LING 402 – Tools and Techniques for Speech and Language Processing

Autumn 2019 Final Project

Instructor: *Yan Tang*
Teaching Assistant: *Chase Adams*
The Department of Linguistics, UIUC

December 09, 2019

1 Project summary

This is the final project for LING 402 Tools and Techniques for Speech and Language Processing. It aims to combine most of topics that have been covered in this course, and to give an opportunity to apply what you have learnt to solve problems in a simulated practical situation. Therefore, it is scenario-based and consists of several tasks. Each task may or may not depend on the output(s) of other tasks. The programming languages that you can use in this project are limited to **Bash** and **Python**. Apart from specific built-in libraries and those provided by the instructors (see below), use of any third-party libraries or code in the submission without approval is strictly restricted, and will result in a significant reduction in your grade. The weight distribution of the tasks is provided in Sec. 4 and Sec. 6. Read this document carefully; it is your responsibility to understand and implement all the requirements on the tasks. The instructors are not obliged to respond or answer any questions whose answers have already been clearly documented in this document and in the code templates supplied along with this document.

Performance on this project makes up **20%** of your final grade in this course. The project is due at **23:59:59 Wednesday, December 18 2019**. The late work and penalty-free late policies regulated in the syllabus of this course will not be applied to this project. Therefore, any form of late submission will lead to a loss of the entire 20% from the final grade.

The instructors' office hours remain in the week commencing December 8. In order to provide necessary support, the instructors will accept appointments for meetings between December 16 and 18. However, this is subject to the availability of the instructors. You are therefore advised to collate all your issues/concerns/questions together, and discuss with the instructors in same meeting.

2 Contents in the project repository

In your project repository, the contents are organised as follows:

```
/
├── LING402_FP.pdf (the current file)
└── project
    └── lib
        ├── DSP_Tools.py
        ├── Plot_Tools.py
        ├── SampleSine.py
        ├── Waveform.py
        └── WAVReader.py
```

```
    WAVWriter.py
  resources
    sentences.txt
    wn.wav
  trans
    hvd_001.trans
    ...
    hvd_100.trans
  wavs
    hvd_001.wav
    ...
    hvd_100.wav
  FPa_prepTrans.sh
  FPb_buildDict.py
  FPc_WhiteNoise.py
  FPd_genWhiteNoise.py
  FPe_genStimuli.py
  FPf_run.sh
```

Besides this document `LING402_FP.pdf`, all the resources provided for you to complete this project are in the "`project`" directory:

- `lib`: a directory containing LING402 Python modules you need.
- `resources`: a directory containing two reference outputs from Sec. 4.1 (`sentences.txt`) and 4.5 (`wn.wav`). These two files are provided to aid you in completing the subsequent tasks which depend on these files, and you are not able to produce them by yourself. Also read Sec. 4.6.
- `trans`: a directory where transcription files for 100 speech utterances are located.
- `wavs`: a directory where WAV files for 100 speech utterances are located.
- `FPa_prepTrans.sh`: a **Bash** template for the task in Sec. 4.1.
- `FPb_buildDict.py`: a **Python** template for the task in Sec. 4.2.
- `FPc_WhiteNoise.py`: a **Python** template for the task in Sec. 4.3.
- `FPd_genWhiteNoise.py`: a **Python** template for the task in Sec. 4.4.

- `FPe_genStimuli.py`: a **Python** template for the task in Sec. 4.5.
- `FPf_run.sh`: a **Bash** template for the task in Sec. 4.6.

3 Project scenario

Conducting a perceptual listening experiment for speech and hearing research involves a number of steps. The early-stage preparations are especially important because the quality of the work consequently determines the accuracy of the outputs from the experiment. Preparation work usually includes corpus recording, transcription and annotation, analysing on corpus, stimulus generation and so on, which usually can be rather tedious and onerous if the work has to be done manually. In this project, you are going to help with some of the preparation work by automating the process using computers programs.

