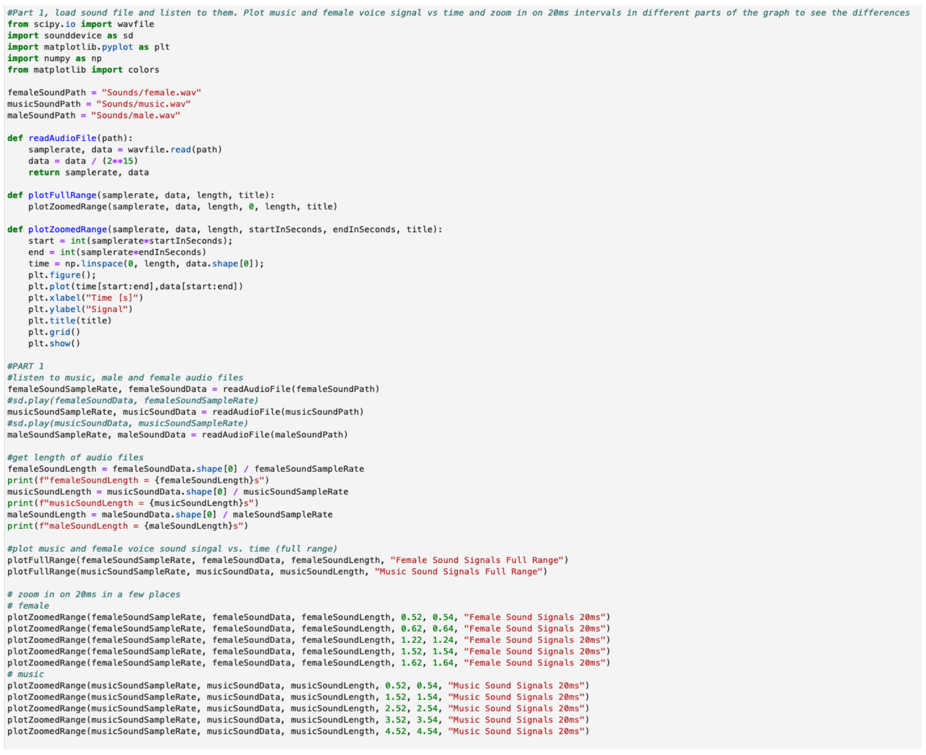
EQ2341 - Assignment2

Matay Mayrany - mayrany@kth.se

April 2022

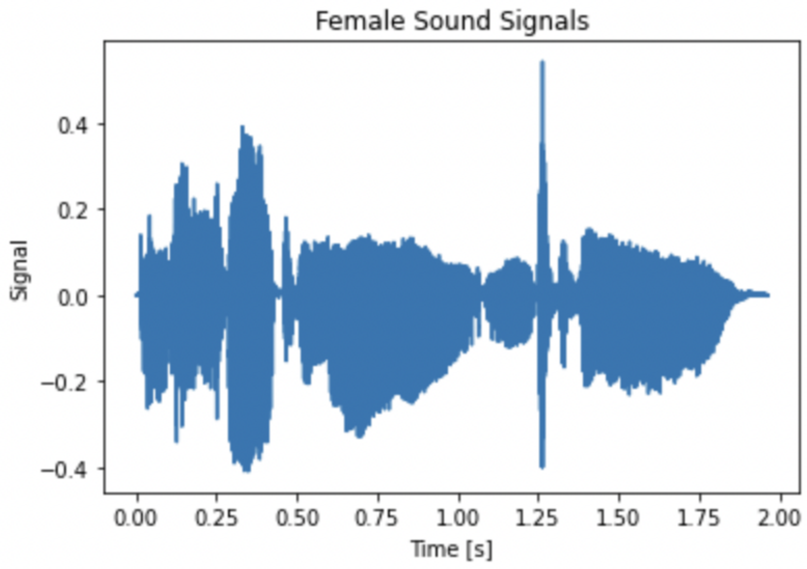
1 Analysing female and music signal over time

The code below was used to observe how the signal over time looks for both the female and the music sounds. This also includes code that was used to zoom in on smaller 20ms intervals and voiced and unvoiced part of the female speech sound. The plots for each of the cases we are asked to observe can be seen in their respective subsections.

