 + 11 * x/60) * sin (2*pi * (1000 + 11000/2 * x/60) * x)
```

Note the "/2" in this formula. Here is the derivation of the formula:

$$\text{frequency}(t) = 1000 + 11000 t / 60$$

$$\text{phase}(t) = \int \text{frequency}(t) dt = 1000 t + 11000 (t^2/2) / 60$$

$$\text{signal}(t) = \sin(\text{phase}(t))$$

Links to this page

- [Scripting examples](#)

© ppgb, February 22, 2004

Script for onset detection

"Can anybody provide me with a script which that detects the onset of sound (i.e. the end of silence)."

You can create an Intensity contour and look for the first frame that is above some predefined threshold:

```
To Intensity... 100 0
n = Get number of frames
for i to n
  intensity = Get value in frame... i
  if intensity > 40
    time = Get time from frame... i
    echo Onset of sound at: 'time:3' seconds.
    exit
  endif
endfor
```

Since the intensity is computed with rather long windows, the result may be 0.01 or 0.02 seconds before the actual start of sound.

Links to this page

- [Scripting examples](#)

© *ppgb*, February 22, 2004

Script for TextGrid boundary drawing

"I want only the dotted lines of the textgrid marked on top of another analysis (e.g. pitch, intensity or so) without the labels being shown below it."

```
n = Get number of intervals... 1
for i to n-1
  t = Get end point... 1 i
  One mark bottom... t no no yes
endfor
```

Links to this page

- [Scripting examples](#)
-

© *ppgb*, February 22, 2004

Script for analysing pitch with a TextGrid

"I want the mean pitch of every interval that has a non-empty label on tier 5."

```
if numberOfSelected ("Sound") <> 1 or numberOfSelected ("TextGrid") <>
1
    exit Please select a Sound and a TextGrid first.
endif
sound = selected ("Sound")
textgrid = selected ("TextGrid")
echo Result:
select sound
To Pitch... 0.0 75 600
pitch = selected ("Pitch")
select textgrid
n = Get number of intervals... 5
for i to n
    tekst = Get label of interval... 5 i
    if tekst <> ""
        t1 = Get starting point... 5 i
        t2 = Get end point... 5 i
        select pitch
        f0 = Get mean... t1 t2 Hertz
        printline 't1:3' 't2:3' 'f0:0' 'tekst$'
        select textgrid
    endif
endfor
select sound
plus textgrid
```

Links to this page

- [Scripting examples](#)

© ppgb, February 22, 2004

FFNet & Pattern: To Categories...

The FFNet is used as a classifier. Each pattern from the Pattern will be classified into one of the FFNet's categories.

Links to this page

- [Categories](#)
- [Feedforward neural networks 1.1. The learning phase](#)
- [Feedforward neural networks 1.2. The classification phase](#)
- [Feedforward neural networks 2. Quick start](#)
- [Feedforward neural networks 4. Command overview](#)

© *djmw*, September 18, 1996

FFNet & Pattern & Categories: Learn...

You can choose this command after selecting one Pattern, one Categories and one FFNet.

Arguments

Maximum number of epochs

the maximum number of times that the complete **Pattern** dataset will be presented to the neural net.

Tolerance of minimizer

when the difference in costs between two successive learning cycles is smaller than this value, the minimization process will be stopped.

Cost function

minimum squared error:

$$cost = \sum_{allPatterns} \sum_{allOutputs} (o_k - d_k)^2, \text{ where}$$

o_k : actual output of unit k

d_k : desired output of unit k

minimum cross entropy:

$$cost = - \sum_{allPatterns} \sum_{allOutputs} (d_k \cdot \ln o_k + (1-d_k) \cdot \ln (1-o_k))$$

Algorithm

The minimization procedure is a variant of conjugate gradient minimization, see for example Press et al. (1992), chapter 10, or Nocedal & Wright (1999), chapter 5.

Links to this page

- [Feedforward neural networks 1.1. The learning phase](#)
- [Feedforward neural networks 2. Quick start](#)
- [Feedforward neural networks 4. Command overview](#)

© djmw, May 11, 2004

Feedforward neural networks 1. What is a feedforward neural network?

A feedforward neural network is a biologically inspired classification algorithm. It consists of a (possibly large) number of simple neuron-like processing *units*, organized in *layers*. Every unit in a layer is connected with all the units in the previous layer. These connections are not all equal, each connection may have a different strength or *weight*. The weights on these connections encode the knowledge of a network. Often the units in a neural network are also called *nodes*.

Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. During normal operation, that is when it acts as a classifier, there is no feedback between layers. This is why they are called *feedforward* neural networks.

In the following figure we see an example of a 2-layered network with, from top to bottom: an output layer with 5 units, a *hidden* layer with 4 units, respectively. The network has 3 input units.

[sorry, no pictures yet in the web version of this manual]

The 3 inputs are shown as circles and these do not belong to any layer of the network (although the inputs sometimes are considered as a virtual layer with layer number 0). Any layer that is not an output layer is a *hidden* layer. This network therefore has 1 hidden layer and 1 output layer. The figure also shows all the connections between the units in different layers. A layer only connects to the previous layer.

The operation of this network can be divided into two phases:

1. The learning phase
2. The classification phase

Links to this page

- [Feedforward neural networks](#)

© djmw, April 26, 2004

Feedforward neural networks 1.2. The classification phase

In the classification phase the weights of the network are fixed.

A pattern, presented at the inputs, will be transformed from layer to layer until it reaches the output layer. Now classification can occur by selecting the category associated with the output unit that has the largest output value. For classification we only need to select an FFNet and a Pattern together and choose To Categories....

In contrast to the learning phase classification is very fast.

Links to this page

- [Feedforward neural networks](#)
- [Feedforward neural networks 1. What is a feedforward neural network?](#)

© *djmw*, April 28, 2004

Initialization script

Your initialization script is a normal Praat script that is run as soon as you start Praat.

You create an initialization script by creating a file named `"/usr/local/praat-startUp"`, or putting a file `".praat-user-startUp"` or `"praat-user-startUp"` in your home directory, if your program is called Praat. If your program is called Ldb, it is `"/usr/local/ldb-startUp"` etc.

If you have more than one of these files, they are run in the above order.

Example

If you like to be greeted by your husband when Praat starts up, you could put the following lines in your initialization script:

```
# This works if you have the Praat phonetics library:
Read from file... /u/miep/halloMiep.aifc
Play
Remove
# Otherwise, this would work on SGI:
unix sfplay /u/miep/halloMiep.aifc
```

What not to use an initialization script for

You could set preferences like the default font in your initialization script, but these will be automatically remembered between invocations of Praat anyway (in your Preferences file), so this would often be superfluous. Added and removed commands are also remembered across Praat sessions (in your Buttons file), but you may want to call a changeable list of them.

Using an initialization script for site-wide customization

If your research group shares a number of Praat scripts, these can be included in everybody's version of the program in the following way:

1. Create a script that adds buttons to the fixed and dynamic menus, using the commands `Add menu command...` and `Add action command....` This script could be a slightly edited copy of someone's Buttons file.
2. Put this script in `"/usr/local/myProg-startUp"`, or have everyone call this script from her start-up file (with `Run script...`).

This procedure allows all members of the group to automatically enjoy all the later changes in the custom command set.

Links to this page

- [Add to dynamic menu...](#)
- [Add to fixed menu...](#)
- [ButtonEditor](#)
- [Hidden commands](#)

© *ppgb*, October 6, 1997

Formulas 8. Data in objects

With square brackets, you can get the values inside some objects.

Object contents in the calculator

The outcomes of the following examples can be checked with the calculator.

Matrix_hello [10, 3]

gives the value in the cell at the third column of the 10th row of the Matrix called *hello*.

Sound_hello [10000]

gives the value (in Pa) of the 1000th sample of the Sound *hello*.

TableOfReal_tokens [5, 12]

gives the value in the cell at the fifth row of the 12th column of the TableOfReal called *tokens*.

TableOfReal_tokens [5, "F1"]

gives the value in the cell at the fifth row of the column labelled *F1* of the TableOfReal *tokens*.

TableOfReal_tokens ["\ct", "F1"]

gives the value in the cell at the row labelled *\ct* of column *F1* of the TableOfReal *tokens*.

Table_listeners [3, "m3ae"]

gives the numeric value in the cell at the third row of column *m3ae* of the Table *listeners*.

Table_listeners [3, 12]

gives the numeric value in the cell at the third row of the 12th column of the Table *listeners*.

Cells outside the objects are considered to contain zeroes.

Interpolation

The values inside some objects can be interpolated.

Sound_hallo (0.7)

gives the value (in Pa) at a time of 0.7 seconds in the Sound "hallo", by linear interpolation between the two samples that are nearest to 0.7 seconds.

Spectrogram_hallo (0.7, 2500)

gives the value at a time of 0.7 seconds and at a frequency of 2500 Hz in the Spectrogram "hallo", by linear interpolation between the four samples that are nearest to that point.

In the interpolation, cells outside the objects are considered to contain zeroes.

Object contents in a modification formula

Suppose you want to do the difficult way of reversing the contents of a Sound called *hello* (the easy way is to choose *Reverse* from the Modify submenu). You select this sound, then choose Copy... to duplicate it to a new Sound, which you name *hello_reverse*. You select this new Sound and choose Formula... from the Modify submenu. The formula will be

```
Sound_hello [ncol + 1 - col]
```

From this example, you see that the indices between [] may be formulas themselves, and that you can use implicit attributes like *ncol* and position references like *col*. An alternative formula is

```
Sound_hello (xmax - x)
```

at least if *xmin* is zero. The advantage of the second method is that it also works correctly if the two sounds have different sampling frequencies; the disadvantage is that it may do some interpolation between the samples, which deteriorates the sound quality.

Object contents in a script

In scripts, the indices between [] and the values between () may be formulas themselves and contain variables. The following script computes the sum of all the cells along the diagonal of a Matrix named *hello*.

```
sumDiagonal = 0
for i to Matrix_hello.ncol
    sumDiagonal += Matrix_hello [i, i]
endfor
echo The sum of cells along the diagonal is 'sumDiagonal'.
```

This example could have been written completely with commands from the dynamic menu:

