

ECE 438 - Laboratory 9a Speech Processing (Week 1)

Last Updated on March 29, 2022

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf
import IPython.display as ipd
from helper import hamming
```

```
In [2]: # specify the size of the plot
plt.rcParams['figure.figsize'] = (16, 6)

# make sure the plot is displayed in this notebook
%matplotlib inline

# for auto-reloading external modules
%load_ext autoreload
%autoreload 2
```

1. Introduction

Speech is an acoustic waveform that conveys information from a speaker to a listener. Given the importance of this form of communication, it is no surprise that many applications of signal processing have been developed to manipulate speech signals. Almost all speech processing applications currently fall into three broad categories: speech recognition, speech synthesis, and speech coding.

Speech recognition may be concerned with the identification of certain words, or with the identification of the speaker. **Isolated word recognition** algorithms attempt to identify individual words, such as in automated telephone services. Automatic speech recognition systems attempt to recognize continuous spoken language, possibly to convert into text within a word processor. These systems often incorporate grammatical cues to increase their accuracy. Speaker identification is mostly used in security applications, as a person's voice is much like a "fingerprint".

The objective in speech synthesis is to convert a string of text, or a sequence of words, into natural-sounding speech. One example is the Speech Plus synthesizer used by Stephen Hawking (although it unfortunately gives him an American accent). There are also similar systems which read text for the blind. Speech synthesis has also been used to aid scientists in learning about the mechanisms of human speech production, and thereby in the treatment of speech-related disorders.

Speech coding is mainly concerned with exploiting certain redundancies of the speech signal, allowing it to be represented in a compressed form. Much of the research in speech compression has been motivated by the need to conserve bandwidth in communication systems. For example, speech coding is used to reduce the bit rate in digital cellular systems.

In this lab, we will describe some elementary properties of speech signals, introduce a tool known as the **short-time discrete-time Fourier Transform**, and show how it can be used to form a **spectrogram**. We will then use the spectrogram to estimate properties of speech waveforms.

This is the first part of a two-week experiment. During the second week, we will study speech models and linear predictive coding.

2. Time Domain Analysis of Speech Signals

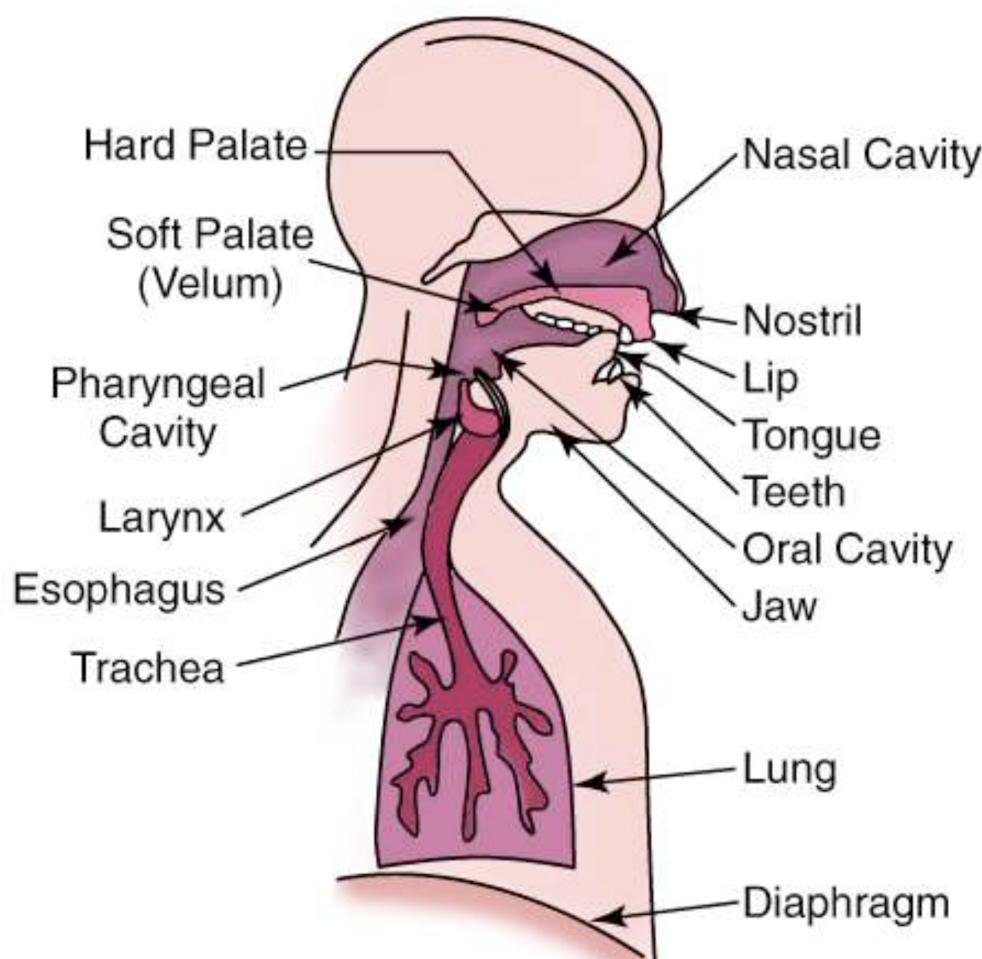