This project mainly focuses on text processing for transcription, generation of a white noise signal and stimuli, which can be used in an experiment for speech perception in noise. You will be working on the following materials:

- WAV files for 100 speech utterances (`project/wavs`)[1]
- Raw transcriptions for the WAV files, output by a forced aligner (using automatic speech recognition) (`project/trans`)

4 Tasks

4.1 Processing the raw transcriptions: `FPa_prepTrans.sh` (25%)

Write a **Bash** script, `FPa_prepTrans.sh`, to parse all the transcription files (`project/trans/*.trans`), in order to generate a file "`sentences.txt`" and a file "`words.txt`":

- `sentences.txt`: this single file contains the WAV file names and the corresponding word-level transcriptions of all the 100 sentences. Each line is for one sentence; WAV file name and transcription are separated by a single "|" character without any space. Words in transcriptions are separated by a single space. See below for examples:

[1] This is the relative path in the project directory

```
hvd_043|sickness kept him home the third week
hvd_044|the wide road shimmered in the hot sun
hvd_045|the lazy cow lay in the cool grass
hvd_046|lift the square stone over the fence
hvd_047|the rope will bind the seven books at once
hvd_048|hop over the fence and plunge in
hvd_049|the friendly gang left the drug store
hvd_050|mesh wire keeps chicks inside
hvd_051|the frosty air passed through the coat
hvd_052|the crooked maze failed to fool the mouse
hvd_053|adding fast leads to wrong sums
hvd_054|the show was a flop from the very start
hvd_055|a saw is a tool used for making boards
hvd_056|the wagon moved on well oiled wheels
hvd_057|march the soldiers past the next hill
```

Figure 1: Snippet of `sentence.txt`

- `words.txt`: this single file lists the ARPABET[2] phoneme-level transcriptions for all the words existing in the corpus. Similar to `sentences.txt`, each word and its transcription are separated by a single "|" character without any space. Phonemes are separated by a single space, as shown in Fig. 2. All the entries or lines in `words.txt` must be *unique*. However, a same word may have more than one possible transcription, in which case all the instances should be included as exemplified in Fig. 2. The lines in this file must be sorted in a lexicographic order.

`FPa_prepTrans.sh` takes 3 positional parameters as arguments:

```
FPa_prepTrans.sh ARGV1 ARGV2 ARGV3
```

where `ARGV1`, `ARGV2` are `ARGV3` are the directory of .trans files, filename of word-level transcriptions, and filename of phone-level transcriptions, respectively. Both `sentences.txt` and `words.txt` must be saved automatically under directory "`project/etc`".

4.2 Building a dictionary for words: `FPb_buildDict.py` (15%)

Produce a **Python** script, `FPb_buildDict.py`, to build a dictionary for this corpus using "`sentences.txt`" generated in Sec. 4.1. In the dictionary, keys and values are the words and their occurrences, respectively. The entries – sorted by occurrence in a descending order – must be saved to a text file "`dict.txt`". For each entry the

[2] https://en.wikipedia.org/wiki/ARPABET

```
along|ax l ao ng
always|ao l w ey z
and|ae n d
and|ax n d
apart|ax p aa t
are|aa
around|ax r aw n d
attacked|ax t ae k t
at|ax t
a|ax
background|b ae k g r aw n d
back|b ae k
badly|b ae d l iy
```

Figure 2: Snippet of `words.txt`

word and its occurrence are separated by a single tab "\t" character, as shown in Fig. 3

```
in  18
of  16
and 15
was 14
is  9
on  8
from    4
for 4
cut 4
blue    3
makes   3
were    3
there   3
```

Figure 3: Snippet of `dict.txt`

`FPb_builtDict.py` takes 2 positional parameters as arguments:

```
FPb_builtDict.py ARGV1 ARGV2
```

where `ARGV1` is the path to "`sentences.txt`", and `ARGV2` is the filename of the dictionary. `dict.txt` must be saved under directory "`project/etc`".

4.3 A Python class for white noise: `FPc_WhiteNoise.py` (10%)

White noise[3] is a broadband noise, whose energy almost equally distributes across its entire frequency range. You need to generate a white noise signal and save it as a WAV file so that it later can be used as the noise masker against the speech signal in the stimuli for experiment.

The first thing to do is to create a **Python** class "`WhiteNoise`" in `FPc_WhiteNoise.py`, taking "`project/lib/SampleSine.py`" as the example. See the template in the `project` folder for more details and requirements.

4.4 Generating the WAV file of white noise: `FPd_genWhiteNoise.py` (20%)

Write a **Python** script, `FPd_genWhiteNoise.py`, to generate a 1-minute long white noise signal using a sampling frequency of 16000 Hz and amplitude of 0.5, and save it as a WAV file.