```
select Matrix hello
sumDiagonal = 0
ncol = Get number of columns
for i to ncol
    value = Get value in cell... i i
    sumDiagonal += value
endfor
echo The sum of cells along the diagonal is 'sumDiagonal'.
```

The first version, which accesses the contents directly, is not only three lines shorter, but also three times faster.

Links to this page

- [Formulas](#)
-

© ppgb, December 4, 2002

Cochleagram: Formula...

A command for changing the data in all selected Cochleagram objects.

See the Formulas tutorial for examples and explanations.

© *ppgb*, December 6, 2002

Excitation: Formula...

A command for changing the data in all selected Excitation objects.

See the Formulas tutorial for examples and explanations.

© *ppgb*, December 6, 2002

Excitation: Get loudness

A query to ask the selected Excitation object for its loudness.

Return value

the loudness in some units.

Algorithm

The loudness is defined as

$$\int df 2^{(e(f) - 40 \text{ phon}) / 10}$$

where f is the frequency in Bark, and $e(f)$ the excitation in phon. For our discrete Excitation object, the loudness is computed as

$$\Delta f \sum 2^{(e_i - 40) / 10}$$

where Δf is the distance between the excitation channels (in Bark).

Links to this page

- [Sound: Get intensity \(dB\)](#)

© ppgb, October 16, 1999

Excitations: To Pattern...

A command to convert every selected Excitations to a Pattern object.

Arguments

Join

the number of subsequent Excitation objects to combine into one row of Pattern. E.g. if an **Excitation** has length 26 and *join* = 2 then each row of **Pattern** contains 52 elements. The number of rows in **Pattern** will be *my size* / 2. In the conversion process the elements of an **Excitation** will be divided by 100.0 in order to guarantee that all patterns have values between 0 and 1.

© djmw, September 18, 1996

Harmonicity: Formula...

A command for changing the data in all selected Harmonicity objects.

See the Formulas tutorial for examples and explanations.

© *ppgb*, December 6, 2002

Harmonicity: Get maximum...

A query to the selected Harmonicity object.

Return value

the maximum value, expressed in dB.

Arguments

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Harmonicity object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Parabolic because of the usual nonlinearity (logarithm) in the computation of harmonicity; sinc interpolation would be too stiff and may give unexpected results.

© ppgb, September 16, 2003

Harmonicity: Get mean...

A query to the selected Harmonicity object.

Return value

the mean value, expressed in dB.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Harmonicity is considered.

Algorithm

The mean harmonicity between the times t_1 and t_2 is defined as

$$1/(t_2 - t_1) \int_{t_1}^{t_2} dt x(t)$$

where $x(t)$ is the harmonicity as a function of time. For our discrete Harmonicity object, this mean is approximated by

$$1/n \sum_{i=m..m+n-1} x_i$$

where n is the number of frame centres between t_1 and t_2 . Frames in which the value is undefined (i.e. in silent intervals) are ignored. If all the frames are silent, the returned value is undefined.

© ppgb, October 16, 1999

Harmonicity: Get minimum...

A query to the selected Harmonicity object.

Return value

the minimum value, expressed in dB.

Arguments

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Harmonicity object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Parabolic because of the usual nonlinearity (logarithm) in the computation of harmonicity; sinc interpolation would be too stiff and may give unexpected results.

© ppgb, September 16, 2003

Harmonicity: Get standard deviation...

A query to the selected Harmonicity object.

Return value

the standard deviation, expressed in dB.

Arguments

From time (s), *To time* (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Harmonicity is considered.

Algorithm

The standard deviation between the times t_1 and t_2 is defined as

$$1/(t_2 - t_1) \int_{t_1}^{t_2} dt (x(t) - \mu)^2$$

where $x(t)$ is the harmonicity as a function of time, and μ its mean. For our discrete Harmonicity object, the standard deviation is approximated by

$$1/(n-1) \sum_{i=m..m+n-1} (x_i - \mu)^2$$

where n is the number of frame centres between t_1 and t_2 . Note the "minus 1".

© ppgb, October 16, 1999

Harmonicity: Get time of maximum...

A query to the selected Harmonicity object for the time associated with its maximum value.

Return value

the time (in seconds) associated with the maximum HNR value.

Arguments

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Harmonicity object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Parabolic because of the usual nonlinearity (logarithm) in the computation of harmonicity; sinc interpolation would be too stiff and may give unexpected results.

© ppgb, September 16, 2003

Harmonicity: Get time of minimum...

A query to the selected Harmonicity object.

Return value

the time (in seconds) associated with the minimum HNR value.

Arguments

From time (s), To time (s)

the selected time domain. Values outside this domain are ignored. If *To time* is not greater than *From time*, the entire time domain of the Harmonicity object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is Parabolic because of the usual nonlinearity (logarithm) in the computation of harmonicity; sinc interpolation would be too stiff and may give unexpected results.

© ppgb, September 16, 2003

Harmonicity: Get value at time...

A query to the selected Harmonicity object to .

Arguments

Time (s)

the time at which the value is to be evaluated.

Interpolation

the interpolation method, see vector value interpolation. The standard is Cubic because of the usual nonlinearity (logarithm) in the computation of harmonicity; sinc interpolation would be too stiff and may give unexpected results.

Return value

an estimate (in dB) of the value at a specified time. If *time* is outside the samples of the Harmonicity, the result is 0.

© ppgb, September 16, 2003

Harmonicity: Get value in frame...

A query to ask the selected Harmonicity object.

Argument

Frame number

the frame whose value is to be looked up.

Return value

the value in a specified frame, expressed in dB. If the index is less than 1 or greater than the number of frames, the result is 0; otherwise, it is $z[1][frame\ number]$.

© ppgb, October 16, 1999

Goodies

The title of a submenu of the Control menu.

Links to this page

- [Calculator...](#)
- [Formulas](#)
- [Formulas 1.1. Formulas in the calculator](#)

© *ppgb*, December 4, 2002

Formulas 1.1. Formulas in the calculator

To use the Praat calculator, go to the Control menu (in MacOS X, this is the Praat menu), and choose Calculator... from the Goodies submenu. Or simply type Command-U anywhere in Praat.

Calculating numbers

You can do arithmetic computations. Type the formula

```
8*17
```

and click OK. The Info window will pop up and show the result:

```
136
```

Calculating strings

You can also do text computations. Type the formula

```
"see" + "king"
```

and click OK. The Info window will show the result:

```
seeking
```

Links to this page

- [Formulas](#)
- [Formulas 1. My first formulas](#)

© ppgb, December 4, 2002

Formulas 1.2. Numeric expressions

All the formulas whose outcome is a number are called numeric expressions. For the following examples, all the outcomes can be checked with the calculator.

Examples with numbers

Some numeric expressions involve numbers only:

8*17

computes a multiplication. Outcome: 136.

2^10

computes the tenth power of 2. Outcome: 1024.

sqrt (2) / 2

computes the square root of 2, and divides the result by 2. Outcome: 0.7071067811865476.

sin (1/4 * pi)

computes the sine of $\pi/4$. Outcome: 0.7071067811865476 (again).

Examples with strings

Some numeric expressions compute numeric properties of strings:

length ("internationalization")

computes the length of the string "internationalization". Outcome: 20.

index ("internationalization", "ation")

computes the location of the first occurrence of the string "ation" in the string "internationalization". Outcome: 7, because the first letter of "ation" lines up with the seventh letter of "internationalization".

rindex ("internationalization", "ation")

computes the location of the last occurrence of the string "ation" in the string "internationalization". Outcome: 16.

Links to this page

- [Formulas](#)
- [Formulas 1. My first formulas](#)

© ppgb, December 1, 2002

Formulas 1.3. String expressions

All the formulas whose outcome is a text string are called string expressions. Again, the outcomes of the following examples can be checked with the calculator.

"see" + "king"

concatenates two strings. Outcome: seeking.

left\$ ("internationalization", 6)

computes the leftmost six letters of the string; the dollar sign is used for all functions whose result is a string. Outcome: intern.

mid\$ ("internationalization", 6, 8)

computes the 8-letter substring that starts at the sixth letter of "internationalization". Outcome: national.

date\$ ()

computes the current date and time. Outcome at the time I am writing this: Mon Dec 2 02:23:45 2002.

Links to this page

- [Formulas](#)
 - [Formulas 1. My first formulas](#)
-

© *ppgb*, April 14, 2004

Formulas 1.4. Representation of numbers

Formulas can work with integer numbers as well as with real numbers.

Real numbers

You can type many real numbers by using a decimal notation, for instance 3.14159, 299792.5, or -0.000123456789. For very large or small numbers, you can use the *e*-notation: $6.022 \cdot 10^{23}$ is typed as 6.022e23 or 6.022e+23, and $-1.6021917 \cdot 10^{-19}$ is typed as -1.6021917e-19. You can also use the percent notation: 0.157 can be typed as 15.7%.

There are some limitations as to the values that real numbers can have in Praat. The numbers must lie between -10^{308} and $+10^{308}$. If you type

```
1e200 * 1e100
```

the outcome will be

```
1e+300
```

but if you type

```
1e300 * 1e100
```

the outcome will be

```
--undefined--
```

Another limitation is that the smallest non-zero numbers lie near -10^{-308} and $+10^{-308}$. If you type

```
1e-200 / 1e100
```

the outcome will be

```
1e-300
```

but if you type

```
1e-300 / 1e100
```

the outcome will be

```
0
```

Finally, the precision of real numbers is limited by the number of bits that every real number is stored with in the computer, namely 64. For instance, if you type

pi

the outcome will be

3.141592653589793

because only 16 digits of precision are stored. This can lead to unexpected results caused by rounding. For instance, the formula

0.34999999999999999 - 0.35

will result in

0

rather than the correct value of $1e-17$. This is because the numbers 0.34999999999999999 and 0.35 cannot be distinguished in the computer's memory. If you simply type

0.34999999999999999

the outcome will be

0.35

(as in this example, the calculator will always come up with the minimum number of digits needed to represent the number unambiguously).

Another example of inaccuracy is the formula

1 / 7 / 59 * 413

Because of rounding errors, the result will be

0.9999999999999999

Integer numbers

Formulas can work with integer (whole) numbers between -1,000,000,000,000,000 and +1,000,000,000,000,000. You type them without commas and without the plus sign: 337, -848947328345289.

You *can* work with larger numbers than that (up to 10^{308}), but there will again be rounding errors. For instance, the formula

10000000000000000 + 1

correctly yields

10000000000000001

but the formula

10000000000000000 + 1

yields an incorrect outcome:

1e16

Links to this page

- [Formulas](#)
- [Formulas 1. My first formulas](#)

© *ppgb*, April 14, 2004

Formulas 1.5. Representation of strings

Formulas can work with strings that are put between two double quotes, as in "goodbye" or "how are you doing?".

If a string has to contain a double quote, you have to type it twice. For instance, if you type

```
"I asked: ""how are you doing?"""
```

into the calculator, the outcome will be

```
I asked: "how are you doing?"
```

Links to this page

- [Formulas](#)
- [Formulas 1. My first formulas](#)

© *ppgb*, December 3, 2002

Formulas 2. Operators

In formulas you can use the numerical and logical operators that are described on this page. The order of evaluation of the operators is the order that is most usual in programming languages. To force a different order, you use parentheses.

The operators with the highest precedence are **negation** (-) and **exponentiation** (^):

--6 -> 6
2^6 -> 64

Sequences of negation and exponentiation are evaluated from right to left:

2^-6 -> 0.015625
-(1+1)^6 -> -64
4^3^2 -> 4^9 -> 262144

Note that changing the spacing does not change the meaning:

4^3 ^ 2 -> 262144

To change the order of evaluation, you have to use parentheses:

(4 ^ 3) ^ 2 -> 4096

The following construction is not allowed because of an ambiguity between a negative number and negation of a positive number:

-2^6 -> ?

Instead, you use any of the following:

(-2)^6 -> 64
-(2^6) -> -64
-(2)^6 -> -64

The operators with the next highest precedence are **multiplication** (*) and **division** (/). They are evaluated from left to right:

1/4*5 -> 1.25 (from left to right)
1 / 4*5 -> 1.25 (spacing does not help)
1 / (4*5) -> 0.05 (use parentheses to change the order)
3 * 2 ^ 4 -> 48 (exponentiation before multiplication)
3*2 ^ 4 -> 48 (this spacing does not matter and is misleading)
(3 * 2) ^ 4 -> 1296 (use parentheses to change the order)

Integer division operators (**div** and **mod**) have the same precedence as ***** and **/**, and are likewise evaluated from left to right:

```
54 div 5 -> 10 (division rounded down)
54 mod 5 -> 4 (the remainder)
54.3 div 5.1 -> 10 (works for real numbers as well)
54.3 mod 5.1 -> 3.3 (the remainder)
-54 div 5 -> -11 (division rounded down; negation before division)
-54 mod 5 -> 1 (the remainder)
-(54 div 5) -> -10 (use parentheses to change the order)
-(54 mod 5) -> -4
3 * 18 div 5 -> 10 (from left to right)
3 * (18 div 5) -> 9
3 * 18 mod 5 -> 4
3 * (18 mod 5) -> 9
54 div 5 * 3 -> 30 (from left to right)
54 div (5 * 3) -> 3
54 mod 5 * 3 -> 12
54 mod (5 * 3) -> 9
```

The operators with the next highest precedence are **addition** (+) and **subtraction** (-), evaluated from left to right:

```
3 - 8 + 7 -> 2 (from left to right)
3 - (8 + 7) -> -12 (use parentheses to change the order)
3 + 8 * 7 -> 59 (multiplication before addition)
(3 + 8) * 7 -> 77 (use parentheses to change the order)
3 + - 2 ^ 4 -> -13 (exponentiation, negation, addition)
3 + 5 / 2 + 3 -> 8.5
(3 + 5) / (2 + 3) -> 1.6
```

The operators with the next highest precedence are the **comparison** operators (= <> < > <= >=). These operators always yield 0 (*false*) or 1 (*true*):

```
5 + 6 = 10 -> 0 (equal)
5 + 6 = 11 -> 1
5 + 6 <> 10 -> 1 (unequal)
5 + 6 <> 11 -> 0
5 + 6 < 10 -> 0 (less than)
5 + 6 < 11 -> 0
5 + 6 > 10 -> 1 (greater than)
5 + 6 > 11 -> 0
5 + 6 <= 10 -> 0 (less than or equal)
5 + 6 <= 11 -> 1
5 + 6 >= 10 -> 1 (greater or equal)
5 + 6 >= 11 -> 1
```

The comparison operators are mainly used in **if**, **while**, and **until** conditions.

The operators of lowest precedence are the **logical** operators (**not**, **and**, and **or**), of which **not** has the highest precedence and **or** the lowest:

```
not 5 + 6 = 10 -> 1
x > 5 and x < 10 (is x between 5 and 10?)
not x <= 5 and not x >= 10 (same as previous line)
not (x <= 5 or x >= 10) (same as previous line)
```

String comparison

a\$ = b\$

gives the value *true* (= 1) if the strings are equal, and *false* (= 0) otherwise.

a\$ <> b\$

gives the value *true* if the strings are unequal, and *false* otherwise.

a\$ < b\$

gives *true* if the string *a\$* precedes the string *b\$* in ASCII sorting order. Thus, "ha" < "hal" and "ha" < "ja" are true, but "ha" < "JA" is false, because all capitals precede all lower-case characters in the ASCII sorting order.

a\$ > b\$

true if *a\$* comes after *b\$* in ASCII sorting order.

a\$ <= b\$

gives the value *true* if the string *a\$* precedes the string *b\$* in ASCII sorting order, or if the strings are equal.

a\$ >= b\$

true if *a\$* comes after *b\$* or the two are equal.

String concatenation and truncation

a\$ + b\$

concatenates the two strings. After

```
text$ = "hallo" + "dag"
```

The variable *text\$* contains the string "hallodag".

a\$ - b\$

subtracts the second string from the end of the first. After

```
soundFileName$ = "hallo.aifc"
textgridFileName$ = soundFileName$ - ".aifc" + ".TextGrid"
```

the variable *textgridFileName\$* contains the string "hallo.TextGrid". If the first string *a\$* does not end in the string *b\$*, the result of the subtraction is the string *a\$*.

Links to this page

- [Formulas](#)
-

© *ppgb*, February 11, 2003

Formulas 3. Constants

pi

π , 3.14159265358979323846

e

e , 2.7182818284590452354

undefined

a special value, see undefined

Links to this page

- [Formulas](#)
-

© *ppgb*, February 11, 2003

Formulas 6. Control structures

if ... then ... else ... fi

You can use conditional expressions in all formulas. For example,

```
3 * if 52% * 3809 > 2000 then 5 else 6 fi
```

evaluates to 15. Instead of *fi*, you can also use *endif*.

Another example: you can clip the absolute amplitude of a Sound to 0.5 by supplying the following formula:

```
if abs(self)>0.5 then if self>0 then 0.5 else -0.5 fi else self fi
```

The semicolon

The semicolon ends the evaluation of the formula. This can be convenient if you do not want to overwrite a long formula in your text field: the formula

```
800;sqrt(2)*sin(2*pi*103*0.5)+10^(-40/20)*randomGauss(0,1)
```

evaluates to 800.

Links to this page

- [Formulas](#)

© ppgb, May 19, 2003

binomialQ

A function that can be used in Formulas. The complement of the cumulative binomial distribution.

Syntax

`binomialQ (p, k, n)`
the probability that in n trials an event with probability p will occur at least k times.

Calculator example

A die is suspected to yield more sixes than a perfect die would do. In order to test this suspicion, you throw it 1,000 times. The result is 211 sixes.

The probability that a perfect die yields at least 211 sixes is, according to Calculator..., `binomialQ (1/6, 211, 1000) = 0.000152`.

Script example

You convert 1000 values of pitch targets in Hz to the nearest note on the piano keyboard. 597 of those values turn out to be in the A, B, C, D, E, F, or G regions (the white keys), and 403 values turn out to be in the A#, C#, D#, F#, or G# regions (the black keys). Do our subjects have a preference for the white keys? The following script computes the probability that in the case of no preference the subjects would target the white keys at least 597 times. This is compared with a χ^2 test.

```
a = 597
b = 403
p = 7/12 ; no preference
echo *** Binomial test 'a', 'b', p = 'p:6' ***
pbin = binomialQ (p, a, a+b)
printline P (binomial) = 'pbin:6'
# Chi-square test with Yates correction:
x2 = (a - 1/2 - p * (a+b))^2/(p*(a+b)) + (b + 1/2 - (1-p) *
(a+b))^2/((1-p)*(a+b))
px2 = chiSquareQ (x2, 1)
printline P (chi-square) = 'px2:6'
```

The result is:

```
*** Binomial test 597, 403, p = 0.583333 ***
P (binomial) = 0.199330
P (chi-square) = 0.398365
```

The χ^2 test is two-sided (it signals a preference for the white or for the black keys), so it has twice the probability of the binomial test.

We cannot conclude from this test that people have a preference for the white keys. Of course, we cannot conclude either that people do not have such a preference.

Links to this page

- [Formulas 4. Mathematical functions](#)
-

© *ppgb*, December 4, 2002

VocalTract: Formula...

A command for changing the data in all selected VocalTract objects.

See the Formulas tutorial for examples and explanations.

© *ppgb*, December 6, 2002

FilterBank: Get frequency in Hertz...

A query to the selected FilterBank object.

Return value

a frequency value in Hertz.

© *djmw*, September 1, 2003

Ltas: Get band from frequency...

A query to the selected Ltas object.

Return value

the band number belonging to the specified frequency, expressed as a real number.

Example

If the Ltas has a band width of 1000 Hz, and the lowest frequency is 0 Hz, the band number associated with a frequency of 1800 Hz is 2.3.

Scripting

You can use this command to put the nearest band centre into a script variable:

```
select Ltas hallo
band = Get band from frequency... 1800
nearestBand = round (band)
```

In this case, the value will not be written into the Info window. To round down or up, use