Figure 1: The Human Speech Production System

2.1. Speech Production

Speech consists of acoustic pressure waves created by the voluntary movements of anatomical structures in the human speech production system, shown in Figure 1. As the diaphragm forces air through the system, these structures are able to generate and shape a wide variety of waveforms. These waveforms can be broadly categorized into **voiced** and **unvoiced** speech.

Voiced sounds, vowels for example, are produced by forcing air through the larynx, with the tension of the vocal cords adjusted so that they vibrate in a relaxed oscillation. This produces quasi-periodic pulses of air which are acoustically filtered as they propagate through the vocal tract, and possibly through the nasal cavity. The shape of the cavities that comprise the vocal tract, known as the **area function**, determines the natural frequencies, or **formants**, which are emphasized in the speech waveform. The period of the excitation, known as the **pitch period**, is generally small with respect to the rate at which the vocal tract changes shape. Therefore, a segment of voiced speech covering several pitch periods will appear somewhat **periodic**. Average values for the pitch period are around 8 ms for male speakers, and 4 ms for female speakers.

In contrast, unvoiced speech has more of a noise-like quality. Unvoiced sounds are usually much smaller in amplitude, and oscillate much faster than voiced speech. These sounds are generally produced by turbulence, as air is forced through a constriction at some point in the vocal tract. For example, an *h* sound comes from a constriction at the vocal cords, and an *f* is generated by a constriction at the lips.

An illustrative example of voiced and unvoiced sounds contained in the word “erase” are shown in Figure 2. The original utterance is shown in (a). The voiced segment in (b) is a time magnification of the “a” portion of the word. Notice the highly periodic nature of this segment. The fundamental period of this waveform, which is about 8.5 ms here, is called the **pitch period**. The unvoiced segment in (c) comes from the “s” sound at the end of the word. This waveform is much more noise-like than the voiced segment, and is much smaller in magnitude.

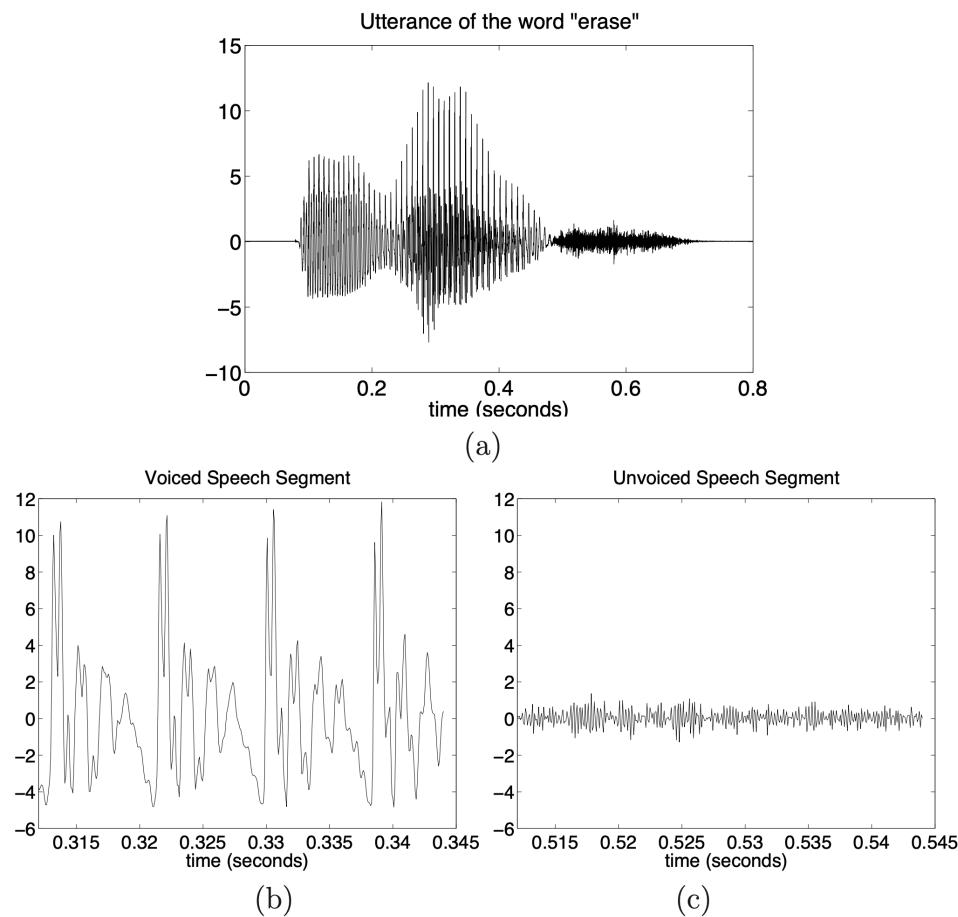


Figure 2: (a) Utterance of the work "erase". (b) Voiced segment. (c) Unvoiced segment

Exercise 2.2: Classification of Voiced/Unvoiced Speech

For many speech processing algorithms, a very important step is to determine the type of sound that is being uttered in a given time frame. In this section, we will introduce two simple methods for discriminating between voiced and unvoiced speech.

1. Load the audio `Start.wav` using `start, fs = sf.read("Start.wav")`.