`FPd_genWhiteNoise.py` takes 3 positional parameters as arguments:

```
FPd_genWhiteNoise.py ARGV1 ARGV2 ARGV3
```

where `ARGV1` is the duration of the signal in seconds, `ARGV2` is the sampling frequency/rate in Hz, and `ARGV3` is the filename of the WAV file, at where the signal should be saved.

You must use the `WhiteNoise` class created in Sec. 4.3 to generate the signal. The procedure is very much similar to how you generated sinusoids using `SampleSine` in your previous homework. The desired WAV file must be saved as "`wn.wav`" under "`project/noise`", using a quantisation resolution of 16 bits.

In order to check if the signal has the correct frequency response, you also need to inspect its spectrum from a small stretch of the one-minute signal. Use the excerpt from the first second of the signal for the analysis. Perform the Fast Fourier Transform on the entire segment; the magnitude should be converted to decibels before plotting. Finally, save the spectrum as "`spect_wn.pdf`" under "`project/etc`". If the white noise signal is created correctly, its spectrum should look similar to that displayed in Fig. 4. The spectrum of your signal does not necessarily look exactly the same to the example, because the process of generating white noise involves random sampling. You just need to check whether the envelope of the spectrum (the spectral profile) tends to be "flat", since as mentioned above all the frequencies

[3] https://en.wikipedia.org/wiki/White_noise

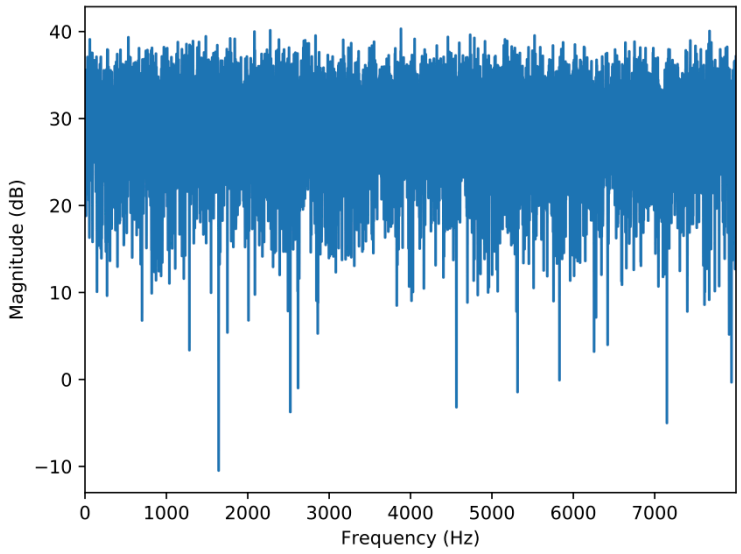

Figure 4: Example of `spect_wn.pdf`

should possess a similar amount of energy. See the template in the `project` folder for more details.

4.5 Generating speech-noise pairs as experiment stimuli: `FPe_genStimuli.py` (30%)

Once the noise WAV file is ready, the stimuli for experiment can be generated from the given speech materials ("`project/wavs/*.wav`") and the noise WAV file ("`project/noise/wn.wav`"). For a speech-in-noise test, listeners normally listen to speech sentences in the presence of background noise. The levels of both the speech and noise signals are strictly controlled in order to elicit listener responses at different noise levels, with the relative level between the speech and noise signals being particularly important. This relative level can be measured as *speech-to-noise ratio* (SNR) in decibels (dB). Mathematically SNR can be expressed as,

$$SNR = 10 * \log_{10}\left(\frac{\sum_{t=1}^{T} s(t)^2}{\sum_{t=1}^{T} n(t)^2}\right) \tag{1}$$

where s and n are the speech and noise signals, respectively. T denotes the total number of the sample points.

In this task, you are going to produce a **Python** script, `FPe_genStimuli.py`, in order to make 100 speech-noise pairs as stimuli for the experiment. While speech WAV files are given, the noise signal must be *randomly* extracted from "`wn.wav`" created in Sec. 4.4 for each of the 100 speech WAV files. In each speech-noise pair, the duration of the noise signal must match that of its counterpart speech signal. One way to do this is to randomly decide a position in `wn.wav` as the starting index for the noise signal first. Since the length of the speech signal, T, is known, then the position where the noise signal should end in `wn.wav` can be calculated. The noise signal matching T can be subsequently acquired using array/list slicing. It is however worth noting that the randomly-determined position in `wn.wav` has to be no greater than $(L_{wn} - T)$, where L_{wn} is the length of '`wn.wav`", otherwise the noise signal extracted from that position will *not* be long enough to match the speech signal.