```
leftBand = floor (band)
rightBand = ceiling (band)
```

© ppgb, October 16, 1999

Ltas: Get band width

A query to the selected Ltas object.

Return value

the with of a band (*dx* attribute), expressed in Hertz.

© *ppgb*, October 16, 1999

Ltas: Get frequency from band...

A query to the selected Ltas object what frequency is .

Return value

the frequency (in Hz) associated with a specified band number.

Band number

the band number whose frequency is sought.

Algorithm

the result is

$$f_1 + (\text{bandNumber} - 1) \cdot \Delta f$$

where f_1 is the frequency associated with the centre of the first band (i.e., the $x1$ attribute), and Δf is the band width (the dx attribute of the Ltas object).

© ppgb, October 16, 1999

Ltas: Get frequency of maximum...

A query to the selected Ltas object.

Return value

the frequency (in Hertz) associated with the maximum energy density.

Arguments

From frequency (s), To frequency (s)

the selected frequency domain. Values outside this domain are ignored. If *To frequency* is not greater than *From frequency*, the entire frequency domain of the Ltas object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is None because of the usual large binning. If the Ltas was computed with Spectrum: To Ltas (1-to-1), a Parabolic or Cubic interpolation would be more appropriate.

© ppgb, September 16, 2003

Ltas: Get frequency of minimum...

A query to the selected Ltas object.

Return value

the frequency (in Hertz) associated with the minimum energy density.

Arguments

From frequency (s), To frequency (s)

the selected frequency domain. Values outside this domain are ignored. If *To frequency* is not greater than *From frequency*, the entire frequency domain of the Ltas object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is None because of the usual large binning. If the Ltas was computed with Spectrum: To Ltas (1-to-1), a Parabolic or Cubic interpolation would be more appropriate.

© ppgb, September 16, 2003

Ltas: Get frequency range

A query to the selected Ltas object.

Return value

the frequency range in Hertz, i.e. the difference between the minimum and maximum frequency.

© *ppgb*, October 16, 1999

Ltas: Get highest frequency

A query to the selected Ltas object.

Return value

the highest frequency (*xmax* attribute), expressed in Hertz.

© *ppgb*, October 16, 1999

Ltas: Get lowest frequency

A query to the selected Ltas object for its lowest frequency.

Return value

the lowest frequency (*xmin* attribute), expressed in Hertz. It is normally 0 Hz.

© *ppgb*, October 16, 1999

Ltas: Get maximum...

A query to the selected Ltas object.

Return value

the maximum value (in dB) within a specified frequency range.

Arguments

From frequency (s), To frequency (s)

the selected frequency domain. Values outside this domain are ignored. If *To frequency* is not greater than *From frequency*, the entire frequency domain of the Ltas object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is None because of the usual large binning. If the Ltas was computed with Spectrum: To Ltas (1-to-1), a Parabolic or Cubic interpolation would be more appropriate.

© ppgb, September 16, 2003

Ltas: Get mean...

A query to the selected Ltas object.

Return value

the mean value (in dB) within a specified frequency domain.

Arguments

From frequency (s), *To frequency* (s)

the selected frequency domain. Values outside this domain are ignored. If *To frequency* is not greater than *From frequency*, the entire frequency domain of the Ltas is considered.

Algorithm

The mean value between the frequencies f_1 and f_2 is defined as

$$1/(f_2 - f_1) \int_{f_1}^{f_2} df x(f)$$

where $x(f)$ is the LTAS as a function of frequency, expressed in dB. For our discrete Ltas object, this mean is approximated by

$$1/n \sum_{i=m..m+n-1} x_i$$

where n is the number of band centres between f_1 and f_2 .

© ppgb, October 16, 1999

Ltas: Get minimum...

A query to the selected Ltas object.

Return value

the minimum value (in dB) within a specified frequency range.

Arguments

From frequency (s), To frequency (s)

the selected frequency domain. Values outside this domain are ignored. If *To frequency* is not greater than *From frequency*, the entire frequency domain of the Ltas object is considered.

Interpolation

the interpolation method (None, Parabolic, Cubic, Sinc) of the vector peak interpolation. The standard is None because of the usual large binning. If the Ltas was computed with Spectrum: To Ltas (1-to-1), a Parabolic or Cubic interpolation would be more appropriate.

© ppgb, September 16, 2003

Ltas: Get number of bands

A query to the selected Ltas object.

Return value

the total number of frequency bands (the *nx* attribute).

© *ppgb*, October 16, 1999

Ltas: Get standard deviation...

A query to the selected Ltas object.

Return value

the standard deviation (in dB) of the LTAS within a specified frequency domain.

Arguments

From frequency (s), *To frequency* (s)

the frequency window. Values outside this domain are ignored. If *To frequency* is not greater than *From frequency*, the entire frequency domain of the Ltas is considered.

Algorithm

The standard deviation between the frequencies f_1 and f_2 is defined as

$$1/(f_2 - f_1) \int_{f_1}^{f_2} df (x(f) - \mu)^2$$

where $x(f)$ is the LTAS as a function of frequency, and μ its mean. For our discrete Ltas object, the standard deviation is approximated by

$$1/(n-1) \sum_{i=m..m+n-1} (x_i - \mu)^2$$

where n is the number of band centres between f_1 and f_2 . Note the "minus 1".

© ppgb, October 16, 1999

Ltas: Get value at frequency...

A query to the selected Ltas object.

Return value

the value (in dB) at a specified frequency. If *frequency* is outside the bands of the Ltas, the result is 0.

Arguments

Frequency (s)

the time at which the value is to be evaluated.

Interpolation

the interpolation method, see vector value interpolation. The standard is None because binning is usually large.

© ppgb, September 16, 2003

Ltas: Get value in band...

A query to the selected Ltas object.

Return value

the LTAS value (in dB) in a specified band. If the band number is less than 1 or greater than the number of bands, the result is 0; otherwise, it is $z [1] [band\ number]$.

Argument

Band number

the band whose value is to be looked up.

© ppgb, October 16, 1999



An abbreviation for Fast Fourier Transform.

Links to this page

- [Matrix](#)

© *ppgb*, November 21, 2001

Eigen & Matrix: Project...

A command to project the columns of the Matrix object onto the eigenspace of the Eigen object.

Arguments

Number of dimensions,

defines the dimension, i.e., the number of rows, of the resulting object.

Algorithm

Project each column of the Matrix on the coordinate system given by the eigenvectors of the Eigen object. This can be done as follows:

$$y_{ji} = \sum_{k=1..numberOfColumns} e_{jk} x_{ki}, \text{ where}$$

y_{ji} is the j -th element of the i -th column of the resulting (matrix) object, e_{jk} is the k -th element of the j -th eigenvector and, x_{ki} is the k -th element of the i -th column of the selected matrix object.

© djmw, April 7, 2004

FFNet: Activation

A Matrix whose elements must be ≥ 0 and ≤ 1 . Classification: the response of a particular layer in a neural net to a Pattern. Learning: the desired response of the output layer in a neural net to a Pattern.

© *djmw*, September 18, 1996

FFNet: Pattern

A Pattern is a Matrix in which each row forms one input pattern (vector) for the neural net.

The number of columns is the dimensionality of the input. The number of rows is the number of patterns.

© *djmw*, September 18, 1996

Matrix: Set value...

A command to change the value of one cell in each selected Matrix object.

Arguments

Row number

the number of the row of the cell whose value you want to change.

Column number

the number of the column of the cell whose value you want to change.

New value

the value that you want the specified cell to have.

© ppgb, March 19, 1998

Create formant table (Peterson & Barney 1952)

A command to create a Table object filled with the fundamental frequency and the first three formant frequency values from 10 American-English monophthongal vowels as spoken in a /h_d/ context by 76 speakers (33 men, 28 women and 15 children). Every vowel was pronounced twice, so that there are 1520 recorded vowels in total.

Table layout

The created table will contain 9 columns:

Column 1, labelled as *Type*

speaker type: "m", "w" or "c" (for *man*, *women* or *child*).

Column 2, labelled as *Sex*

speaker sex: either "m" or "f" (for *male* or *female*).

Column 3, labelled as *Speaker*

speaker id: a number from 1 to 76.

Column 4, labelled as *Vowel*

the vowel name. The following list gives the vowel in a *h_d* context word together with its representation in this column: (*heed*, iy), (*hid*, ih), (*head*, eh), (*had*, ae), (*hod*, aa), (*hawed*, ao), (*hood*, uh), (*who'd*, uw), (*hud*, ah), (*heard*, er).

Column 5, labelled as *IPA*

the IPA notation for the vowels.

Column 6, labelled as *F0*

the fundamental frequency in Hertz.

Column 7, 8 and 9, labelled as *F1*, *F2* and *F3*

the frequencies in Hertz of the first three formants.

We originally downloaded the data from a University of Pennsylvania FTP site, where they were reportedly based on a printed version supplied by Ignatius Mattingly. More details about these data and how they were measured can be found in Peterson & Barney (1952).

© djmw, May 12, 2004

Plomp (1967)

Reinier Plomp (1967): "Frequencies dominant in the perception of pitch of complex sounds." *Journal of the Acoustical Society of America* **42**: 191-198.

Links to this page

- [Create Sound from tone complex...](#)
-

© ppgb, December 15, 2002

Patterson & Wightman (1976)

R. Patterson & F. Wightman (1976): "Residue pitch as a function of component spacing." *Journal of the Acoustical Society of America* **59**: 1450-1459.

Links to this page

- [Create Sound from tone complex...](#)
-

© ppgb, December 15, 2002

gamma-tone

A gamma-tone is the product of a rising polynomial, a decaying exponential function, and a cosine wave.

It can be described with the following formula:

$$\text{gammaTone}(t) = a t^{\gamma-1} e^{-2\pi \cdot \text{bandwidth} \cdot t} \cos(2\pi \cdot \text{frequency} \cdot t + \text{initialPhase}),$$

where γ determines the order of the gamma-tone.

The gammatone function has a monotone carrier (the tone) with an envelope that is a gamma distribution function. The amplitude spectrum is essentially symmetric on a linear frequency scale. This function is used in some time-domain auditory models to simulate the spectral analysis performed by the basilar membrane. It was popularized in auditory modeling by Johannesma (1972). Flanagan (1960) already used it to model basilar membrane motion.

Links to this page

- [Create Sound from gamma-tone...](#)
- [Sound: Filter \(gammatone\)...](#)

© djmw, July 13, 1998

Irino & Patterson (1996)

T. Irino & R.D. Patterson (1996), "A time-domain, level-dependent auditory filter: The gammachirp", *J.Acoust.Soc.Am.* **101**, 412-419.

Links to this page

- [Create Sound from gamma-tone...](#)
-

© *djmw*, January 23, 1998

equivalent rectangular bandwidth

The *equivalent rectangular bandwidth* (ERB) of a filter is defined as the width of a rectangular filter whose height equals the peak gain of the filter and which passes the same total power as the filter (given a flat spectrum input such as white noise or an impulse).

Links to this page

- [Create Sound from gamma-tone...](#)
-

© *djmw*, July 13, 1998

Shepard (1964)

R.N. Shepard (1964), "Circularity in Judgments of Relative Pitch", *J.Acoust.Soc.Am.* **36**, 2346-2353.

Links to this page

- [Create Sound from Shepard tone...](#)
-

© *djmw*, January 14, 1998

Shift-click

One of the ways to control Editors.

How to Shift-click

1. Position the mouse above the object that you want to Shift-click.
2. Press a Shift key.
3. Press and release the (left) mouse button.

Usage in the Praat program

Whereas plain clicking is used for selecting only one object while deselecting all previously selected objects, **Shift-click** is used for selecting a mark, target, or boundary, *without* deselecting the previously selected objects:

- PitchTierEditor
- PointEditor

© *ppgb*, September 13, 1996

Hermes (1988)

D.J. Hermes (1988), "Measurement of pitch by subharmonic summation", *J.Acoust.Soc.Am.* **83**, 257-264.

Links to this page

- [Sound: To Pitch \(shs\)...](#)
-

© *djmw*, January 23, 1998

Willems (1986)

Lei Willems (1986): "Robust formant analysis." *IPO report 529*: 1-25. Eindhoven: Institute for Perception Research.

Links to this page

- [Sound: To Formant \(sl\)...](#)
-

© *ppgb*, December 15, 2002

Markel & Gray (1976)

J.D. Markel & A.H. Gray, Jr. (1976), *Linear Prediction of Speech*, Springer Verlag, Berlin.

Links to this page

- [LFCC: To LPC...](#)
 - [LPC: To LFCC...](#)
 - [Sound: To LPC \(autocorrelation\)...](#)
 - [Sound: To LPC \(covariance\)...](#)
-

© djmw, January 14, 1998

Marple (1980)

L. Marple (1980), " A new autoregressive spectrum analysis algorithm", *IEEE Trans. on ASSP* **28**, 441-454.

Links to this page

- [Sound: To LPC \(marple\)...](#)
-

© *djmw*, January 14, 1998

Resource fork

One of the two *forks* of a Macintosh file (the other is the *data fork*). If a Macintosh file is moved to another system directly, the resource fork is lost. To backup your Macintosh files, use compression, for instance with **DropStuff**TM.

Links to this page

- [Macintosh sound files](#)
-

© *ppgb*, March 16, 2003

MelFilter: To MFCC...

A command to create a MFCC object from each selected MelFilter object.

Mel frequency cepstral coefficients result from the Discrete Cosine Transform of the filterbank spectrum (in dB). The following formula shows the relation:

$$c_i = \sum_{j=1}^N P_j \cos(i\pi/N(j-0.5)),$$

where N represents the number of filters and P_j the power in dB in the j^{th} filter.

This transformation was first used by Davis & Mermelstein (1980).

Links to this page

- [Sound: To MFCC...](#)

© *djmw*, April 4, 2001

TextTier: Add point...

A command to add a point to each selected TextTier.

Arguments

Time (s)

the time at which a point is to be added.

Text

the text of the requested new point.

Behaviour

The tier is modified so that it contains the new point. If a point at the specified time was already present in the tier, nothing happens.

© ppgb, March 24, 1998

Keyboard shortcuts

A way to accelerate the control of Editors in PRAAT.

Purpose

to choose a menu command with the keyboard. All of these commands can also be chosen from a menu.

Command key

When mentioning the *Command key*, this manual refers to the key immediately to the left of the space bar. It is positioned in such a way that the most common keyboard shortcuts (Command-X: Cut, Command-C: Copy, Command-V: Paste, Command-Z: Undo, Command-Q: Quit, Command-W: Close) can be typed easily with one hand.

On Macintosh, this key is labelled with an Apple. On SGI, this key is labelled **Alt**.

Option key

When mentioning the *Option key*, this manual refers to the key immediately to the left of the Command key. In Praat, this key is sometimes used together with the Command key for destructive actions that are the reverse of the actions invoked by using the Command key only. For instance, if Command-T means "add a target at the cursor position", Option-Command-T may mean "remove the selected targets".