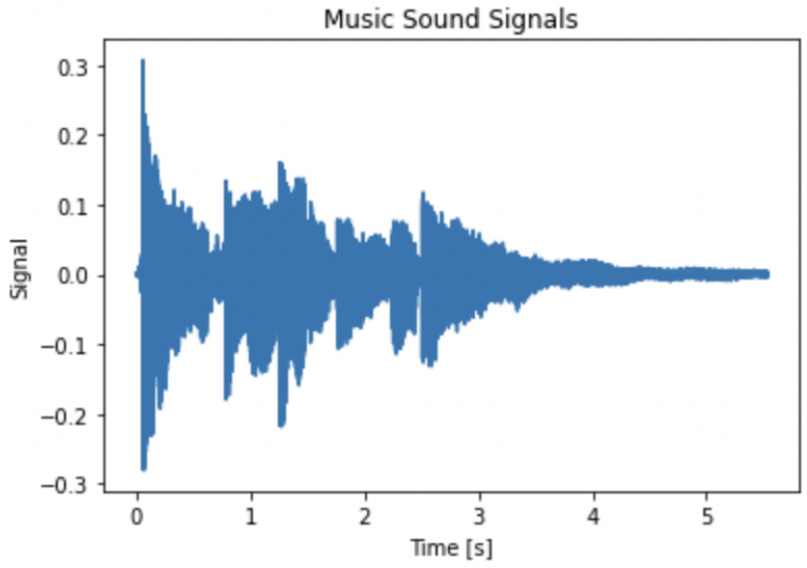


```
#Part 1, load sound file and listen to them. Plot music and female voice signal vs time and zoom in on 20ms intervals in different parts of the graph to see the differences
from scipy.io import wavfile
import sounddevice as sd
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors

femaleSoundPath = "Sounds/female.wav"
musicSoundPath = "Sounds/music.wav"
maleSoundPath = "Sounds/male.wav"

def readAudioFile(path):
    samplerate, data = wavfile.read(path)
    data = data / (2**15)
    return samplerate, data

def plotFullRange(samplerate, data, length, title):
    plotZoomedRange(samplerate, data, length, 0, length, title)

def plotZoomedRange(samplerate, data, length, startInSeconds, endInSeconds, title):
    start = int(samplerate*startInSeconds);
    end = int(samplerate*endInSeconds)
    time = np.linspace(0, length, data.shape[0]);
    plt.figure();
    plt.plot(time[start:end],data[start:end])
    plt.xlabel("Time [s]")
    plt.ylabel("Signal")
    plt.title(title)
    plt.grid()
    plt.show()

#PART 1
#listen to music, male and female audio files
femaleSoundSampleRate, femaleSoundData = readAudioFile(femaleSoundPath)
#sd.play(femaleSoundData, femaleSoundSampleRate)
musicSoundSampleRate, musicSoundData = readAudioFile(musicSoundPath)
#sd.play(musicSoundData, musicSoundSampleRate)
maleSoundSampleRate, maleSoundData = readAudioFile(maleSoundPath)

#get length of audio files
femaleSoundLength = femaleSoundData.shape[0] / femaleSoundSampleRate
print(f"femaleSoundLength = {femaleSoundLength}s")
musicSoundLength = musicSoundData.shape[0] / musicSoundSampleRate
print(f"musicSoundLength = {musicSoundLength}s")
maleSoundLength = maleSoundData.shape[0] / maleSoundSampleRate
print(f"maleSoundLength = {maleSoundLength}s")

#plot music and female voice sound singal vs. time (full range)
plotFullRange(femaleSoundSampleRate, femaleSoundData, femaleSoundLength, "Female Sound Signals Full Range")
plotFullRange(musicSoundSampleRate, musicSoundData, musicSoundLength, "Music Sound Signals Full Range")

# zoom in on 20ms in a few places
# female
plotZoomedRange(femaleSoundSampleRate, femaleSoundData, femaleSoundLength, 0.52, 0.54, "Female Sound Signals 20ms")
plotZoomedRange(femaleSoundSampleRate, femaleSoundData, femaleSoundLength, 0.62, 0.64, "Female Sound Signals 20ms")
plotZoomedRange(femaleSoundSampleRate, femaleSoundData, femaleSoundLength, 1.22, 1.24, "Female Sound Signals 20ms")
plotZoomedRange(femaleSoundSampleRate, femaleSoundData, femaleSoundLength, 1.52, 1.54, "Female Sound Signals 20ms")
plotZoomedRange(femaleSoundSampleRate, femaleSoundData, femaleSoundLength, 1.62, 1.64, "Female Sound Signals 20ms")
# music
plotZoomedRange(musicSoundSampleRate, musicSoundData, musicSoundLength, 0.52, 0.54, "Music Sound Signals 20ms")
plotZoomedRange(musicSoundSampleRate, musicSoundData, musicSoundLength, 1.52, 1.54, "Music Sound Signals 20ms")
plotZoomedRange(musicSoundSampleRate, musicSoundData, musicSoundLength, 2.52, 2.54, "Music Sound Signals 20ms")
plotZoomedRange(musicSoundSampleRate, musicSoundData, musicSoundLength, 3.52, 3.54, "Music Sound Signals 20ms")
plotZoomedRange(musicSoundSampleRate, musicSoundData, musicSoundLength, 4.52, 4.54, "Music Sound Signals 20ms")
```

