Demonstrating Parselmouth: Integrating Praat into a complex Python workflow

Yannick Jadoul, Bill Thompson, Bart de Boer – Artificial Intelligence Lab, Vrije Universiteit Brussel
Yannick.Jadoul@ai.vub.ac.be

Parselmouth

Interdisciplinary science requires interdisciplinary tools: as different scientific disciplines combine methods and approaches to research language and speech, existing algorithms and tools need to be combined. One of these tools is Praat (Boersma and Weenink, 2018), a software package implementing a wide range of acoustic and phonetic algorithms and analyses. While Python and other scripting languages often allow different parts of the research to be automated and glued together into a single workflow, doing so with Praat and its scripting language is not always as straightforward. To simplify the integration of Praat into a complex workflow, we have developed Parselmouth, a Python interface to Praat.

Why and when use Parselmouth?

- Complex workflows (vs. interactive views in Praat)
- Interactive and adaptive use (vs. preprocessing data)
- Integration with other Python libraries (e.g. NumPy)
- Custom control over data, results, or visualization

Installation and usage

```
$ pip install praat-parselmouth

>>> import parselmouth
```

Parselmouth @ GitHub
https://github.com/YannickJadoul/Parselmouth

Web service example @ ReadTheDocs
https://parselmouth.readthedocs.io/en/latest/examples/web_service.html

Plotting example @ ReadTheDocs
https://parselmouth.readthedocs.io/en/latest/examples/plotting.html

PsychoPy experiments example @ ReadTheDocs
http://parselmouth.readthedocs.io/en/latest/examples/psychopy_experiments.html

Python web server

- Web experiment, e.g. Amazon Mechanical Turk
- Praat functionality on web server
- Participant/client side independent from Praat

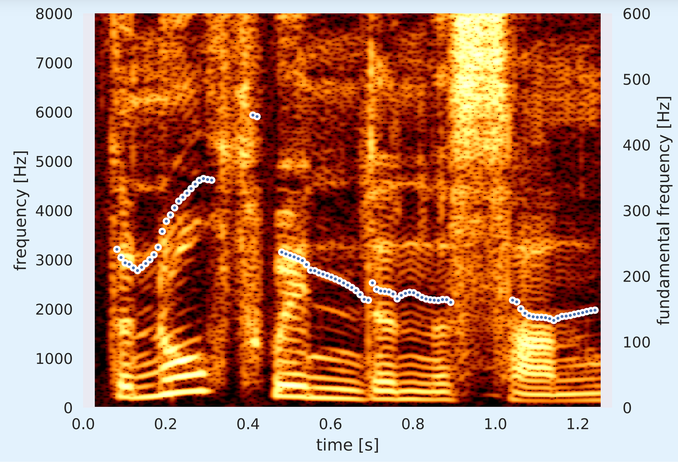
Python server

```
from flask import Flask, request, jsonify
import tempfile

app = Flask(__name__)

@app.route('/pitch_track', methods=['POST'])
def pitch_track():
    import parselmouth
    with tempfile.NamedTemporaryFile() as tmp:
        tmp.write(request.files['audio'].read())
        sound = parselmouth.Sound(tmp.name)
    return jsonify(list(sound.to_pitch().selected_array['frequency']))
```

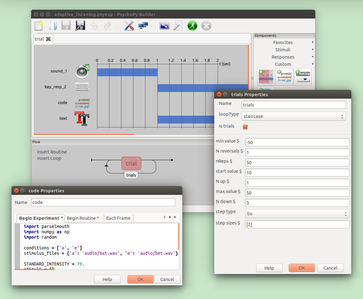
Sample client

```
import requests
import json

files = {'audio': open('the_north_wind_and_the_sun.wav', 'rb')}
reply = requests.post('http://127.0.0.1:5000/pitch_track', files=files)
print(json.loads(reply.text))
```

Custom visualizations in Python



Adaptive PsychoPy experiments



- Experimental framework in Python with graphical user interface
- Insert custom Python code
- Access Praat functionality with Parselmouth
- Generate stimuli on the fly
- Adapt stimuli to responses

Before Experiment

```
import parselmouth
import numpy as np
import random

conditions = ['a', 'e']
stimulus_files = {'a': 'audio/bat.wav', 'e': 'audio/bet.wav'}

STANDARD_INTENSITY = 70.
stimuli = {}
for condition in conditions:
    stimulus = parselmouth.Sound(stimulus_files[condition])
    stimulus.scale_intensity(STANDARD_INTENSITY)
    stimuli[condition] = stimulus
```

Before Routine

```
random_condition = random.choice(conditions)
random_stimulus = stimuli[random_condition]

noise_samples = np.random.normal(size=random_stimulus.n_samples)
noisy_stimulus = parselmouth.Sound(noise_samples,
                    sampling_frequency=random_stimulus.sampling_frequency)
noisy_stimulus.scale_intensity(STANDARD_INTENSITY - level)
noisy_stimulus.values += random_stimulus.values
noisy_stimulus.scale_intensity(STANDARD_INTENSITY)

# 'filename' variable is set by PsychoPy and contains base file name
# of saved log/output files, so we'll use that to save our custom stimuli
stimulus_file_name = filename + '_stimulus_' + str(trials.thisTrialN) + '.wav'
noisy_stimulus.resample(44100).save(stimulus_file_name, "WAV")
sound_1.setSound(stimulus_file_name)
```

After Routine

```
trials.addResponse(key_resp_2.keys == random_condition)
```