On Macintosh, this key may be labelled **Option** or **Alt**. On SGI, this key is labelled **Ctrl**.

© ppgb, March 16, 2003

Read Strings from raw text file...

A command to read a Strings object from a simple text file. Each line is read as a separate string. See Strings for an example.

Links to this page

- [WordList](#)
-

© *ppgb*, May 2, 1999

MFCC: To MelFilter...

A command to reconstruct MelFilter objects from the selected MFCC objects .

Settings

From coefficient, To coefficient
the range of coefficients that will be used in the reconstruction.

Details

The output of the triangular filters in a mel filter bank will be synthesized by applying the inverse cosine transform:

$$P_j = 2/N (c_0/2 + \sum_{j=from}^{to} c_i \cos (j\pi /N (i-0.5))),$$

where N represents the number of filters and c_i the i -th cepstral coefficient.

© djmw, April 7, 2004

CC: To DTW...

You can choose this command after selecting 2 objects with cepstral coefficients (two MFCC's or LFCC's). With this command you perform dynamic time warping.

Algorithm

First we calculate distances between cepstral coefficients:

The distance between frame i (from me) and j (from thee) is:

$$wc \cdot d1 + wle \cdot d2 + wr \cdot d3,$$

where wc , wle & wr are user-supplied weights and

$$d1 = \sum_{k=1..nCoefficients} (c_{ik} - c_{jk})^2$$

$$d2 = (c_{i0} - c_{j0})^2$$

$$d3 = \sum_{k=1..nCoefficients} (r_{ik} - r_{jk})^2, \text{ with}$$

r_{ik} the regression coefficient of the cepstral coefficients from the frames within a time span of dtr seconds. c_{ij} is j -th cepstral coefficient in frame i .

Next we find the optimum path through the distance matrix with a Viterbi-algorithm.

Links to this page

- DTW

© djmw, September 18, 1996

LFCC: To LPC...

You can choose this command after selecting 1 or more LFCC's.

Settings

Number of coefficients

the desired number of linear predictive coefficients.

Behaviour

The transformation from cepstral coefficients to a -coefficients as described in Markel & Gray (1976).

© *djmw*, April 7, 2004

Excitations: Append

You can choose this command after selecting two objects of type Excitations.

A new object is created that contains the second object appended after the first.

© *djmw*, September 18, 1996

Create Vocal Tract from phone...

A way to create a VocalTract object.

Purpose

to translate a phone symbol like [a], [u], etc., into a vocal-tract area function.

Behaviour

The resulting VocalTract will appear in the list of objects, with the same name as the phone.

Algorithm

The area function of the resulting VocalTract is taken from the Russian speaker from Fant (1960).

© *ppgb*, September 8, 1996

Create FFNet (linear outputs)...

Create a FFNet feedforward neural network whose output units are linear.

Arguments

Number of inputs

the dimension of the input of the neural net.

Number of outputs (≥ 1)

the number of different categories that you want the net to learn.

Number of units in hidden layer 1, Number of units in hidden layer 2

determine the number of units in the hidden layers. If you want a neural net with no hidden layers, both numbers have to be 0. If you want a neural net with only 1 hidden layer then one of these numbers has to differ from 0.

© djmw, April 22, 2004

Create FFNet...

Create a new feedforward neural net of type FFNet.

Arguments

Number of inputs

the dimension of the input of the neural net.

Number of outputs (≥ 1)

the number of different categories that you want the net to learn.

Number of units in hidden layer 1, Number of units in hidden layer 2

determine the number of units in the hidden layers. If you want a neural net with no hidden layers, both numbers have to be 0. If you want a neural net with only 1 hidden layer then one of these numbers has to differ from 0.

Links to this page

- [Feedforward neural networks 4. Command overview](#)
-

© djmw, April 20, 2004

Create iris example...

A FFNet feedforward neural net will be created together with two other objects: a Pattern and a Categories. The Pattern will contain the observations in the iris data set, and the Categories will contain the 3 different iris species categorized by numbers.

Arguments

Number of units in hidden layer 1, Number of units in hidden layer 2

determine the number of units in the hidden layers. If you want a neural net with no hidden layers, both numbers have to be 0. If you want a neural net with only 1 hidden layer then one of these numbers has to differ from 0.

For this simple data you can leave both hidden layers empty.

Links to this page

- [Feedforward neural networks 2. Quick start](#)

© *djmw*, April 23, 2004

FFNet & Pattern & Categories: Learn slow...

To learn an association you have to select a FFNet, a Pattern and a Categories object.

Preconditions

The number of columns in a **Pattern** must equal the number of input units of **FFNet**.

Algorithm

Steepest descent

Preconditions

The number of rows in a **Pattern** must equal the number of categories in a **Categories**.

The number of unique categories in a **Categories** must equal the number of output units in **FFNet**.

Links to this page

- [Feedforward neural networks 4. Command overview](#)

© *djmw*, September 18, 1996

FFNet: Draw cost history...

You can choose this command after selecting 1 or more FFNet's.

Arguments

Iteration range

determine the horizontal range of the plot.

Cost range

determine the vertical range of the plot.

Garnish

determines whether a box and axis labels are drawn.

Behaviour

Draws the history of the cost versus iteration number during previous learning.

Links to this page

- [Feedforward neural networks 4. Command overview](#)

© djmw, February 18, 1997

FFNet: Draw topology

You can choose this command after selecting 1 or more FFNet's.

Behaviour

Draws all units and all connections of a feedforward neural net.

Links to this page

- [Feedforward neural networks 4. Command overview](#)
-

© *djmw*, February 18, 1997

FFNet: Draw weights...

Draws the weights in a layer of the selected FFNet feedforward neural net.

Arguments

Layer number

determines the layer.

Garnish

determines whether additional information is drawn.

Behaviour

The weights are arranged in a matrix. The columns of this matrix are indexed by the units in the layer, while the rows are indexed by the units in the previous layer. There is one extra row for the biases. The values of the weights are shown as rectangles. The area of a rectangle is proportional to the value. Negative values are shown as filled black rectangles.

Links to this page

- [Feedforward neural networks 4. Command overview](#)

© *djmw*, April 22, 2004

FFNet: Get number of hidden units...

Queries the selected FFNet for the number of units in a hidden layer.

Argument

Hidden layer number
determines the layer that is queried.

Layer numbering

The number of hidden layers is always one less than the total number of layers in a FFNet. A network with the output units connected to the inputs therefor has only 1 layer, the output layer and no hidden layers.

© *djmw*, April 20, 2004

FFNet: Get number of hidden weights...

Queries the selected FFNet for the number of weights in a hidden layer.

Argument

Hidden layer number
determines the layer that is queried.

© *djmw*, April 20, 2004

FFNet: Get number of inputs

Queries the selected FFNet for the number of inputs.

For a network with only one layer, the inputs are connected directly to the output layer. In a two-layer network the inputs are connected to a hidden layer.

© *djmw*, April 20, 2004

FFNet: Get number of outputs

Queries the selected FFNet for the number of output units in the output layer.

© *djmw*, April 20, 2004

FFNet: Principal components

When you select FFNet and Eigen the decision planes of layer 1 are drawn in the PC-plane.

© *djmw*, September 18, 1996

FFNet: Reset...

You can choose this command after selecting 1 or more FFNet's.

WARNING

This command destroys all previous learning.

Arguments

Range

determines the upper limit of the $[-range, +range]$ interval from which new weights will be randomly selected.

Behaviour

All (selected) weights are reset to random numbers uniformly drawn from the interval $[-range, +range]$. This command also clears the cost history.

Links to this page

- [Feedforward neural networks 4. Command overview](#)

© *djmw*, April 20, 2004

FFNet: Select biases...

Selects only the biases in one particular layer as subject for modification during learning of the FFNet.

Argument

Layer number
determines the layer whose biases.

Behaviour

This command induces very specific behaviour during a following learning phase. Instead of all the weights, only the biases in the specified layer will be changed during learning and the rest of the weights stays fixed.

© *djmw*, April 22, 2004

Pattern & Categories: To FFNet...

Create a new FFNet feedforward neural network. The number of inputs of the newly created FFNet will be equal to the number of columns in the Pattern and the number of outputs will be equal to the number of unique categories in the Categories.

Arguments

Number of units in hidden layer 1, Number of units in hidden layer 2

determine the number of units in the hidden layers. If you want a neural net with no hidden layers, both numbers have to be 0. If you want a neural net with only 1 hidden layer then one of these numbers has to differ from 0.

Links to this page

- [Feedforward neural networks 4. Command overview](#)

© *djmw*, April 22, 2004

FFNet: Categories

The categories for training a neural net with a **Pattern**.

Preconditions

The number of categories in a **Categories** must equal the number of rows in **Pattern**.

© *djmw*, September 18, 1996

CategoriesEditor

An editor for manipulating Categories.

To make a selection, use the left mouse button.

The Ctrl key extends a selection (discontinuously).

The Shift key extends a selection contiguously.

Links to this page

- [Categories: Edit](#)

© *djmw*, September 18, 1996

Categories: Append

You can choose this command after selecting 2 objects of type Categories. A new object is created that contains the second object appended after the first.

© *djmw, September 18, 1996*

Eigen: Get contribution of component...

A command to ask the selected Eigen for the contribution of the j^{th} eigenvalue to the total sum of eigenvalues.

Details

The contribution is defined as:

$$\text{eigenvalue}[j] / \sum_{i=1..numberOfEigenvalues} \text{eigenvalue}[i]$$

© djmw, November 9, 1998

Eigen: Get cumulative contribution of components...

A command to ask the selected Eigen for the contribution of the sum of the eigenvalues[*from..to*] to the total sum of eigenvalues.

Details

The contribution is defined as:

$$\sum_{i=from..to} eigenvalue[i] / \sum_{i=1..numberOfEigenvalues} eigenvalue[i]$$

© djmw, November 9, 1998

Eigen: Get eigenvalue...

A command to query the selected Eigen for the i^{th} eigenvalue.

© *djmw*, February 25, 2004

Eigen: Get eigenvector element...

A command to query the selected Eigen for the j^{th} element of the i^{th} eigenvector.

© *djmw*, February 25, 2004

Create Polynomial...

A command to create an Polynomial from a list of coefficients.

Arguments

Xmin, Xmax

define the domain of the polynomial.

Degree

defines the degree of the basis polynomials.

Coefficients

define the coefficients of the polynomial. The coefficient of the highest power of x comes last.

© djmw, April 7, 2004

LegendreSeries: To Polynomial

A command to transform the selected LegendreSeries object into a Polynomial object.

We find polynomial coefficients c_k such that

$$\sum_{k=1..numberOfCoefficients} c_k x^k = \sum_{k=1..numberOfCoefficients} l_k P_k(x)$$

We use the recurrence relation for Legendre polynomials to calculate these coefficients.

© djmw, June 20, 1999

ChebyshevSeries: To Polynomial

A command to transform the selected ChebyshevSeries object into a Polynomial object.

We find polynomial coefficients c_k such that

$$\sum_{k=1..numberOfCoefficients} c_k x^k = \sum_{k=1..numberOfCoefficients} l_k T_k(x)$$

We use the recurrence relation for Chebyshev polynomials to calculate these coefficients.

© djmw, June 20, 1999

Polynomial: Get function value...

A command to compute $p(x)$ for the selected Polynomial object.

© *djmw*, June 10, 1999

Polynomial: Get minimum...

A command to compute, on a specified interval, the minimum value of the selected Polynomial object.

© *djmw*, June 10, 1999

Polynomial: Get x of minimum...

A command to compute, on a specified interval, the location of the minimum of the selected Polynomial object.

© *djmw*, June 10, 1999

Polynomial: Get maximum...

A command to compute, on a specified interval, the maximum value of the selected Polynomial object.

© *djmw*, June 10, 1999

Polynomial: Get x of maximum...

A command to compute, on a specified interval, the location of the maximum of the selected Polynomial object.

© *djmw*, June 10, 1999

Polynomial: Get area...

A command to compute the area below the selected Polynomial object.

Arguments

Xmin, Xmax
define the interval.

The area is defined as $\int_{xmin}^{xmax} p(x) dx$.

© djmw, June 10, 1999

Polynomial: To Polynomial (derivative)

A command to compute the derivative of the selected Polynomial object.

© *djmw*, June 10, 1999

Polynomial: To Polynomial (primitive)

A command to compute the primitive of the selected Polynomial object.

© *djmw*, June 10, 1999

Polynomial: To Roots

A command to compute the roots of the selected Polynomial objects.

Algorithm

The roots are found from the polished eigenvalues of a special companion matrix. For further explanation on these methods see Press et al. (1992).

© *djmw*, June 8, 1999

Polynomial: Scale x...

A command to transform the selected Polynomial object to a new domain.

$Xmin, Xmax$

define the new domain

Behaviour

The polynomial is transformed from domain $[x_{\min}, x_{\max}]$ to domain $[Xmin, Xmax]$ in such a way that its form stays the same. This is accomplished by first calculating:

$f(x') = \sum_{k=1..numberOfCoefficients} c_k x'^k$, where

$$x' = a x + b,$$

and then collecting terms of equal degree. The a and b are defined as

$$a = (x_{\min} - x_{\max}) / (Xmin - Xmax)$$

$$b = x_{\min} - a Xmin$$

© djmw, June 10, 1999

Polynomials: Multiply

A command to multiply two polynomials with each other.

The result of multiplying $1 + 2x$ and $2 - x^2$ will be the polynomial:

$$2 + 4x - x^2 - 2x^3.$$

© *djmw*, June 16, 1999

Chebyshev polynomials

The Chebyshev polynomials $T_n(x)$ of degree n are special orthogonal polynomial functions defined on the domain $[-1, 1]$.

Orthogonality:

$$\int_{-1}^1 W(x) T_i(x) T_j(x) dx = \delta_{ij}$$

$$W(x) = (1 - x^2)^{-1/2} \quad (-1 < x < 1)$$

They obey certain recurrence relations:

$$T_n(x) = 2x T_{n-1}(x) - T_{n-2}(x)$$

$$T_0(x) = 1$$

$$T_1(x) = x$$

Links to this page

- [ChebyshevSeries](#)
- [ChebyshevSeries: To Polynomial](#)
- [Create ChebyshevSeries...](#)

© *djmw*, June 20, 1999

Create ChebyshevSeries...

A command to create a ChebyshevSeries from a list of coefficients.

Arguments

Xmin, Xmax

define the domain of the polynomials.

Coefficients

define the coefficients of each Chebyshev polynomial. The coefficient of the polynomial with the highest degree comes last.

© djmw, April 7, 2004

Legendre polynomials

The Legendre polynomials $P_n(x)$ of degree n are special orthogonal polynomial functions defined on the domain $[-1, 1]$.