1.1 Full Range Plots

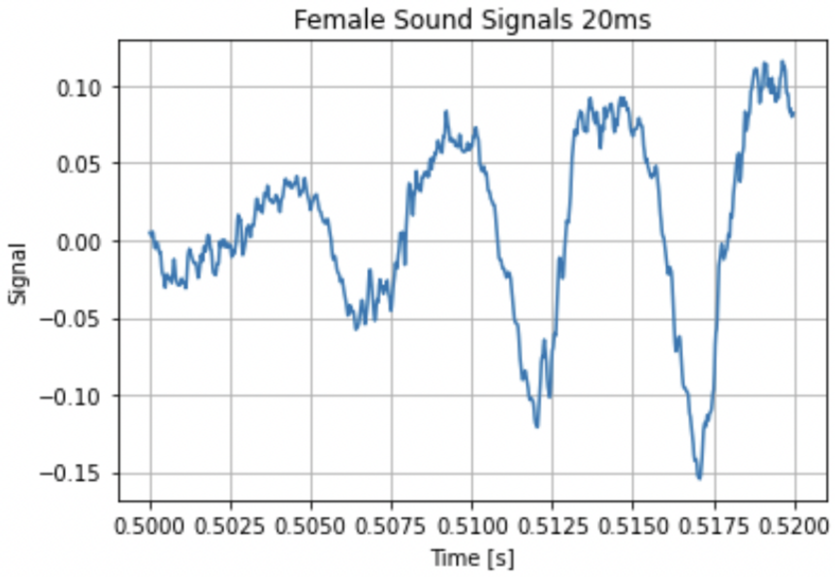


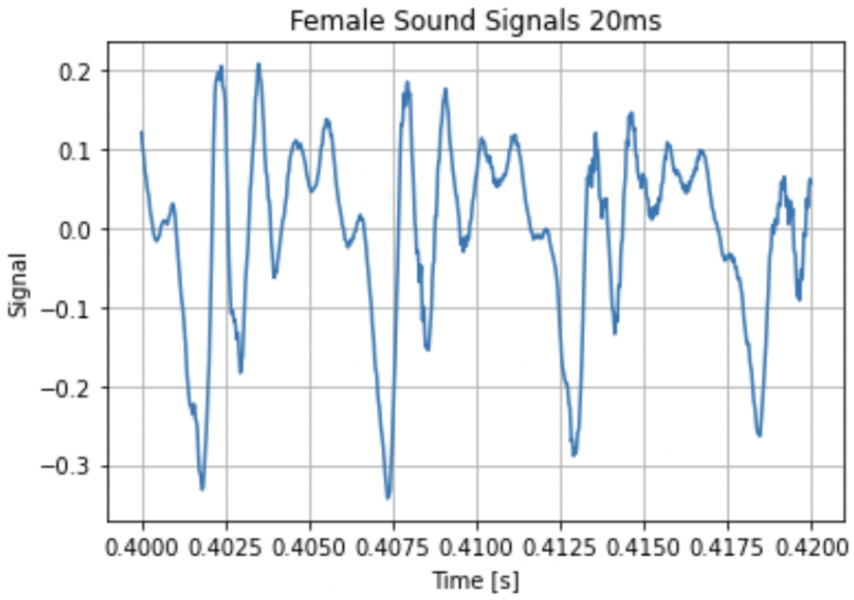
```
musicSoundLength = 5.537959183673469s
```