```
In [3]: # insert your code here
```

2. Plot (not stem) the speech signal. Then identify two segments of the signal: one segment that is voiced and a second segment that is unvoiced. To identify them, you can choose one from the following options:

- a. Circle the regions of the plot of the speech signal corresponding to these two segments.
- b. Plot the regions corresponding to these two segments separately in new cells.
- c. Print the starting and ending indices of these two regions.

```
In [4]: # temporarily make the plot interactive  
%matplotlib notebook  
# specify the size of the plot  
plt.rcParams['figure.figsize'] = (12, 6)
```

```
In [5]: # insert your code here
```

```
In [6]: # make the plot not interactive
%matplotlib inline
# specify the size of the plot
plt.rcParams['figure.figsize'] = (16, 6)
```

2. Extract 300 samples from the voiced segment of the speech into a NumPy vector called `VoicedSig`. Also, extract 300 samples from the unvoiced segment of the speech into a NumPy vector called `UnvoicedSig`.

```
In [7]: # insert your code here
```

3. Plot the two signals, `VoicedSig` and `UnvoicedSig`.

```
In [8]: # insert your code here
```

4. Explain how you selected your voiced and unvoiced regions.

insert your answer here

5. Estimate the pitch period for the voiced segment. Keep in mind that these speech signals are sampled at 8 KHz, which means that the time between samples is 0.125 milliseconds (ms). Typical values for the pitch period are 8 ms for male speakers, and 4 ms for female speakers. Based on this, would you predict that the speaker is male, or female?

insert your answer here

One way to categorize speech segments is to compute the average energy, or power. Recall this is defined by the following:

$$P = \frac{1}{L} \sum_{n=1}^L x^2[n] \quad (1)$$

where L is the length of the frame $x[n]$.

6. Complete the function below that calculates equation (1) to compute the average energy.

