ECE 5110 Lab Report

NAME: Haotian Liu   NETID: hl2584   DATE:11/02/2025

Acknowledgment Statement: Some of the *code* was generated by GPT and is acknowledged in the comments. This *report* is written **WITHOUT ANY GENERATIVE AI to provide texts**. I used the AI tools to refine my expressions.

1. Introduction

Reducing noise in sound signals is playing an increasingly important role in our daily lives, with applications ranging from commercial telecommunications to space and military uses.

In theory, a Butterworth filter, designed using `NumPy` and `SciPy` in a Python 3 environment, can be applied to denoise voice signals. My objective is to design, implement, and evaluate a low-pass Butterworth filter for audio signal denoising in Python 3 on a Linux (specifically the Debian) operating system.

2. Methodology and Results

2.1 Initialization

To set up a Python3 coding environment on a Linux OS, here are several steps to initialize: First, open a new terminal, and run the following commands to install the required components:

```
sudo apt update

sudo apt install python3-pip python3-setuptools python3-wheel
python3-numpy python3-scipy -y
```

Then, run these commands in order to ensure the installed components are added to the correct PATH:

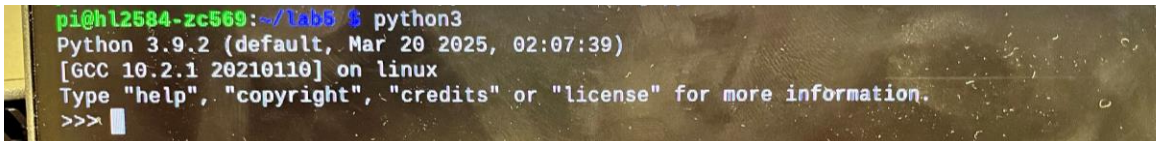
```
echo ‘export PATH=$PATH:~/.local/bin’ >> ~/.bashrc

source ~/.bashrc
```

After that, it is advised to run the following commands to check whether the Python3, `NumPy` and `SciPy` can be found in the environment:

```
python3
```

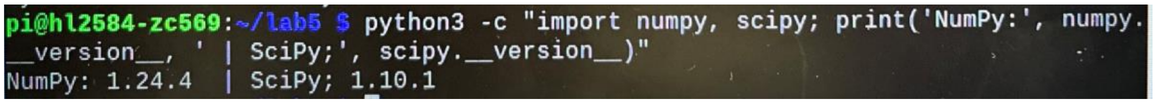
```
python3 -c "import numpy, scipy; print('NumPy:',
numpy.__version__, ' | SciPy:', scipy.__version__)"
```

The results should be the current version of them (as shown in Figure 2-1-1, and Figure 2-1-2):



```
pi@hl2584-zc569:~/lab5 $ python3
Python 3.9.2 (default, Mar 20 2025, 02:07:39)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Figure 2-1-1



```
pi@hl2584-zc569:~/lab5 $ python3 -c "import numpy, scipy; print('NumPy:', numpy.
__version__, ' | SciPy;', scipy.__version__)"
NumPy: 1.24.4   | SciPy; 1.10.1
```

Figure 2-1-2

Finally, create a directory for the lab, import the original voice file `earhart.wav`, and create a python file named `denoising.py`. The components (`NumPy` and `SciPy`) should be imported at the start of the python code:

```
import scipy.io.wavfile as wav

# from scipy.io import wavfile

import scipy.signal as signal

import scipy.fft as fft

import numpy as np
```

2.2 Data Acquisition

At the beginning, load the original audio file into Python3 from the same directory as the python file:

```
# (a) Read Input Audio

F, X = wav.read('earhart.wav')
```

in which `F` will contain the sampling rate, and `X` will contain the audio samples.

2.3 Preprocessing

For the preprocessing part, the `earhart.wav` file is encoded using 16-bit signed integers, so the elements of X will be of this type.

When processing the samples, it can be helpful to normalize them to floating-point numbers between −1 and 1 ([-1.0, 1.0]):

```
# (b) Normalize to [-1.0, 1.0] Float Range

amp16 = np.iinfo(np.int16).max

X     = X/amp16
```

2.4 Butterworth Low-Pass Filter Design

Now it is time to design the low-pass filter. The Butterworth Low-Pass Filter can be applied by `scipy.signal.butter`:

```
# (c) Design a LPF (Butterworth Type)

# AI Generated Code Starts Here

cutoff= 1000

b, a  = signal.butter(6, cutoff / (0.5 * F), btype='low')