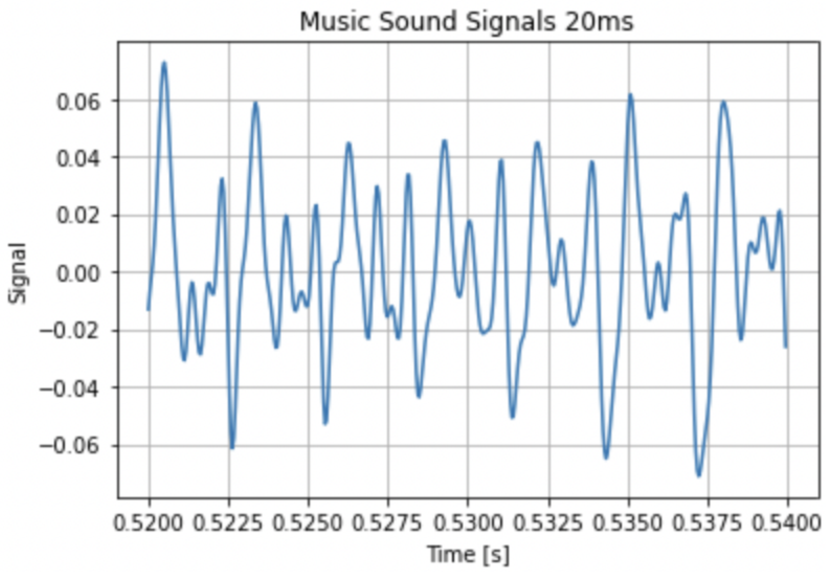


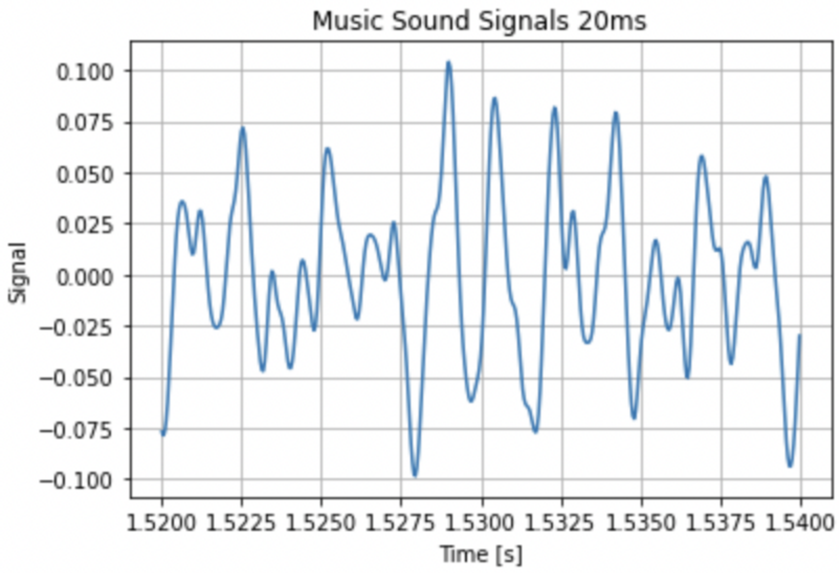
1.2 20ms zooms

Below we can see a few examples of the 20ms zooms for the female and music sounds. Here we can observe the ”oscillatory behaviour” described in the assignment.



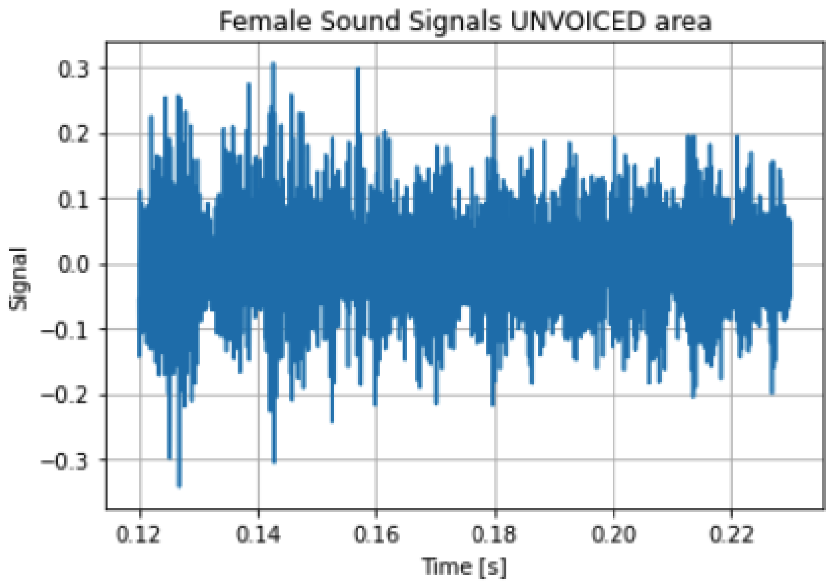


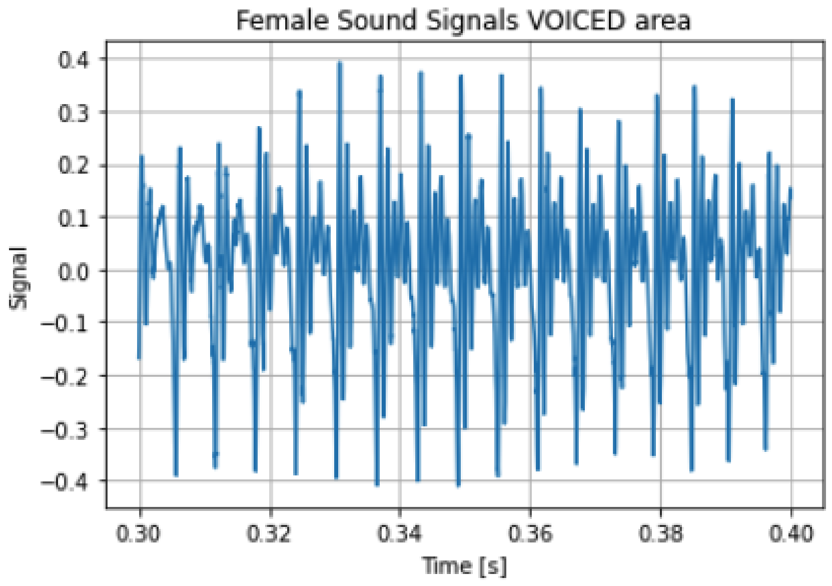

Music Sound Signals 20ms
Signal
Time [s]
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.5200
0.5225
0.5250
0.5275
0.5300
0.5325
0.5350
0.5375
0.5400


Music Sound Signals 20ms
Signal
Time [s]
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
1.5200
1.5225
1.5250
1.5275
1.5300
1.5325
1.5350
1.5375
1.5400

1.3 Voiced vs Unvoiced zoom

The figures below show the signal pattern as we zoom on voiced and unvoiced parts of speech in the female sound. We can see the difference between clearly and can tell the voiced part has more obvious oscillations.