For each speech-noise pair, there should be a random starting index in `wn.wav` for the noise signal as aforementioned. You are required to save all the indices as integers along with the file names of the speech signals to a file "`wn_idx.txt`" under "`project/etc`", in which the file name and the corresponding index are separated by a single tab "`\t`" character. See Fig. 5 for example. Note that you should

```
hvd_042.wav 175403
hvd_043.wav 727588
hvd_044.wav 548104
hvd_045.wav 228529
hvd_046.wav 464542
hvd_047.wav 561564
hvd_048.wav 238144
hvd_049.wav 145656
hvd_050.wav 22731
hvd_051.wav 408786
hvd_052.wav 240377
hvd_053.wav 248032
```

Figure 5: Snippet in an example of `wn_idx.txt`

not check your outputs against the example, because all the numbers are generated stochastically. `wn_idx.txt` will be used during grading to regenerate noise segments, in order to compare to those supplied by you.

As per the requirement on the signal levels, all speech-noise pairs must have a target SNR, $SNR_{tar} = 0$ dB. That is, the energy of the speech signal is equal to that of the noise. The procedure of adjusting the SNR is similar to setting the

theoretical sound pressure level of a signal to a target value in Homework 13. In practice, altering SNR can be achieved by either changing the speech level while maintaining the noise level, or the opposite. In this task, you must implement this by keeping the *speech level* unchanged while altering the noise level. This can be done by scaling the noise signal by a factor k,

$$SNR_{tar} = 10 * \log_{10}\left(\frac{\sum_{t=1}^{T} s(t)^2}{k^2 \cdot \sum_{t=1}^{T} n(t)^2}\right) \tag{2}$$

The method to compute SNR, `snr()`, is provided in `DSP_tools.py` in "`project/lib`". You are expected to define a method, `setSNR()`, at the beginning of `FPe_genStimuli.py`, and call your method when needed. Your `setSNR()` should take three arguments: (1) the speech signal, (2) the noise signal, and (3) the desired SNR; it then returns variables: (1) the level-adjusted noise signal, leading to the target SNR, and (2) the scaling factor k.

Once the noise signal is ready, save the speech-noise pairs as a stereo WAV file. This is no different to how you wrote WAV files in your homework, apart from that the speech signal and the noise signal need to be combined as a $T \times 2$ 2-D matrix as the data taken by `WAVWriter`. In the saved WAV file, the speech signal should always be on the left channel, i.e. it takes the very first column in the 2-D matrix mentioned above. All the WAV files containing speech-noise pairs must be named using the names of their corresponding speech WAV files, and are saved under "`project/stimuli`".

`FPe_genStimuli` takes 4 positional parameters as arguments:

```
FPe_genStimuli ARGV1 ARGV2 ARGV3 ARGV4
```

where `ARGV1` is the path to the original speech WAV files, `ARGV2` is the path where the stimuli WAV files are going to be saved, `ARGV3` is the path to "`wn.wav`", and `ARGV4` is "`etc/wn_idx.txt`"

4.6 BONUS: Running all the scripts in a batch: `FPf_run.sh` (10%)

If you have successfully completed all the tasks above, each script, except for `FPc_WhiteNoise.py` (Sec. 4.3), should be independently executable provided that the required inputs are available. In this bonus task, write another **Bash** script, `FPf_run.sh`, to perform all the processing in a batch. `FPf_run.sh` must include executions of the following scripts:

- `FPa_prepTrans.sh` (Sec. 4.1)

