UNIVERSITY OF EAST ANGLIA

School of Computing Sciences

CMP-6026A/CMP-7016A – Audio-visual Processing

LABORATORY SHEET 1 – Analysing sounds

Aims

The aim of this lab is to first use Python to record, display, analyse and play speech signals. The signals will be examined both in the time-domain and as spectrograms. Lastly, you'll examine some Python code which can record synchronous audio and video signals. You'll need to think about you might adapt/redevelop this simple program to save multiple speech signals. This will be important for the speech recognition coursework.

1. Recording audio signals in Python

This section will show how audio can be recorded using Python's `sounddevice`. You are strongly recommended to use Spyder to run your Python code but ultimately the choice of development environment is up to you.

The TAs in the lab will provide guidance on the precise steps required to start Spyder. However, in summary, connect to your designated cmprem-XX machine (making sure you've set the audio to both play and record from 'this' machine), start an Anaconda prompt on the Desktop, then type `conda activate c:\anaconda\envs\avp` and then type `spyder`.

Next, connect a microphone to the PC's line in port.

Import the required Python library:

```
import sounddevice as sd
```

Set the sampling frequency to a reasonable value, for example 16,000Hz. Set the recording duration:

```
fs = 16000
seconds = 5
```

When you run the following code, the recording will start and Python will sleep until it is complete. Record yourself speaking digits. This will provide a good signal for further analysis:

```
r = sd.rec(seconds * fs, samplerate=fs, channels=1)
sd.wait()
```

Now play back the audio:

```
sd.play(r, fs)
```

The audio can be plotted using the `plotly` library. We're going to change the renderer to 'browser' so we can make interactive plots (The default renderer will make non-interactive, low-resolution plots):

```
import plotly.io as pio
import plotly.express as px

pio.renderers.default='browser'
fig = px.line(r)
fig.show()
```

Looking at the plot, can you identify the digits and periods of silence?

The x-axis of the plot shows the sample number (labelled as 'index'). One second of audio corresponds to 16,000 samples. It is often more useful to show time in seconds. This can be done by creating a time variable `t`, using the `numpy` library:

```
import numpy as np
t = np.arange(1/fs, 1/fs + len(r)/fs, 1/fs)
```

Have a look at the values this variable contains. For example, to see the first 10 values:

```
t[0:9] #Remember that Python indexing starts at 0
```

Let's create a `pandas` dataframe to store our `t` and `r` variables:

```
import pandas as pd
df = pd.DataFrame(
    {'x': t,
     'y': r.squeeze()})
```

Look at the size of `t` and `r` in Spyder's Variable Explorer. `r.squeeze()` in the code above is removing a redundant dimension in the data.

Now, plot the audio signal against time in seconds:

```
fig = px.line(df, x='x', y='y')
fig.show()
```

Have a look at the online documentation for `plotly` to see what it can do:

https://plotly.com/python/line-charts/

Now add a label for the x-axis:

```
fig = px.line(df, x='x', y='y', labels=dict(x="Time (Seconds)"))
fig.show()
```

Next, add a label for the y-axis.

Use the interactive functions in `plotly` to examine the waveform in more detail.

Now load a wav file containing some speech. Download 'look_out.wav' from Blackboard. The `soundfile` library can read wav files.

```
import soundfile as sf
r2, fs2 = sf.read('look_out.wav', dtype='float32')
```

Python's `matplotlib` library will display the spectrogram of the speech signal:

```
import matplotlib.pyplot as plt
plt.specgram(r2, window=np.hamming(512), noverlap=400, …
NFFT=512, Fs=fs2)
```

This produces a narrowband spectrogram. Refer to the online help to see how this works: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.specgram.html#matplotlib.pyplot.specgram

Now try (note the additional *pad_to* parameter):

```
plt.specgram(r2, window=np.hamming(64),… noverlap=40, …
NFFT=64, pad_to=512, Fs=fs2)
```

This produces a wideband spectrogram. What are the differences?

Measuring your fundamental frequency or pitch

Record yourself saying a vowel sound (e.g. "oo" as in sh**oe**) for a couple of seconds using your normal speaking style.

Create the time variable, `t`, and plot the waveform.

Now zoom into the signal (by clicking and dragging in a `plotly` plot) so you can see cycles of the waveform. You should see something like that shown in Figure 1.

