Pyraat Documentation

Release 0.3.0

Michael McAuliffe

Nov 27, 2017

Contents

Contents:

CHAPTER 1

Basic usage

To use Pyraat's Praat functions, the PraatAnalysisFunction is first imported and then constructed as follows:

```python
from pyraat import PraatAnalysisFunction

script_path = '/path/to/script.praat'
praat_path = '/path/to/praat'
wav_file = '/path/to/file.wav'

func = PraatAnalysisFunction(script_path, praat_path)

output = func(wav_file)
```

The `praat_path` argument can be omitted if `praat` is available on the system path.

Note: The above snippet assumes a script that analyzes a whole file. See *Praat script types* for more information on the types of Praat scripts.

Arguments to the Praat script that are constant over multiple evaluations on wav files:

```python
from pyraat import PraatAnalysisFunction

script_path = '/path/to/script.praat'
praat_path = '/path/to/praat'
wav_file = '/path/to/file.wav'
wav_file2 = '/path/to/file2.wav'

func = PraatAnalysisFunction(script_path, praat_path, arguments=[1, 2, 3])

output = func(wav_file)
output2 = func(wav_file2)
```

If the arguments depend on the file, they can be specified on each evaluation:

```python
from pyraat import PraatAnalysisFunction

script_path = '/path/to/script.praat'
praat_path = '/path/to/praat'
wav_file = '/path/to/file.wav'
wav_file2 = '/path/to/file2.wav'

func = PraatAnalysisFunction(script_path, praat_path)

output = func(wav_file, 1, 2, 3)
output2 = func(wav_file2, 4, 5, 6)
```

CHAPTER 2

Praat script types

There are two main types of scripts that can be run. The first loads and analyzes an entire wav file (best for short files), and the second analyzes segments of a wav file without loading all of the file.

1. *Load and analyze full wav file*
2. *Load and analyze just a segment of a wav file*

1. *Output point measures*
2. *Output track measures*

2.1 Load and analyze full wav file

The first supported style of script uses an entire file to generate an output. Examples of such scripts would be calculating the long-term average spectra of the file, or if words/phones have been extracted previously into short wav files.

2.1.1 Header

For shorter wav files, the following header is used to specify arguments. The only required argument is for `filename` as that will point to which file to analyze.

```
form Variables
    sentence filename
    real measurement_point
    integer nformants
    real ceiling
endform
```

2.1.2 Required lines in the script

To load the wav file, use the following style:

```
Read from file... 'filename$'
```

This will load the file into memory (so therefore it's best to use this style with short files). Following this, the rest of your script and processing can be done.

2.2 Load and analyze just a segment of a wav file

The second supported style of script uses just one part of the file from a longer file, ranging anywhere from several minutes to several hours. Loading those longer sound files into memory is prohibitive in many cases, so this style takes advantage of the `Open long sound file` feature of Praat.

2.2.1 Header

For analyzing one segment of a file, the following header is used to specify arguments. The first four arguments are required, specifying the name of the sound file, the begin time of the segment of interest, its end time, which channel to extract (for mono files, this will always be 0 in Python), and any padding to use surrounding the file (often helpful for algorithms that have long windows, such as pitch or intensity.

```
form Variables
    sentence filename
    real begin
    real end
    integer channel
    real padding
    real measurement_point
    integer nformants
    real ceiling
endform
```

2.2.2 Required lines in the script

To load the wav file in the script, use the following style (assuming the header above):

```
Open long sound file... 'filename$'


Extract part... begin end 1
channel = channel + 1
Extract one channel... channel

Rename... segment_of_interest
```

These lines first load the sound file as a "long" sound file, which does not immediately load the file into memory (necessary for longer sound files).

The next several lines extract one part of the file based on the `begin`, `end`, and `channel` arguments from the form above.

Note: The above script assumes that channels are specified from 0 (left = 0, right = 1) in the Python code, whereas Praat begins from 1 (left = 1, right = 2).

Once the segment has been extracted, the final line renames the segment to something more referrable for later in the script (since the default name in Praat for the segment will be a combination of the filename, begin, end, and channel).

2.3 Output types

For all supported Praat scripts, Pyraat inspects the script for a line containing `echo` followed by some variable name that stores the output of the script, i.e.:

```
echo 'output$'
```

From this, Pyraat can see how this variable was built up and where there is a `time` column associated with output, which determines whether the output is a track of values over time or just a single measurement for the whole file (i.e., a point measure or an averaged measure).

2.3.1 Output point measures

If there is no time column in the output variable, then the output type is a point measure. The expected output of a point measure Praat script should look something like the following:

```
Point_measure_name1  Point_measure_name2
30  40.54
```

The output consists of names of the point measures on the first line, separated by white space (any number of spaces or tabs), and the corresponding values on the second line (likewise, separated by white space).

To generate such an output, the Praat script should have something like:

```
cog$ = fixed$(cog, 4)
peak$ = fixed$(peak, 4)
slope$ = fixed$(slope, 4)
spread$ = fixed$(spread, 4)
output$ = "peak slope cog spread" + newline$ + peak$ + " " + slope$ + " "+ cog$+ " "␣
↪+ spread$
echo 'output$'
```

2.3.2 Output track measures

For outputs involving time points, the output should be a track measure, like the following:

```
time measure_name1 measure_name2
0.01 10 20
0.02 11 19
0.03 12 18
```

As above, the columns are separated by white space (any number of tabs or spaces), but there must be one column named `time`.

To generate such an output, the Praat script (i.e., for time series of formants) should look something like:

```
output$ = "time"
for i from 1 to nformants
    formNum$ = string$(i)
    output$ = output$ +tab$+ "F"+formNum$ + tab$ + "B" + formNum$
endfor

for f from 1 to frames
    t = Get time from frame number... 'f'
    t$ = fixed$(t, 3)
    output$ = output$ + t$
    for i from 1 to nformants
        formant = Get value at time... 'i' 't' Hertz Linear
        formant$ = fixed$(formant, 2)
        bw = Get bandwidth at time... 'i' 't' Hertz Linear
        bw$ = fixed$(bw, 2)
        output$ = output$ + tab$ + formant$ + tab$ + bw$
    endfor
    output$ = output$ + newline$
endfor
```

The above part of the script generates an output variable (`output$`) that has the first line as the column headers (containing `time` and a column for each formant and their respective bandwidth up to a number of formants). It then loops through the frames in a Formant object and gets each frame's time point and formant and bandwidth values. These are then added to the output line separated by tabs and each successive frame is separated by a newline.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search