Problem Set 1

Jungwoong Yoon (jy8963)

February 13, 2023

Exercise 1

1.1

1/4 WL equation: $f_n = \frac{c}{4l}(2n-1)$

$$f_1 = \frac{34,000}{4*9}(2-1) \approx 944 \text{ Hz} \quad f_1 = \frac{34,000}{4*7}(2-1) \approx 1,214 \text{ Hz} \tag{1}$$

$$f_2 = \frac{34,000}{4*9}(4-1) \approx 2,833 \text{ Hz} \quad f_2 = \frac{34,000}{4*7}(4-1) \approx 3,642 \text{ Hz} \tag{2}$$

$F_1 = 944Hz$, $F_2 = 1,214Hz$, $F_3 = 2,833$ Hz
Vowel [ɑ] would be a good match for this vocal tract configuration.

1.2

Helmholtz:

$$f = \frac{c}{2\pi}(\frac{A_2}{A_1 l_1 l_2})^{\frac{1}{2}} = \frac{34,000}{2\pi}(\frac{1}{9*9*5})^{\frac{1}{2}} \approx 269 \text{ Hz}$$

1/2 WL:

$$f_1 = \frac{c}{2l_1}(n) = \frac{34,000}{2*9}(1) \approx 1,889 \text{ Hz}$$

1/2 WL:

$$f_1 = \frac{c}{2l_2}(n) = \frac{34,000}{2*5}(1) = 3,400 \text{ Hz}$$

$F_1 = 269Hz$, $F_2 = 1,889Hz$, $F_3 = 3,400$ Hz
Vowel [ɪ] would be a good match for this vocal tract configuration.

Exercise 2

2.1

1. Fricative - C
2. Stop - L
3. Semivowel - M
4. Nasal - G
5. Front Vowel - F
6. Back Vowel - B
7. Alveolar Consonant - H
8. Retroflex - K

2.2

Number of pitch periods from 0.5 to 0.6 seconds = 15

$\frac{15}{0.1} = 150Hz$

Exercise 3

3.1 Loading the audio file

```
# Loading the audio file
signal = scipy.io.wavfile.read("signal.wav")
print(signal)
x1 = signal[1]
```

```
(16000, array([ 174575,  449637,  316148, ..., -315117, -454616, -419864],
      dtype=int32))
```

1. What is the sampling rate of the audio, and how many samples are in the recording?

- Sampling rate: 16,000
- 18,091 samples

2. How many seconds worth of audio does this correspond to?

- ? second

3.2 Mean subtraction

```
# Mean subtraction
x0 = np.mean(x1)
x2 = x1 - x0
```

3.3 Pre-emphasis

```
# Pre-emphasis
x2_temp = np.append(0.0, x2[:len(x2) - 1])
x3 = x2 - 0.97 * x2_temp
```

3.4 Computing frames

```
# Computing frames
def compute_frames(x3, L, S):
    N = math.ceil((len(x3) - L) / S)
    frames = np.zeros((N + 1, L))

    for k in range(N + 1):
        for n in range(L):
            if k * S + n < len(x3):
                frames[k, n] = x3[k * S + n]


    return frames
```

3.5 Applying the window function

```
# Applying the window function
def apply_window(frame):
    w = scipy.signal.hamming(len(frame))
    return frame * w
```

3.6 Computing the Fourier transform

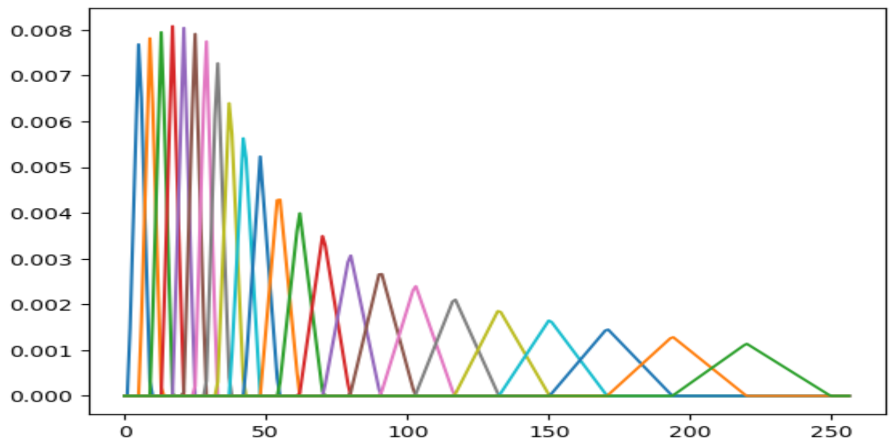
```
# Computing the Fourier transform
def dft(windowed_frame):
    return scipy.fft.fft(windowed_frame, n=512)
```