```
In [9]: def get_average_energy(x):
    """
    Parameters
    ---
    x: the input signal

    Returns
    ---
    P: the average energy of the signal
    """
    P = None
    return P
```

7. Use this function to compute the average energy of the voiced and unvoiced segments that you plotted above. Print the values.

```
In [10]: # insert your code here
```

8. For which segment is the average energy greater?

insert your answer here

Another method for discriminating between voiced and unvoiced segments is to determine the rate at which the waveform oscillates by counting number of zero-crossings that occur within a frame (the number of times the signal changes sign).

9. Complete the function below to compute the number of zero-crossings that occur within a vector.

```
In [11]: def get_zero_cross(x):
    """
    Parameters
    ---
    x: the input signal

    Returns
    ---
    cnt: the number of zero-crossings
    """

    cnt = 0
    return cnt
```

10. Compute and print the numbers of zero-crossings of both `voicedSig` and `unvoicedSig`.

```
In [12]: # insert your code here
```

11. Which segment has more zero-crossings?

insert your answer here

2.3. Phonemes

American English can be described in terms of a set of about 42 distinct sounds called **phonemes**, illustrated in Figure 3. They can be classified in many ways according to their distinguishing properties. **Vowels** are formed by exciting a fixed vocal tract with quasi-periodic pulses of air. **Fricatives** are produced by forcing air through a constriction (usually towards the mouth end of the vocal tract), causing turbulent air flow. Fricatives may be voiced or unvoiced. **Plosive** sounds are created by making a complete closure, typically at the frontal vocal tract, building up pressure behind the closure and abruptly releasing it. A **diphthong** is a gliding monosyllabic sound that starts at or near the articulatory position for one vowel, and moves toward the position of another. It can be a very insightful exercise to recite the phonemes shown in Figure 3, and make a note of the movements you are making to create them.

It is worth noting at this point that classifying speech sounds as voiced/unvoiced is not equivalent to the vowel/consonant distinction. Vowels and consonants are **letters**, whereas voiced and unvoiced refer to types of speech **sounds**. There are several consonants, /m/ and /n/ for example, which when spoken are actually voiced sounds.

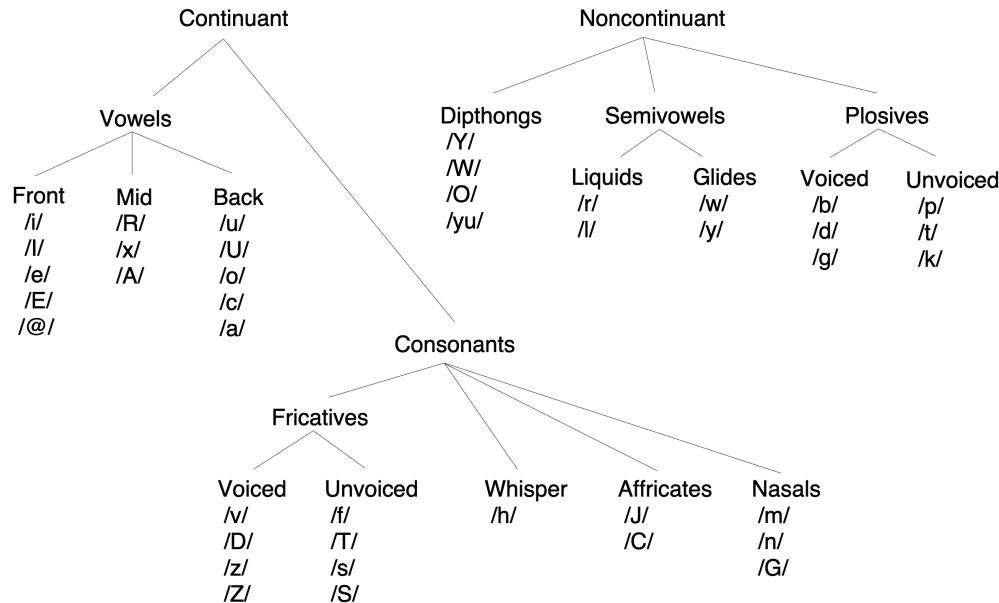


Figure 3: Phonemes in American English. See [1] for more details

3. Short-Time Frequency Analysis

As we have seen from previous sections, the properties of speech signals are continuously changing, but may be considered to be stationary within an appropriate time frame. If analysis is performed on a “segment-by-segment” basis, useful information about the construction of an utterance may be obtained. The average energy and zero-crossing rate, as previously discussed, are examples of short-time feature extraction in the time-domain. In this section, we will learn how to obtain short-time frequency information from generally non-stationary signals.

3.1. stDTFT

A useful tool for analyzing the spectral characteristics of a non-stationary signal is the **short-time discrete-time Fourier Transform**, or **stDTFT**, which we will define by the following:

$$X_m(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \quad (2)$$

Here, $x[n]$ is our speech signal, and $w[n]$ is a window of length L . Notice that if we fix m , the stDTFT is simply the DTFT of $x[n]$ multiplied by a shifted window. Therefore, $X_m(e^{j\omega})$ is a collection of DTFTs of windowed segments of $x[n]$.

As we examined in the Digital Filter Design Lab, windowing in the time domain causes an undesirable ringing in the frequency domain. This effect can be reduced by using some form of a raised cosine for the window $w[n]$.

Exercise 3.2

1. Complete the function below to compute the DFT of a windowed length L segment of the vector x .

Note:

- You should use a Hamming window of length L to window x
- Your window should start at the index m of the signal x .
- Your DFTs should be of length N
- You may use `hamming(L)` function to obtain the Hamming window.
- You may use `np.fft.fft()` function to compute the DFTs.