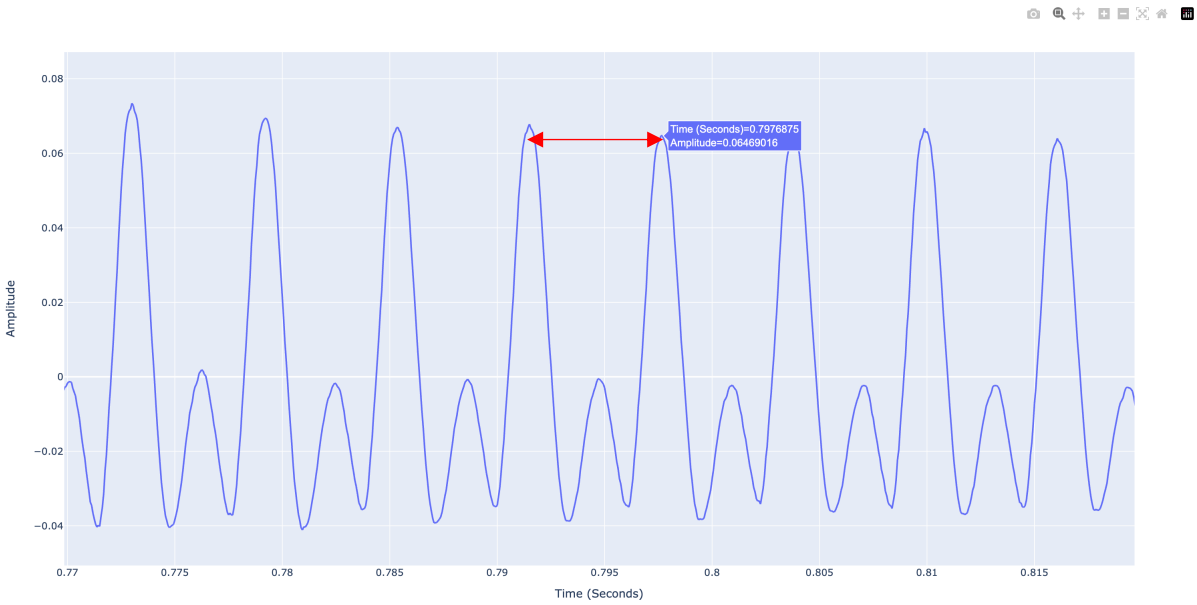

Figure 1: Speech waveform.

Measure the duration of one cycle of the waveform – in the example above:

$$T = 0.7977 – 0.7915 = 0.0062 \text{ seconds}$$

This is known as the *pitch period*.

Now compute the pitch (or *fundamental frequency*):

$$f_0 = 1 / T = 1 / 0.0062 = 161.29 \text{ Hz}$$

What is your fundamental frequency?

Record yourself saying a vowel sound but now start at a low pitch and gradually increase to a high pitch. Measure the range of your fundamental frequency.

You can also use the spectrogram to examine the change in fundamental frequency.

Take the spectrogram of the waveform just recorded. You may need to use `numpy`'s `squeeze` function on your audio before you pass it to `specgram`. This is because microphone data is stored in a matrix with two dimensions (e.g. (numSamples, 1)) whereas single channel audio read from a wav file is one dimensional (e.g. (x,)). `specgram` expects the latter.

Can you see the fundamental frequency changing? Does it change in accordance with your measurements?

2 Saving audio files in Python

The `write` function within the `soundfile` library can be used to save signals as wav files. The signal amplitudes to be saved must be in the range between –1 and +1.

Using the signal from the previous section, use the `min` and `max` functions to determine the dynamic range of signal.

The signal can be normalised using:

```
rNorm = 0.99 * r / max(abs(r))
```

Plot both `r` and `rNorm` to see the effect of normalisation. Use different colours and zoom in to see the details.

Now write the speech out as a wav file:

```
import soundfile as sf
sf.write('speech.wav', r, fs)
```

Wav files can be read into Python using the `soundfile` library:

```
r_in, fs_in = sf.read('speech.wav', dtype='float32')
```

Now, compare `r` and `fs` with `r_in` and `fs_in` to make sure they're the same.

3 Automated speech capture in Python

For the speech recognition coursework you will need to record and save many speech files. A Python loop can be used to record and save a set of files. A very simple structure would take the form:

```
for fileNumber in range(0, 10):
     # prompt speaker to say utterance
     # record audio
     # normalise audio
     # save audio to next wav file
```

But this is too simple! We have made a template program available on Blackboard, `VideoRecorder.py`, which records audio and video simultaneously. You are free to use and adapt this code for use in your own coursework. Take a look at the code in the template, think about out how it works and how you might adapt it. Note that to run it you should create a temporary directory on the local disk (e.g. c:\temp), copy the script into this location and run it from that location.

Jake Newman – September 2023