# AI Generated Code Ends Her
```

There are several crucial attributes that need to be addressed for this filter:

- `cutoff` means the cutoff frequency (F=1000 Hz);
- 6 means the filter order (N=6);
- the `a` and `b` mean the denominator (`a`) and numerator (`b`), which are coefficients of the filter's transfer function

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}.$$

By `0.5 * F`, we have the Nyquist Frequency. Then we apply `cutoff / (0.5 * F)` to acquire the normalized cutoff frequency to make the filter works on a scale of 0 (DC) to 1 (Nyquist).

By combining `a`, `b`, and the audio sample (input) `X`, we can move on to the next step: filtering.

2.5 Implementation

Since an infinite impulse response (IIR) filter was designed, it introduced phase delay, which should be eliminated. By utilizing the `signal.filtfilt()` function, it can perform forward-backward filtering, which in turn, adjusts the total phase shift to 0.

```
# (d) Apply 0-Phase Filtering

# AI Generated Code Starts Here

filtered = signal.filtfilt(b, a, X)

# AI Generated Code Ends Here
```

2.6 Output Generation

Since the form was converted to float in the previous steps, it is necessary to convert it back to `int16`, and pass both the sample rate `F` and the integer array to the estimate `out.wav`.

```
# (e) Write Filtered Output (convert back to int16)

wav.write('out.wav', F, np.int16(filtered * amp16))
```

To run the code, use this command in the terminal:

```
sudo python3 denoising.py
```

make sure to check whether we are in the correct directory.

The full version of the `denoising.py` can be found in the code appendix and as a file in the attachments**.**

3. Conclusion

During this lab, I de-noised a voice signal and generated an estimate of the original signal, which contains the same samples as the original one. I utilized the Linux operating system, which can be effective with a Python3 environment and is capable of computing all the attributes with a MCU. The lower power efficiency compared to Windows OS makes it ideal to be implemented as an embedded system, which can be useful in a variety of fields.

That said, not everything went smoothly. The Linux OS uses quite a different approach – the terminal – to set up everything about the Python3 environment. I encountered some problems during the package installation stage, failing to install some of the packages due to an unstable network connections. Additionally, I ran the python program and found that numpy and scipy were not imported. That was weird because it did not occur on a Windows OS. Then I did more search and called ChatGPT for help. It turned out that the installed components were not added to the correct PATH, which highlights the importance of checking everything before implementing.

For future iterations, there are plenty of improvements available to my work. Firstly, the data structure (float) might be not capable of converting the signal as a array variable when the sound signal is much larger than this earhart.wav. Then we might need to replace it with a data structure that handles more bits (double for example). Besides, there's no need to compute the PSD of the signal in digital Butterworth filtering. But when it comes to optimal filters like the Wiener filter, it is necessary to compute the power spectral density (PSD) of the signal and store it in the variable `Xpsd` using `fft.fft()` algorithm provided by `scipy.fft`:

```
Xspec = fft.fft(X)
```

This will compute the Discrete Fourier Transform (DFT) of `X`, which can be viewed as the DTFT evaluated at discrete frequencies. `Xspec` represents the spectrum with the frequency variable varying from 0 to 1 cycle per sample, as opposed to −1/2 to 1/2.

According to the lab guide, the PSD can be calculated using this code:

```
Xpsd  = np.abs(Xspec)**2 /len(Xspec)
```

References

Lab Guide:

filtering.pdf
https://canvas.cornell.edu/courses/80275/files/13981018?module_item_id=3411990

Textbook：

Leon-Garcia, Alberto. *Probability, Statistics, and Random Processes for Electrical Engineering*. Third edition, Pearson/Prentice Hall, 2008.

Textbooks and notes from my undergraduate Signals and Systems course (forgive me about the citation in Chinese, such as, the 郑君里. 信号与系统（第 3 版） [M]. 高等教育出版社, 2011.)

Generative AI

Code Appendix

```
import scipy.io.wavfile as wav

# from scipy.io import wavfile

import scipy.signal as signal

import scipy.fft as fft

import numpy as np


# (a) Read Input Audio

F, X = wav.read('earhart.wav')


# (b) Normalize to [-1.0, 1.0] Float Range

amp16 = np.iinfo(np.int16).max

X     = X/amp16


# (c) Design a LPF (Butterworth Type)

# AI Generated Code Starts Here

cutoff= 1000

b, a  = signal.butter(6, cutoff / (0.5 * F), btype='low')

# AI Generated Code Ends Here


# (d) Apply 0-Phase Filtering

# AI Generated Code Starts Here

filtered = signal.filtfilt(b, a, X)

# AI Generated Code Ends Here
# (e) Write Filtered Output (convert back to int16)

wav.write('out.wav', F, np.int16(filtered * amp16))
```