```
In [13]: def DFTwin(x, L, m, N):  
    """  
        Parameters  
        ---  
        x: the input signal  
        L: the length of the Hamming window  
        m: the starting index of the signal x where the Hamming window is applied  
        N: the length of DFT  
  
        Returns  
        ---  
        X: the DFT of a windowed length L segment of x  
    """  
  
    X = None  
    return X
```

2. Load the file `go.au`.

```
In [14]: # insert your code here
```

3. Plot the signal and locate a voiced region, which should cover six pitch periods.

```
In [15]: # temporarily make the plot interactive  
%matplotlib notebook  
%matplotlib notebook  
import matplotlib.pyplot as plt  
# specify the size of the plot  
plt.rcParams['figure.figsize'] = (12, 6)
```

```
In [16]: # insert your code here
```

```
In [17]: # make the plot not interactive  
%matplotlib inline  
import matplotlib.pyplot as plt  
# specify the size of the plot  
plt.rcParams['figure.figsize'] = (16, 6)
```

4. Plot the voiced region.

```
In [18]: # insert your code here
```

5. Use `DFTwin()` to compute a 512-point DFT of the speech segment, with a window that covers 6 pitch periods within the voiced region, and then plot it for $0 \leq \omega \leq \pi$.

```
In [19]: # insert your code here
```

6. Describe the general shape of the spectrum, and estimate the formant frequencies for the region of voiced speech.

insert your answer here

3.3. The Spectrogram

As previously stated, the short-time DTFT is a collection of DTFTs that differ by the position of the truncating window. This may be visualized as an image, called a **spectrogram**. A spectrogram shows how the spectral characteristics of the signal evolve with time. A spectrogram is created by placing the DTFTs vertically in an image, allocating a different column for each time segment. The convention is usually such that frequency increases from bottom to top, and time increases from left to right. The pixel value at each point in the image is proportional to the magnitude (or squared magnitude) of the spectrum at a certain frequency at some point in time. A spectrogram may also use a “pseudo-color” mapping, which uses a variety of colors to indicate the magnitude of the frequency content, as shown in Figure 4.

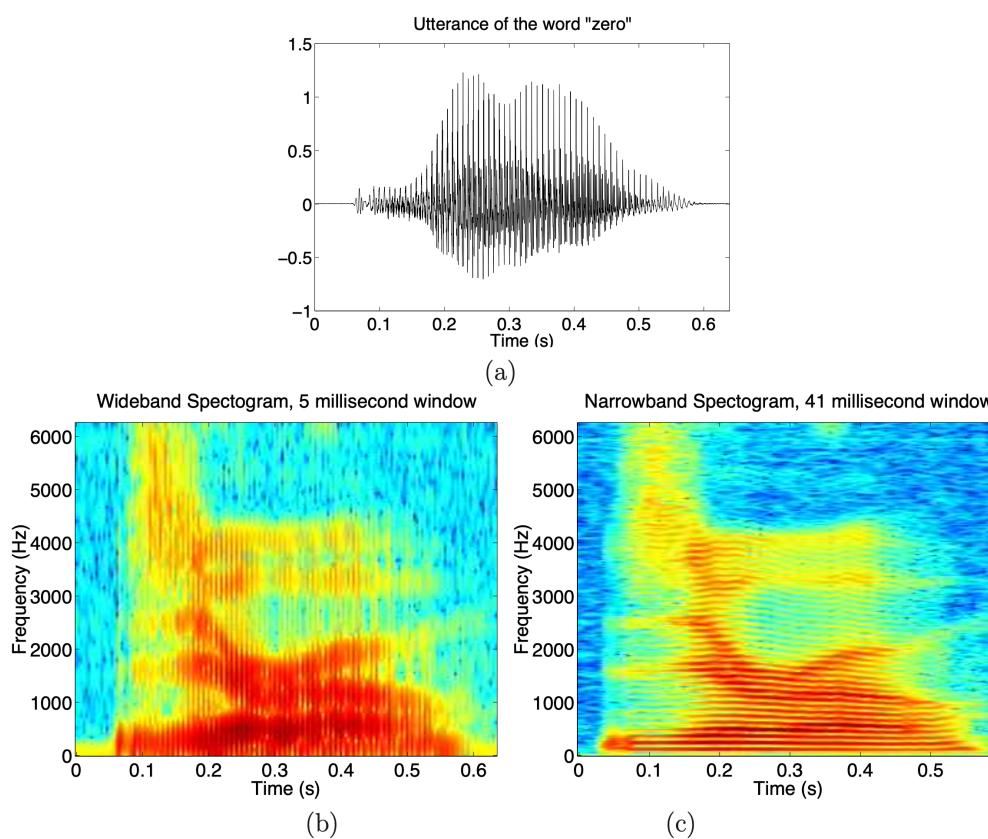