3.7 Computing the magnitude and power spectra

```
# Computing the magnitude and power spectra
def power_spectra(dft_frame):
    return np.square(np.abs(dft_frame))
```

3.8 Mel-filterbank application

```
# Mel-filterbank application
mel_filters = np.load("mel_filters.npy")
print(mel_filters.shape)

plt.plot(mel_filters.T)
plt.show()

def mel_filter(X_pow):
    return np.matmul(X_pow[:, :mel_filters.shape[1]], mel_filters.transpose())
```



3.9 Taking the log

```
# Taking the log
def logmel(X_mel):
    X_logmel = np.empty(X_mel.shape)

    for m in range(X_logmel.shape[0]):
        for k in range(X_logmel.shape[1]):
            X_logmel[m, k] = max(-50, np.log(X_mel[m, k]))

    return X_logmel
```

3.10 Computing the DCT and “liftering”

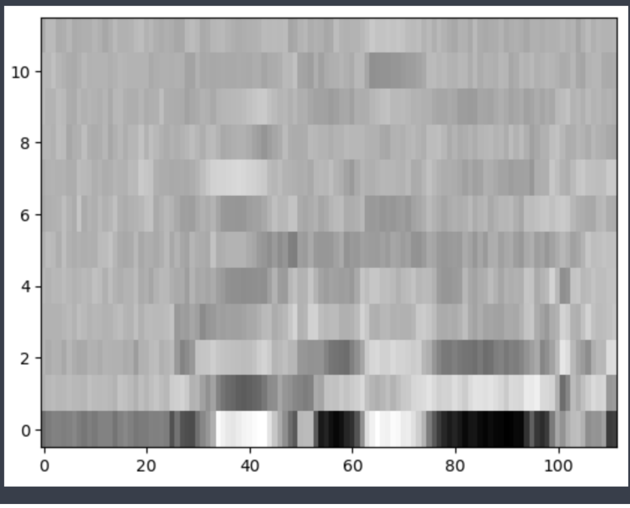
```python
# Computing the DCT and "liftering"
def dct(X_logmel):
    C = np.zeros((X_logmel.shape[0], 13))

    for m in range(C.shape[0]):
        for i in range(C.shape[1]):
            for k in range(X_logmel.shape[1]):
                C[m, i] += X_logmel[m, k] * math.cos(((math.pi * i) / X_logmel.shape[1]) * (k + 0.5))

    return C
```

3.11 Putting it all together

```python
"""
Putting it all together
"""

def get_mfcc(length_time, shift_time):
    L, S = int(signal[0] * length_time), int(signal[0] * shift_time)

    frames = compute_frames(x3, L, S)
    X_pow = np.empty((frames.shape[0], 512))

    for m in range(X_pow.shape[0]):
        windowed_frame = apply_window(frames[m])
        dft_frame = dft(windowed_frame)
        X_pow[m] = power_spectra(dft_frame)

    X_mel = mel_filter(X_pow)
    X_logmel = logmel(X_mel)

    return dct(X_logmel)
```

```python
mfcc = get_mfcc(0.025, 0.01)
plt.imshow(mfcc[:, 1:].transpose(), cmap="gray", origin="lower", aspect="auto")

mfcc_ref = np.load("reference_mfcc.npy")
mse = (np.square(np.subtract(mfcc_ref, mfcc)).mean())
print("MSE: {}".format(mse))
```

```
MSE: 2.0871469444709202e-26
```