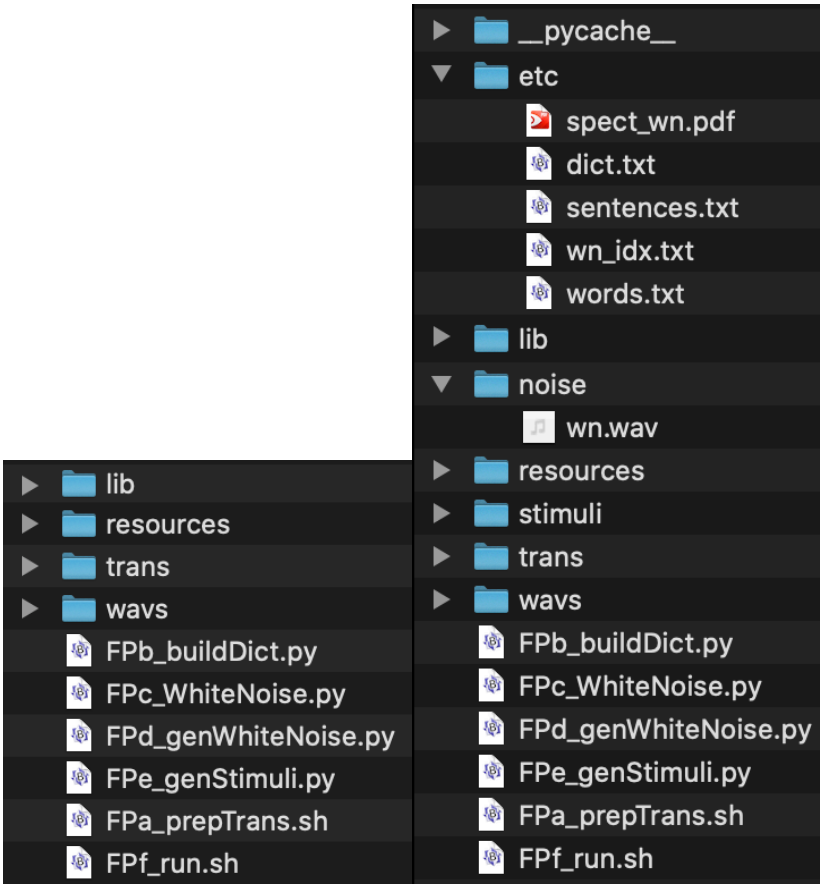

Figure 6: Contents in "`project`" before and after running `FPf_run.sh`

- `FPb_buildDict.py` (Sec. 4.2)
- `FPd_genWhiteNoise.py` (Sec. 4.4)
- `FPe_genStimuli.py` (Sec. 4.5)

All the extra directories required by any other scripts, including "`project/etc`", '`project/noise`" and "`project/stimuli`", must be created in `FPf_run.sh` using Bash commands. Your code should check the existence of each directory first. If the target directory exists, remove the directory along with its contents, then re-create it. When calling the four external scripts, the positional parameters of each script must be supplied using Bash variables. `FPf_run.sh` should also print out timely messages to stdout, prompting the progress of the processing. Fig. 6 shows the folder structure within "`project`" before and after executing `FPf_run.sh`.

A part of expected outputs, `sentences.txt`, from running `FPa_prepTrans.sh` and `wn.wav` from running `FPd_genWhiteNoise.py` are provided in "`project/resources`", in order to maintain the coherence of this project. This means that even if you are not able to complete the task in Sec. 4.1, you can move on to the subsequent tasks. Likewise, if you are having trouble generating `wn.wav`, you can skip it and continue the remaining tasks. Similarly, instead of calling the nonfunctional scripts in `FPf_run.sh`, you can use Bash commands to create the three required directories and copy `sentences.txt` and `wn.wav` into the appropriate locations to keep the workflow going. However, your `FPf_run.sh` must print out messages to stdout when the execution of any of the above scripts is skipped. By doing so, you will still receive bonus points.

5 Submission

The Github Classroom is set to shut down the repositories for this project *one second* after the due time **23:59:59 Wednesday, December 18 2019**. This means any submission attempts after that time will be rejected. Therefore, do make sure you push back all your work before the deadline.

Besides what were originally in your repository while checking out, you must push back all the directories and files generated by executing your scripts upon submission. See the graph on the right in Fig. 6 for details. The instructors may or may not run your code again while grading. Therefore, any missing files from your submission will be assumed to be due to the scripts, which are responsible for producing the files, not being functional.

6 Grading

A summary of weight distributions of all the tasks in the project:

- **25%**: `FPa_prepTrans.sh`
- **15%**: `FPb_buildDict.py`:
- **10%**: `FPc_WhiteNoise.py`:
- **20%**: `FPd_genWhiteNoise.py`:
- **30%**: `FPe_genStimuli.py`:
- **Extra credit: 10%**: `FPf_run.sh`:

The first five main tasks from `FPa` to `FPe` make up 100%, with `FPf` being the extra 10%. The extra points are additive to your overall points for the main tasks. However, the total points for this project cannot go over 100%. Namely, if the sum of points for the main and bonus tasks is greater than 100, 100 is your final score for this project.