Figure 4: (a) Utterance of the word "zero". (b) Wideband Spectrogram. (c) Narrowband Spectrogram.

For quasi-periodic signals like speech, spectrograms are placed into two categories according to the length of the truncating window. **Wideband** spectrograms use a window with a length comparable to a single period. This yields high resolution in the time domain but low resolution in the frequency domain. These are usually characterized by vertical striations, which correspond to high and low energy regions within a single period of the waveform. In **narrowband** spectrograms, the window is made long enough to capture several periods of the waveform. Here, the resolution in time is sacrificed to give a higher resolution of the spectral content. Harmonics of the fundamental frequency of the signal are resolved, and can be seen as horizontal striations. Care should be taken to keep the window short enough, such that the signal properties stay relatively constant within the window.

Often when computing spectrograms, not every possible window shift position is used from the stDTFT, as this would result in mostly redundant information. Successive windows usually start many samples apart, which can be specified in terms of the **overlap** between successive windows. Criteria in deciding the amount of overlap include the length of the window, the desired resolution in time, and the rate at which the signal characteristics are changing with time.

Exercise 3.4

1. Complete the function below that can create a spectrogram using your `DFTwin()` function from the previous section. You will do this by creating a matrix of windowed DFTs, oriented as described above.

Important hints:

- You can start by initializing `A` as an empty list, then keep appending the DFT results from `DFTwin()` to it. At the end, convert it to a NumPy array by `A = np.array(A)`.
- After you do the step above, you should find that the frequency components are on the x-axis, but we want them to be on the y-axis. You might use `np.transpose()`.
- Your `DFTwin()` function returns the DT spectrum for frequencies between 0 and 2π . Therefore, you will only need to use the first or second half of these DFTs.
- The statement `B[:, n]` references the entire n th (zero-based index) column of the matrix `B`.