MSE: 2.0871469444709202e-26

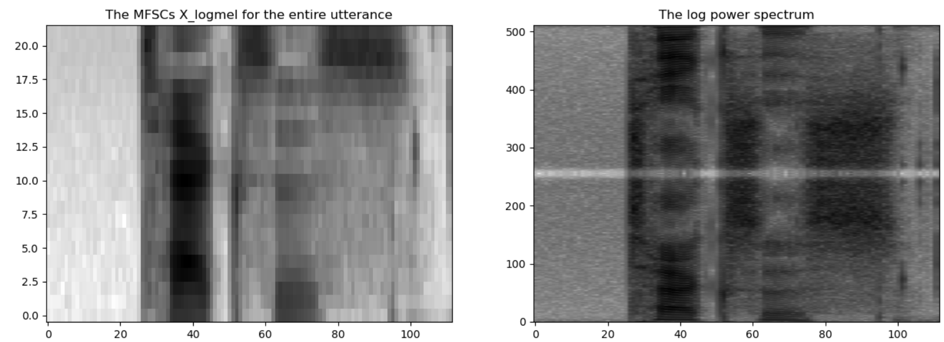


Figure 1: L = 0.025s, S = 0.01s

Two plots above show significant differences. The X_{logmel} plot (left) has less resolution than the log power spectrum plot (right). Moreover, the overall shapes of these plots are different.

However, if we take a close look at the log power spectrum plot, it is symmetric about the horizontal band, and bottom half of the plot actually has similar spectrum shape to the X_{logmel} plot. This is because the log power spectrum omits the application of the Mel-filterbanks therefore contains the entire 512 frames whereas the X_{logmel} only contains first half of it due to the application of the Mel-filterbanks.

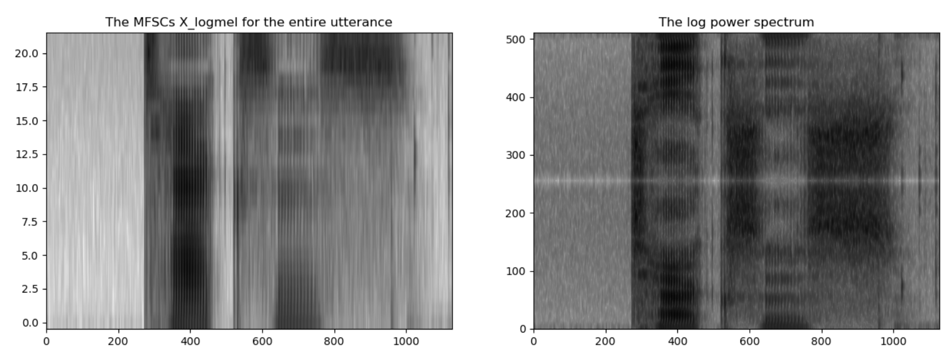


Figure 2: L = 0.004s, S = 0.001s

These plots seems to have better time resolution but worse frequency resolution than those of 25 millisecond windows due to the window tradeoff; the window length controls the tradeoff between time resolution and frequency resolution therefore smaller window length gives better time resolution, worse frequency resolution.

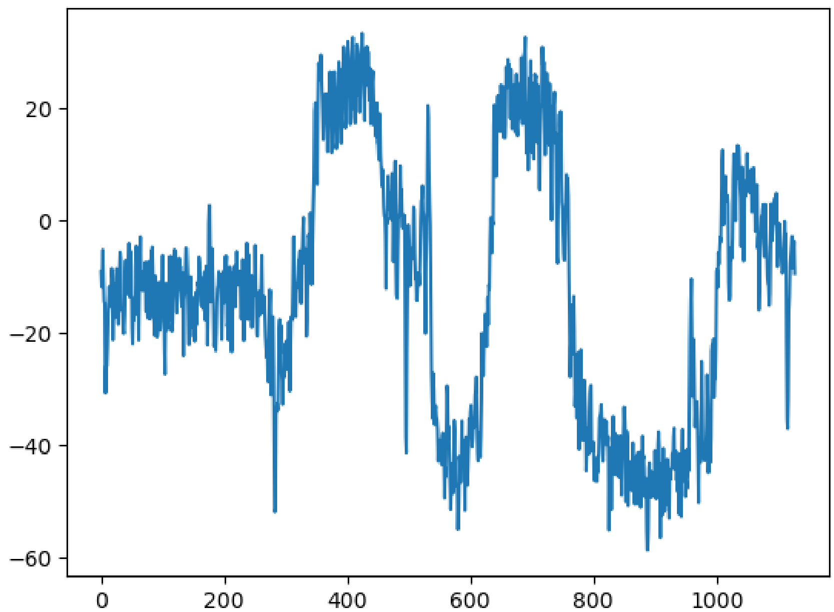


Figure 3: A plot of C_1

During time period of a vowel sound, the value of C_1 goes significantly high (around 20 or higher) whereas during time period of a fricative, the the value of C_1 becomes significantly low (-40 or lower).

This is because C_1 basically tracks the first formant on vowels to specify correlation between vowels and non-vowels.