Orthogonality:

$$\int_{-1}^1 W(x) P_i(x) P_j(x) dx = \delta_{ij}$$

$$W(x) = 1 \quad (-1 < x < 1)$$

They obey certain recurrence relations:

$$n P_n(x) = (2n - 1) x P_{n-1}(x) - (n - 1) P_{n-2}(x)$$

$$P_0(x) = 1$$

$$P_1(x) = x$$

We may *change* the domain of these polynomials to $[x_{min}, x_{max}]$ by using the following transformation:

$$x' = (2x - (x_{max} + x_{min})) / (x_{max} - x_{min}).$$

We subsequently use $P_k(x')$ instead of $P_k(x)$.

Links to this page

- [Create LegendreSeries...](#)
- [LegendreSeries](#)
- [LegendreSeries: To Polynomial](#)

© djmw, June 20, 1999

Create LegendreSeries...

A command to create a LegendreSeries from a list of coefficients.

Arguments

Xmin, Xmax

define the domain of the polynomials.

Coefficients

define the coefficients of each Legendre polynomial. The coefficient of the polynomial with the highest degree comes last.

© djmw, April 7, 2004

Create ISpline...

A command to create an ISpline from a list of coefficients.

Arguments

Xmin, Xmax

define the domain of the polynomial spline.

Degree

defines the degree of the polynomial spline.

Coefficients

define the coefficients of the basis polynomials.

Interior knots

define the positions in the domain where continuity conditions are defined.

Behaviour

The number of coefficients and the number of interior knots must satisfy the following relation:

$$\text{numberOfCoefficients} = \text{numberOfInteriorKnots} + \text{degree}$$

© djmw, April 7, 2004

Create MSpline...

A command to create an MSpline from a list of coefficients.

Arguments

Xmin, Xmax

define the domain of the polynomial spline.

Degree

defines the degree of the polynomial spline.

Coefficients

define the coefficients of the basis polynomials.

Interior knots

define the positions in the domain where continuity conditions are defined.

Behaviour

The number of coefficients and the number of interior knots must satisfy the following relation:

$$\text{numberOfCoefficients} = \text{numberOfInteriorKnots} + \text{degree} + 1$$

© djmw, April 7, 2004

DTW: Get time along path...

Queries the selected DTW object for the time along the minimal path given the time along the "x-direction".

Arguments

Time

the time along the "x-direction".

In case of ambiguity choose,

determines which time to return when several possibilities exist. Ambiguity arises when the path at chosen time frame has insertions (a plot of the path will then show a line parallel to the y-direction in this time frame). The options *highest* and *lowest* return the highest and lowest value of this parallel line. With no insertions present, the returned times will be exactly equal for both options.

Behaviour

When the *input* time is in the interval $[xmin, xmax]$, the *returned* time will be in the interval $[ymin, ymax]$, where $[xmin, xmax]$ and $[ymin, ymax]$ are the domains of the two "objects" from which the DTW-object was constructed. For all other input times we assume that the two object are aligned. The returned time value will therefor simply equal the input time.

© djmw, April 7, 2004

Weller & Romney (1990)

S.C. Weller & A.K. Romney (1990) *Metric Scaling: Correspondence Analysis*, Sage University Paper Series on Quantitative Applications in the Social Sciences 07-075, Newbury Park, CA: Sage.

Links to this page

- [Correspondence analysis](#)
-

© djmw, December 16, 1997

Gifi (1990)

A. Gifi (1990), *Nonlinear Multivariate Analysis*, John Wiley & Sons Ltd., reprint 1996

Links to this page

- [ContingencyTable: To Configuration \(ca\)...](#)
- [Correspondence analysis](#)

© djmw, December 7, 1997

AffineTransform: Invert

Get the inverse of the selected AffineTransform object.

The inverse from

$$\mathbf{y} = \mathbf{A} \mathbf{x} + \mathbf{t}$$

is:

$$\mathbf{x} = \mathbf{A}^{-1} \mathbf{y} - \mathbf{A}^{-1} \mathbf{t}.$$

© *djmw*, October 8, 2001

CCA: Get zero correlation probability...

Get the probability that for the selected CCA object the chosen canonical correlation coefficient is different from zero.

Arguments

Index

is the index of the canonical correlation coefficient that you want to test.

Algorithm

Wilks' statistic: the probability that coefficient ρ_{index} differs from zero is

$$probability = \text{chiSquareQ}(\chi^2, ndf),$$

where the *number of degrees of freedom* parameter equals

$$ndf = (n_y - index + 1)(n_x - index + 1)$$

and the chi-squared parameter is

$$\chi^2 = -(numberOfObservations - (n_y + n_x + 3)/2) \log(\Lambda_{index}),$$

In the formulas above the variables n_y and n_x are the dimensions of the dependent and the independent data sets whose canonical correlations have been obtained, and Wilks' lambda is:

$$\Lambda_{index} = \prod_{i=index..min(ny,nx)} (1 - \rho_i^2)$$

© djmw, April 7, 2004

confidence interval

The confidence interval gives an estimated range of values which is likely to include an unknown population parameter. The estimated range is calculated from a given set of observations.

Examples

At the α level of significance a two sided confidence interval for the true mean μ for normally distributed data with mean *mean* and known standard deviation σ can be constructed as:

$$mean - z_{\alpha/2} \sigma / \sqrt{N} \leq \mu \leq mean + z_{\alpha/2} \sigma / \sqrt{N},$$

where $z_{\alpha/2} = \text{invGaussQ}(\alpha/2)$ and N is the number of observations.

If the standard deviation is *not* known, we have to estimate its value (s) from the data and the formula above becomes:

$$mean - t_{\alpha/2;N} s / \sqrt{N} \leq \mu \leq mean + t_{\alpha/2;N} s / \sqrt{N},$$

where $t_{\alpha/2;N} = \text{invStudentQ}(\alpha/2, N-1)$.

For $\alpha=0.05$ and $N=20$ we get $z_{0.025}=1.96$ and $t_{0.025;20}=2.039$. This shows that when we have to estimate the standard deviation from the data, the confidence interval is wider than when the standard deviation is known beforehand.

Links to this page

- [confidence level](#)
- [Correlation: Confidence intervals...](#)

© djmw, November 5, 2001

confidence level

The confidence level is the probability value $1-\alpha$ associated with a confidence interval, where α is the level of significance. It can also be expressed as a percentage $100(1-\alpha)\%$ and is than sometimes called the *confidence coefficient*.

Links to this page

- [Correlation: Confidence intervals...](#)
-

© djmw, November 5, 2001

Bonferroni correction

In general, if we have k independent significance tests at the α level, the probability p that we will get no significant differences in all these tests is simply the product of the individual probabilities: $(1 - \alpha)^k$. For example, with $\alpha = 0.05$ and $k = 10$ we get $p = 0.95^{10} = 0.60$. This means, however, we now have a 40% chance that one of these 10 tests will turn out significant, despite each individual test only being at the 5% level. In order to guarantee that the overall significance test is still at the α level, we have to adapt the significance level α' of the individual test.

This results in the following relation between the overall and the individual significance level:

$$(1 - \alpha')^k = 1 - \alpha.$$

This equation can easily be solved for α' :

$$\alpha' = 1 - (1 - \alpha)^{1/k},$$

which for small α reduces to:

$$\alpha' = \alpha / k$$

This is a very simple recipe: If you want an overall significance level α and you perform k individual tests, simply divide α by k to obtain the significance level for the individual tests.

Links to this page

- [Correlation: Confidence intervals...](#)

© djmw, November 7, 2001

Boomsma (1977)

A. Boomsma (1977), "Comparing approximations of confidence intervals for the product-moment correlation coefficient", *Statistica Neerlandica* **31**, 179-186.

Links to this page

- [Correlation: Confidence intervals...](#)
-

© *djmw*, May 24, 2002

Bartlett (1954)

M.S. Bartlett(1954), "A note on multiplying factors for various chi-squared approximations", *Journal of the Royal Statistical Society, Series B*, vol. 16, 296-298

Links to this page

- [SSCP: Get diagonality \(bartlett\)...](#)
-

© *djmw*, November 11, 2001

Regular expressions 1. Special characters

The following characters are the *meta* characters that give special meaning to the regular expression search syntax:

\ the backslash *escape* character.

The backslash gives special meaning to the character following it. For example, the combination "\n" stands for the *newline*, one of the control characters. The combination "\w" stands for a "word" character, one of the convenience escape sequences while "\1" is one of the substitution special characters.

Example: The regex "aa\n" tries to match two consecutive "a"s at the end of a line, inclusive the newline character itself.

Example: "a+" matches "a+" and not a series of one or "a"s.

^ the caret is the start of line anchor or the negate symbol.

Example: "^a" matches "a" at the start of a line.

Example: "[^0-9]" matches any non digit.

\$ the dollar is the end of line anchor.

Example: "b\$" matches a "b" at the end of a line.

Example: "^b\$" matches the empty line.

{ } the open and close curly bracket are used as range quantifiers.

Example: "a{2,3}" matches "aa" or "aaa".

[] the open and close square bracket define a character class to match a *single* character.

The "^" as the first character following the "[" negates and the match is for the characters *not* listed. The "-" denotes a range of characters. Inside a "[]" character class construction most special characters are interpreted as ordinary characters.

Example: "[d-f]" is the same as "[def]" and matches "d", "e" or "f".

Example: "[a-z]" matches any lowercase characters in the alfabet.

Example: "[^0-9]" matches any character that is not a digit.

Example: A search for "[()<>.*?]" in the string "[()<>.*?" followed by a replace string "r" has the result "rrrrrrrrrr". Here the search string is *one* character class and all the meta characters are interpreted as ordinary characters without the need to escape them.

() the open and close parenthesis are used for grouping characters (or other regex).

The groups can be referenced in both the search and the substitution phase. There also exist some special constructs with parenthesis.

Example: "(ab)\1" matches "abab".

. the dot matches any character except the newline.

Example: ".a" matches two consecutive characters where the last one is "a".

Example: ".*\txt\$" matches all strings that end in ".txt".

* the star is the match-zero-or-more quantifier.

Example: "^.*\$" matches an entire line.

+ the plus is the match-one-or-more quantifier.

? the question mark is the match-zero-or-one quantifier. The question mark is also used in special constructs with parenthesis and in changing match behaviour.

| the vertical pipe separates a series of alternatives.

Example: "(a|b|c)a" matches "aa" or "ba" or "ca".

< > the smaller and greater signs are anchors that specify a left or right word boundary.

- the minus indicates a range in a character class (when it is not at the first position after the "[" opening bracket or the last position before the "]" closing bracket.

Example: "[A-Z]" matches any uppercase character.

Example: "[A-Z-]" or "[-A-Z]" match any uppercase character or "-".

& the and is the "substitute complete match" symbol.

Links to this page

- [Regular expressions](#)
-

© *djmw*, July 18, 2001

Regular expressions 2. Quantifiers

Quantifiers specify how often the preceding regular expression should match.

* Try to match the preceding regular expression zero or more times.

Example: "(ab)c*" matches "ab" followed by zero or more "c"s, i.e., "ab", "abc", "abcc", "abccc" ...

+ Try to match the preceding regular expression one or more times.

Example: "(ab)c+" matches "ab" followed by one or more "c"s, i.e., "abc", "abcc", "abccc" ...

{*m*, *n*} Try to match the preceding regular expression between *m* and *n* times.

If you leave *m* out, it is assumed to be zero. If you leave *n* out it is assumed to be infinity. I.e., "{,*n*}" matches from *zero* to *n* times, "{*m*,}" matches a minimum of *m* times, "{,}" matches the same as "*" and "{*n*}" is shorthand for "{*n*, *n*}" and matches exactly *n* times.

Example: "(ab){1,2}" matches "ab" and "abab".

? Try to match zero or one time.

Changing match behaviour

Default the quantifiers above try to match as much as possible, they are *greedy*. You can change greedy behaviour to *lazy* behaviour by adding an extra "?" after the quantifier.

Example: In the string "cabddde", the search "abd{1,2}" matches "abdd", while the search for "abd{1,2}?" matches "abd".

Example: In the string "cabddde", the search "abd+" matches "abddd", while the search for "abd+?" matches "abd".

Links to this page

- [Regular expressions 1. Special characters](#)

© djmw, July 8, 2001

Regular expressions 3. Anchors

Anchors let you specify a very specific position within the search text.

^ Try to match the (following) regex at the beginning of a line.

Example: "**^**ab" matches "ab" only at the beginning of a line and not, for example, in the line "cab".

\$ Try to match the (following) regex at the end of a line.

< Try to match the regex at the *start* of a word.

The character class that defines a *word* can be found at the convenience escape sequences page.

> Try to match the regex at the *end* of a word.

\B Not a word boundary

- Regular expressions
- Regular expressions 1. Special characters

© djmw, July 8, 2001

Regular expressions 4. Special constructs with parenthesis

Some special constructs exist with parenthesis.

(?:*regex*) is a grouping-only construct.

They exist merely for efficiency reasons and facilitate grouping.

(?=*regex*) is a positive look-ahead.

A match of the regular expression contained in the positive look-ahead construct is attempted. If the match succeeds, control is passed to the regex following this construct and the text consumed by this look-ahead construct is first unmatched.

(?!*regex*) is a negative look-ahead.

Functions like a positive look-ahead, only the *regex* must *not* match.

Example: "abc(?!.*abc.*)" searches for the *last* occurrence of "abc" in a string.

(?*iregex*)** is a case insensitive regex.

(?*Iregex*)** is a case sensitive regex.

Default a regex is case sensitive.

Example: "(?iaa)" matches "aa", "aA", "Aa" and "AA".

(?*nregex*)** matches newlines.

(?*Nregex*)** doesn't match newlines.

All the constructs above do not capture text and can not be referenced, i.e., the parenthesis are not counted. However, you can make them capture text by surrounding them with *ordinary* parenthesis.

Links to this page

- [Regular expressions](#)
- [Regular expressions 1. Special characters](#)

© djmw, July 10, 2001

Regular expressions 5. Special control characters

Special control characters in a regular expression specify characters that are difficult to type.

\a alert (bell).

\b backspace.

\e ASCII escape character.

\f form feed (new page).

\n newline.

\r carriage return.

Example : a search for "\r\n" followed by a replace "\r" changes Windows text files to Macintosh text files.

Example : a search for "\r" followed by a replace "\n" changes Macintosh text files to Unix text files.

Example : a search for "\r\n" followed by a replace "\n" changes Windows text files to Unix text files.

\t horizontal tab.

\v vertical tab.

Links to this page

- [Regular expressions 1. Special characters](#)
- [Regular expressions 8. Substitution special characters](#)