```
In [20]: def Specgm(x, L, overlap, N):
    """
    Parameters
    ---
    x: the input signal
    L: the window length
    overlap: the number of points common to successive windows
    N: the number of points computed in each DFT

    Returns
    ---
    A: the matrix of spectrogram
    """

    A = None
    return A
```

2. Load the file `signal.npy` using `np.load()`. Plot the signal.

```
In [21]: # insert your code here
```

3. Plot the magnitude (in dB) of the wideband spectrogram, using a window length of 50 samples and an overlap of 30 samples.

- In labeling the axes of the image, assume a sampling frequency of 8 KHz. Then the frequency will range from 0 to 4000 Hz.
- Set the parameter `extent=[0, len(signal) / 8000, 1, 4001]` in `plt.imshow()` to correctly label the axes.
- Set the parameter `aspect='auto'` in `plt.imshow()`.
- You can get a pseudo-color mapping by setting the parameter `cmap='jet'` in `plt.imshow()`.

- Set the parameter `origin='lower'` or `origin='upper'` in `plt.imshow()` to place the origin of your plot in the lower/upper left corner, depending on your matrix returned by `Specgm()`.
- For more information, see the online help for the `plt.imshow()` command.
(https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.axes.Axes.imshow.html)

In [22]: # insert your code here

4. Plot the magnitude (in dB) of the narrowband spectrogram, using a window length of 320 samples and an overlap of 60 samples.

In [23]: # insert your code here

5. Do you see vertical striations in the wideband spectrogram? Similarly, do you see horizontal striations in the narrowband spectrogram? In each case, what causes these lines, and what does the spacing between them represent?

insert your answer here

3.5. Formant Analysis

The shape of an acoustic excitation for voiced speech is similar to a triangle wave. Therefore it has many harmonics at multiples of its fundamental frequency, $1/T_p$. As the excitation propagates through the vocal tract, acoustic resonances, or standing waves, cause certain harmonics to be significantly amplified. The specific wavelengths, hence the frequencies, of the resonances are determined by the shape of the cavities that comprise the vocal tract. Different vowel sounds are distinguished by unique sets of these resonances, or formant frequencies. The first three average formants for several vowels are given in Figure 5.

Formant Frequencies for the Vowels				