© *djmw*, July 8, 2001

Regular expressions 6. Convenience escape sequences

Convenience escape sequences in a regular expression present a shorthand for some character classes.

`\d` matches a digit: `[0-9]`.

Example: `"-?\d+"` matches any integer.

`\D` *not* a digit: `[^0-9]`.

`\l` a letter: `[a-zA-Z]`.

`\L` *not* a letter: `[^a-zA-Z]`.

`\s` whitespace: `[\t\n\r\f\v]`.

`\S` *not* whitespace: `[^\t\n\r\f\v]`.

`\w` "word" character: `[a-zA-Z0-9_]`.

Example: `"\w+"` matches a "word", i.e., a string of one or more characters that may consist of letters, digits and underscores.

`\W` *not* a "word" character: `[^a-zA-Z0-9_]`.

`\b` any character that is *not* a word-delimiter.

Links to this page

- [Regular expressions 1. Special characters](#)
- [Regular expressions 3. Anchors](#)

© *djmw*, July 8, 2001

Regular expressions 7. Octal and hexadecimal escapes

An octal number can be represented by the octal escape "\0" and maximally three digits from the digit class [0-7]. The octal number should not exceed \0377.

A hexadecimal number can be represented by the octal escape "\x" or "\X" and maximally two characters from the class [0-9A-F]. The maximum hexadecimal number should not exceed \xFF.

Example: \053 and \X2B both specify the "+" character.

Links to this page

- [Regular expressions](#)
- [Regular expressions 8. Substitution special characters](#)

© *djmw*, July 9, 2001

Regular expressions 8. Substitution special characters

The substitution string is mostly interpreted as ordinary text except for the special control characters, the octal and hexadecimal escapes and the following character combinations:

`\1 ... \9` are backreferences at sub-expressions 1 ... 9 in the match.

Any of the first nine sub-expressions of the match string can be inserted into the replacement string by inserting a ``` followed by a digit from 1 to 9 that represents the string matched by a parenthesized expression within the regular expression. The numbering is left to right.

Example: A search for `"(a)(b)"` in the string `"abc"`, followed by a replace `"\2\1"` results in `"bac"`.
`&` reference at entire match.

The entire string that was matched by the search operation will be substituted.

Example: a search for `"."` in the string `"abcd"` followed by the replace `"&&"` doubles every character in the result `"aabbccdd"`.

`\U \u` to uppercase.

The text inserted by `"&"` or `"\1" ... "\9"` is converted to *uppercase* (`"\u"` only changes the *first* character to uppercase).

Example: A search for `"(aa)"` in the string `"aabb"`, followed by a replace `"\U\1bc"` results in the string `"AAbcbb"`.

`\L \l` to lowercase.

The text inserted by `"&"` or `"\1" ... "\9"` is converted to *lowercase* (`"\l"` only changes the *first* character to lowercase).

Example: A search for `"(AA)"` with a replace `"\l\1bc"` in the string `"AAbb"` results in the string `"aAbcbb"`.

Links to this page

- [Regular expressions](#)
- [Regular expressions 1. Special characters](#)

© djmw, July 8, 2001

Friedl (1997)

J.E.F. Friedl (1997), *Mastering Regular Expressions*, O'Reilly & Associates.

Links to this page

- [Regular expressions](#)
-

© *djmw*, July 10, 2001

box plot

A box plot provides a simple graphical summary of data. These plots originate from the work of Tukey (1977).

Definitions

The following figure shows an annotated box plot.

[sorry, no pictures yet in the web version of this manual]

To understand the box plot we need the following definitions:

$q25$ = lower quartile, 25% of the data lie below this value

$q50$ = median, 50% of the data lie below this value

$q75$ = upper quartile, 25% of the data lie above this value

The following definitions all depend on these quantiles:

$hspread = |q75 - q25|$ (50% interval)

$lowerOuterFence = q25 - 3.0 * hspread$ (not in figure)

$lowerInnerFence = q25 - 1.5 * hspread$ (not in figure)

$upperInnerFence = q75 + 1.5 * hspread$

$upperOuterFence = q75 + 3.0 * hspread$

$lowerWhisker$ = smallest data value larger than $lowerInnerFence$

$upperWhisker$ = largest data value smaller than $upperInnerFence$

The box plot is a summary of the data in which:

- the horizontal lines of the rectangle correspond to $q25$, $q50$ and $q75$, respectively.
- the dotted line corresponds to the mean.
- the outliers outside the *outerFences* are drawn with an 'o'.
- the outliers in the intervals ($lowerOuterFence$, $lowerInnerFence$) and ($upperInnerFence$, $upperOuterFence$) are drawn with an '*'.
- the whisker lines outside the rectangle connect $q25$ with $lowerWhisker$, and, $q75$ with $upperWhisker$, respectively. With no outliers present, the whiskers mark minimum and/or maximum of the data.

Links to this page

- [TableOfReal: Draw box plots...](#)

differenceLimensToPhon

A routine for converting intensity difference limens into sensation level, the inverse of `phonToDifferenceLimens`.

Formula

$$\text{differenceLimensToPhon}(ndli) = \ln(1 + ndli / 30) / \ln(61 / 60)$$

Links to this page

- [Formulas 4. Mathematical functions](#)
-

© *ppgb*, December 15, 2002

iris data set

A data set with 150 random samples of flowers from the iris species *setosa*, *versicolor*, and *virginica* collected by Anderson (1935). From each species there are 50 observations for sepal length, sepal width, petal length, and petal width in cm. This dataset was used by Fisher (1936) in his initiation of the linear-discriminant-function technique.

Links to this page

- [Create iris example...](#)
- [Feedforward neural networks 2. Quick start](#)

© djmw, October 15, 1996

FAQ (Frequently Asked Questions)

FAQ: Formant analysis

FAQ: Pitch analysis

FAQ: Spectrograms

FAQ: Scripts

© *ppgb*, August 24, 2001

Quantile algorithm

An algorithm to compute the specified quantile of a sorted array of real numbers.

The $n\%$ *quantile* of a continuous real-valued distribution is the value below which $n\%$ of the values is expected to lie. If we are given an array of real numbers that we want to interpret as having been drawn from a distribution, we can *estimate* the quantiles of the underlying distribution.

1. The median

The *median* is a special case of a quantile: it is the 50% quantile. It is usually estimated as follows: from an odd number of values, take the middle value; from an even number, take the average of the two midmost values. For instance, if our values are 15, 20, and 32, the median is 20; if our values are 15, 20, 32, and 60, the median is 26.

This estimate is direction-independent: if we multiply all values by -1 (i.e., they become -60, -32, -20, and -15), the median is also multiplied by -1 (it becomes -26).

2. Percentiles?

The n th *percentile* of a set of values is usually defined as the highest attested value for which at most $n\%$ of all attested values are less or equal. For instance, if our values are 15, 20, 32, and 60, the 30th percentile is 15. Here is an extensive list:

Percentile number Value

0-

10-

20-

30 15

40 15

50 20

60 20

70 20

8032

9032

10060

However, this procedure does not yield an estimate of the quantiles of the underlying distribution. For instance, the estimate is direction-dependent: if we multiply all values by -1, the 50th percentile becomes -32 instead of -20, and the 70th percentile becomes -32 instead of the expected -15, which is minus the 30th percentile of the original data set.

3. Unbiased quantiles

To get a better estimate of the quantiles of the underlying distribution, the interpolation that we used to determine the median, is generalized to *any* quantile.

We assume that the attested values 15, 20, 32, and 60 each take up one quarter of the "quantile space". These four values are in the middles of those quarters, so they are at the 0.125, 0.375, 0.625, and 0.875 quantiles.

Quantiles in between 0.125 and 0.875 are evaluated by linear interpolation: the 0.25, 0.50, and 0.75 quantiles are 17.5, 26, and 46, respectively. Note that the 0.50 quantile is the median. The 0.40 quantile, for example, is estimated as $20 + (32 - 20) \cdot (0.40 - 0.375) / (0.625 - 0.375) = 21.2$.

Quantiles between 0 and 0.125 or between 0.875 and 1 are evaluated by linear extrapolation from the lowest or highest pair of values: the 0% quantile is estimated as $15 - 1/2 (20 - 15) = 12.5$, and the 100% quantile is estimated as $60 + 1/2 (60 - 32) = 74$. The 0.10 quantile is estimated as $12.5 + (15 - 12.5) \cdot (0.10 - 0.0) / (0.125 - 0.0) = 14.5$.

Note that the estimated values for the very low or high quantiles can lie outside the range of attested values. In fact, the computed 0% and 100% quantiles are thought to be estimates of the minimum and maximum values of the distribution. For uniform distributions, these estimates are reasonable; for a normal distribution, of course, the 0% and 100% quantiles are meaningless.

Links to this page

- Formant: Get quantile...

© ppgb, January 1, 1998

Wakita (1977)

H. Wakita (1977), "Normalization of vowels by vocal-tract length and its application to vowel identification", *IEEE Trans. on ASSP* **25**, 183-192.

Links to this page

- [LPC: To VocalTract \(slice\)...](#)
-

© *djmw*, January 14, 1998

Jesteadt, Wier & Green (1977)

W. Jesteadt, G.C. Wier, & D.M. Green (1977): "Intensity discrimination as a function of frequency and sensation level." *Journal of the Acoustical Society of America* **61**: 169-177.

Links to this page

- [phonToDifferenceLimens](#)
-

© *ppgb*, December 15, 2002

Johnson (1998)

D.E. Johnson (1998), *Applied Multivariate methods*

Links to this page

- [SSCP: Get confidence ellipse...](#)
-

© *djmw*, May 25, 2000

invFisherQ

`invFisherQ(q, df1, df2)` returns the value f for which `fisherQ(f, df1, df2) = q`.

Links to this page

- [SSCP: Get confidence ellipse...](#)
-

© *djmw*, May 25, 2000

Kaiser (1958)

H.F. Kaiser (1958), "The varimax criterion for analytic rotation in factor analysis", *Psychometrika* **23**, 187-200.

Links to this page

- [Configuration: To Configuration \(varimax\)...](#)
-

© *djmw*, April 4, 1998

Ten Berge (1995)

J.M.F. ten Berge (1995), "Suppressing permutations or rigid planar rotations: A remedy against nonoptimal varimax rotations", *Psychometrika* **60**, 437-446.

Links to this page

- [Configuration: To Configuration \(varimax\)...](#)
-

© *djmw*, April 4, 1998

procrustus transform

A transformation that only uses a combination of a translation, a scaling and a rigid transformation to transform one Configuration such that it matches as closely as possible another Configuration.

For more information about the Procrustus transform and its algorithm see chapter 19 in Borg & Groenen (1997).

Links to this page

- [Configuration: To Configuration \(procrustus\)](#)
 - [Configurations: To AffineTransform \(congruence\)...](#)
 - [Configurations: To Procrustus](#)
-

© *djmw*, January 19, 1998

Brokken (1983)

F.B. Brokken (1983), "Orthogonal Procrustus rotation maximizing congruence", *Psychometrika* **48**, 343-352.

Links to this page

- [Configurations: To AffineTransform \(congruence\)...](#)
-

© *djmw*, April 6, 1998

Kiers & Groenen (1996)

H.A.L. Kiers & P. Groenen (1996), "A monotonically convergent algorithm for orthogonal congruence rotation", *Psychometrika* **61**, 375-389.

Links to this page

- [Configurations: To AffineTransform \(congruence\)...](#)
-

© *djmw*, December 19, 1997

Young, Takane & Lewyckyj (1978)

F.W. Young, Y. Takane & R. Lewyckyj (1978), "Three notes on ALSCAL", *Psychometrika* **43**, 433-435.

Links to this page

- CANDECOMP
 - Distance: To Configuration (ytl)...
-

© djmw, December 1, 1997

Cailliez (1983)

F. Cailliez (1983), "The analytical solution of the additive constant problem", *Psychometrika* **48**, 305-308.

Links to this page

- Dissimilarity: Get additive constant
-

© djmw, December 1, 1997

Abramowitz & Stegun (1970)

M. Abramowitz & I. Stegun (1970), *Handbook of Mathematical Functions*, Dover Publications, Inc., New York.

Links to this page

- [concentration ellipse](#)
 - [Confusion: To Dissimilarity \(pdf\)...](#)
-

© djmw, December 1, 1997

smacof

Scaling by Majorizing a Complicated Function, the iterative algorithm to find an optimal Configuration.

1. Initialize
 - 1.a. Get initial Configuration \mathbf{Z}
 - 1.b. Set stress $\sigma_n^{[0]}$ to a very large value.
 - 1.c. Set iteration counter $k = 0$
2. Increase iteration counter by one: $k = k + 1$
3. Calculate distances $d_{ij}(\mathbf{Z})$.
4. Transform dissimilarities δ_{ij} into disparities d'_{ij} .
5. Standardize the disparities so that $\tau_{d'}^2 = n(n-1)/2$.
6. Compute the Guttman transform $\mathbf{X}^{[k]}$ of \mathbf{Z} .
7. Compute new distances $d_{ij}(\mathbf{X}^{[k]})$.
8. Compute normalized stress $\sigma_n(\mathbf{d}', \mathbf{X}^{[k]})$
9. If $|\sigma_n^{[k]} - \sigma_n^{[k-1]}| / \sigma_n^{[k-1]} < \epsilon$ or $k > \text{maximumNumberOfIterations}$, then stop
10. Set $\mathbf{Z} = \mathbf{X}^{[k]}$, and go to 2.

This algorithm goes back to De Leeuw (1977).

© djmw, January 19, 1998

Van Nierop et al. (1973)

D.J.P.J. Van Nierop, L.C.W. Pols & R. Plomp (1973), "Frequency analysis of Dutch vowels from 25 female speakers", *Acustica* **29**, 110-118

Links to this page

- [Create formant table \(Pols & Van Nierop 1973\)](#)
-

© *djmw*, June 20, 2002

Klein, Plomp & Pols (1970)

W. Klein, R. Plomp, & L.C.W. Pols (1970), "Vowel Spectra, Vowel Spaces, and Vowel Identification", JASA 48, 999-1009.

Links to this page

- [Confusion: To Similarity...](#)
-

© *djmw*, December 1, 1997

Feedforward neural networks 4. Command overview

FFNet commands

Creation:

- Pattern & Categories: To FFNet...
- Create FFNet...

Learning:

- FFNet & Pattern & Categories: Learn...
- FFNet & Pattern & Categories: Learn slow...

Classification:

- FFNet & Pattern: To Categories...

Drawing:

- FFNet: Draw topology
- FFNet: Draw weights...
- FFNet: Draw cost history...

Queries

Analysis:

- FFNet & Pattern: To Activation...

Modification:

- FFNet: Reset...
- FFNet: Select biases...
- FFNet: Select all weights

Links to this page

- [Feedforward neural networks](#)
-

© djmw, April 26, 2004

Nocedal & Wright (1999)

J. Nocedal & S.J. Wright, *Numerical optimization*, Springer, 1999.

Links to this page

- [FFNet & Pattern & Categories: Learn...](#)

© *djmw*, May 11, 2004

Preferences file

The file into which some of your preferences are saved across your sessions with PRAAT. For instance, if you change the font used by the Picture window to Palatino and leave PRAAT, the Picture-window font will still be Palatino when you enter PRAAT again.

The preferences file is written to disk when you leave PRAAT, and it is read when you enter PRAAT. It is a simple text file that you can read (but should not edit) with any text editor.

Unix

If your home directory is `/people/miep` the preferences file is `/people/miep/.praat-dir/prefs`. If the directory `.praat-dir` does not exist, it is created when you enter PRAAT.

Macintosh

In MacOS X, the preferences file is called *Prefs*, and it will be in the folder *Praat Prefs* in the *Preferences* folder of your personal *Library* folder. On my iBook, the preferences file is **`/Users/pboersma/Library/Preferences/Praat Prefs/Prefs`**.

In MacOS 7, 8, or 9, the preferences file is also called *Prefs*, and it will be in the folder *Praat Preferences* in the Preferences folder in your System Folder. If your hard disk is called "Hæl' sji'f", and you have a Dutch system, the complete path to your preferences file is:

```
Hæl' sji'f:Stysteemmap:Voorkeuren:Praat Preferences:Prefs
```

Windows

The preferences file is called *Preferences.ini*, and it will be in the directory *Praat* in your Windows directory, or in the directory *Praat* in your home directory if you work on a shared computer. On my Virtual PC Windows XP Home Edition computer, the preferences file is **`C:\Documents and Settings\Paul Boersma\Praat\Preferences.ini`**.

Links to this page

- Initialization script

© ppgb, May 19, 2003

Peterson & Barney (1952)

G.E. Peterson & H.L. Barney (1952), "Control methods used in a study of the vowels", *J.Acoust.Soc.Am.* **24**, 175-184

Links to this page

- [Create formant table \(Peterson & Barney 1952\)](#)
-

© *djmw*, June 20, 2002

Johannesma (1972)

P.I.M. Johannesma (1972): "The pre-response stimulus ensemble of neurons in the cochlear nucleus", in *Symposium on Hearing Theory* (IPO, Eindhoven, Holland), 58-69.

Links to this page

- [gamma-tone](#)
-

© djmw, January 23, 1998

Flanagan (1960)

J.L. Flanagan (1960), "Models for approximating basilar membrane displacement", *Bell Sys. Tech. J.* **39**, 1163-1191.

Links to this page

- [gamma-tone](#)
-

© *djmw*, July 13, 1998

Sound: Filter (gammatone)...

A command to filter a Sound by a fourth order gammatone bandpass filter.

Arguments

Centre frequency, Bandwidth
determine the passband of the filter.

Algorithm

The impulse response of the filter is a 4-th order gamma-tone. This filter is implemented as a simple 8-th order recursive digital filter with 4 zeros and 8 poles (these 8 poles consist of one conjugate pole pair to the 4-th power). In the Z-domain its formula is:

$$H(z) = (1 + \sum_{i=1..4} a_i z^{-i}) / (1 + \sum_{j=1..8} b_j z^{-j})$$

The derivation of the filter coefficients a_i and b_j is according to Slaney (1993). The gain of the filter is scaled to unity at the centre frequency.

© djmw, July 12, 1998

Davis & Mermelstein (1980)

S.B. Davis & P. Mermelstein (1980), "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences", *IEEE Trans. on ASSP* **28**, 357-366.

Links to this page

- [MeIFilter: To MFCC...](#)
-

© *djmw*, April 19, 2001

Fant (1960)

Gunnar Fant (1960): *Acoustic Theory of Speech Production*. Mouton, The Hague.

Links to this page

- [Create Vocal Tract from phone...](#)
-

© ppgb, February 1, 1998

Categories: Edit

You can choose this command after selecting one **Categories**. A CategoriesEditor will appear on the screen, with the selected **Categories** in it.

© *djmw*, September 18, 1996

Tukey (1977)

J.W. Tukey (1977), *Exploratory data analysis*, Addison-Wesley, Reading, Mass.

Links to this page

- [box plot](#)

© *djmw*, May 24, 2000

Anderson (1935)

E. Anderson (1935), "The irises of the Gaspé peninsula", *Bulletin of the American Iris Society* **59**, 2-5.

Links to this page

- [iris data set](#)

© *djmw*, April 23, 2004

Fisher (1936)

R.A. Fisher (1936), "The use of multiple measurements in taxonomic problems", *Annals of Eugenics* **7**, 179-188.

Links to this page

- [iris data set](#)
-

© *djmw*, January 14, 1998

FAQ: Spectrograms

Problem: the background is grey instead of white (too little contrast)

Solution: reduce the "dynamic range" in the spectrogram settings. The standard value is 50 dB, which is fine for detecting small things like plosive voicing in well recorded speech. For gross features like vowel formants, or for noisy speech, you may want to change the dynamic range to 40 or even 30 dB.

Links to this page

- [FAQ \(Frequently Asked Questions\)](#)
-

© *ppgb*, September 16, 2003

fisherQ

`fisherQ(f, df1, df2)` returns the area under Fisher's F-distribution from f to $+\infty$.

Links to this page

- [invFisherQ](#)
-

© *djmw*, May 25, 2000

Slaney (1993)

M. Slaney (1993), "An efficient implementation of the Patterson-Holdsworth auditory filterbank", *Apple Computer Technical Report 35*, 41 pages.

Links to this page

- [Sound: Filter \(gammatone\)...](#)
-

© *djmw*, July 12, 1998