Typewritten Symbol for the Vowel	Typical Word	F1 (Hz)	F2 (Hz)	F3 (Hz)
IY	(beet)	270	2290	3010
I	(bit)	390	1990	2550
E	(bet)	530	1840	2480
AE	(bat)	660	1720	2410
UH	(but)	520	1190	2390
A	(hot)	730	1090	2440
OW	(bought)	570	840	2410
U	(foot)	440	1020	2240
OO	(boot)	300	870	2240
ER	(bird)	490	1350	1690

Figure 5: Average Formant Frequencies for the Vowels. See [1].

A possible technique for speech recognition would be to determine a vowel utterance based on its unique set of formant frequencies. If we construct a graph that plots the second formant versus the first, we find that a particular vowel sound tends to lie within a certain region of the plane. Therefore, if we determine the first two formants, we can construct decision regions to estimate which vowel was spoken. The first two average formants for some common vowels are plotted in Figure 6. This diagram is known as the vowel triangle due to the general orientation of the average points.

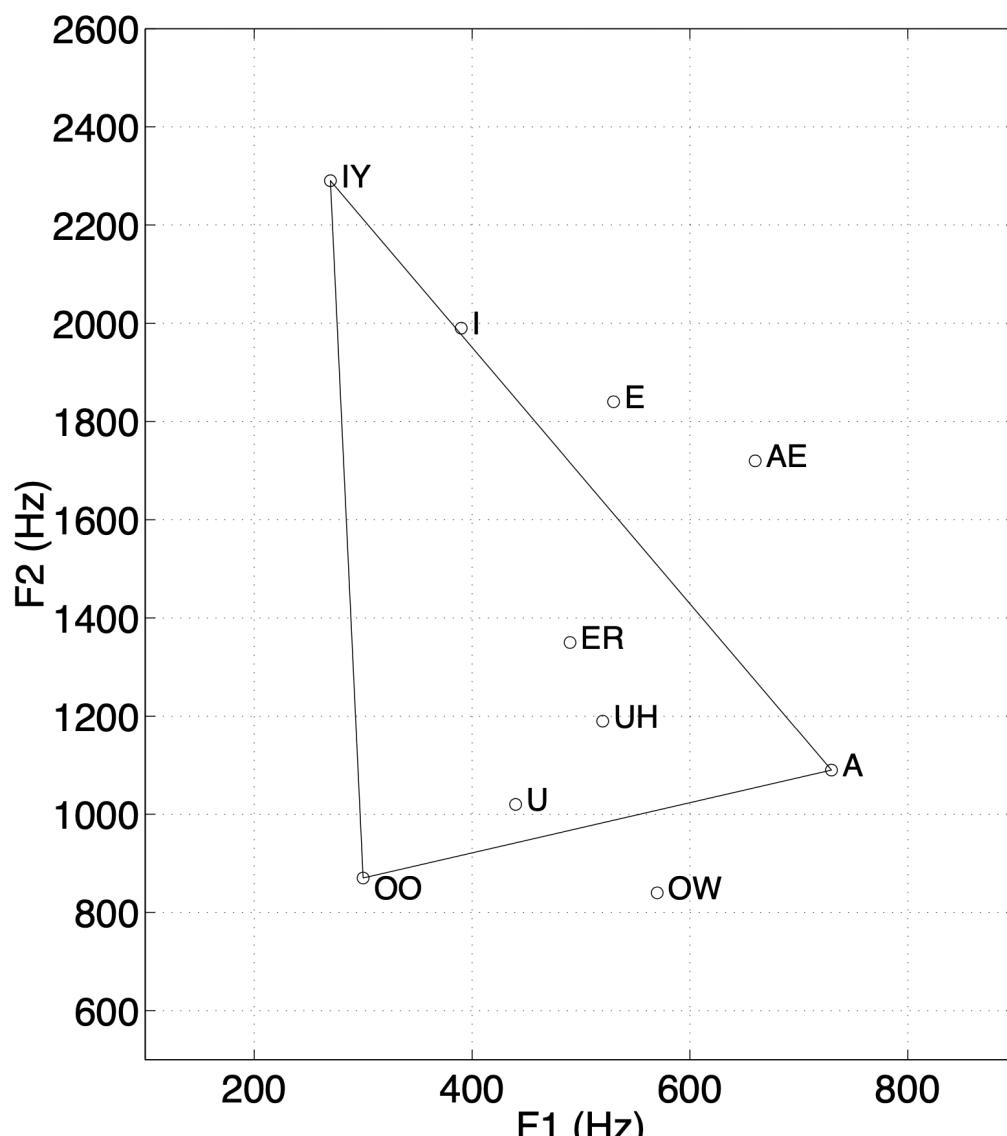


Figure 6: The Vowel Triangle.

Keep in mind that there is a continuous range of vowel sounds that can be produced by a speaker. When vowels are used in speech, their formants most often slide from one position to another.

Exercise 3.6

1. Run the following code to load the vowel utterances *a*, *e*, *i*, *o*, and *u* from a female speaker.

```
In [24]: a = np.load("a.npy")
e = np.load("e.npy")
i = np.load("i.npy")
o = np.load("o.npy")
u = np.load("u.npy")
```

2. Plot the magnitude (in dB) of the narrowband spectrogram of each of the utterances.

```
In [25]: # insert your code here
```

3. For the vowels *a* and *u*, estimate the first two formant frequencies using the functions you created in the previous sections. Make your estimates at a time frame toward the beginning of the utterance, and another set of estimates toward the end of the utterance. You may want to use both the `Specgm()` and `DFTwin()` functions to determine the formants. Plot these four points in the vowel triangle provided in the file `vowel_triangle.pdf`. For each vowel, draw a line connecting the two points, and draw an arrow indicating the direction the formants are changing as the vowel is being uttered.

```
In [27]: # insert your code here
```

remember to append `vowel_triangle.pdf` when you submit the report on Gradescope.

4. References

- [1] J. R. Deller, Jr., J. G. Proakis, J. H. Hansen, Discrete-Time Processing of Speech Signals, Macmillan, New York, 1993.