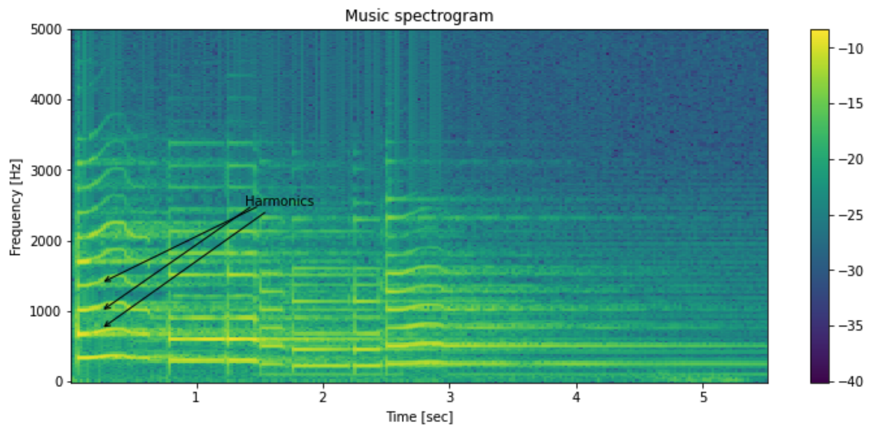


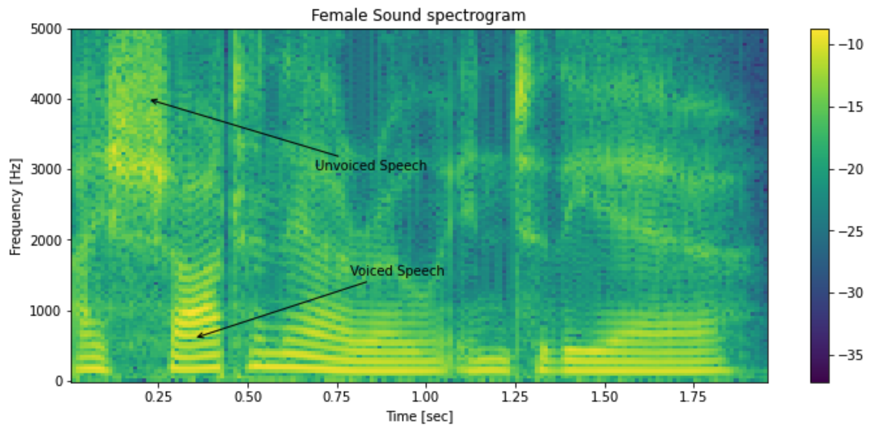
2 Spectrograms

The code below was used to plot spectrograms for all three sounds. The annotation command was used to point out the areas on the plot corresponding to harmonics in the music sound, as well as voiced and unvoiced speech in the female sound.

```python
# ## PART 2 forier transform and spectrograms
from scipy import signal

def plotSpectrogram(data, sampleRate, Title):
    freqs, times, spectrogram = signal.spectrogram(data, sampleRate, window=signal.windows.hamming(1024), noverlap=1024/2)
    spectrogram = np.log(spectrogram)
    plt.figure(figsize=(12, 5))
    plt.pcolormesh(times, freqs, spectrogram)
    plt.ylabel('Frequency [Hz]')
    plt.xlabel('Time [sec]')
    plt.ylim(top = 5000)
    plt.title(Title)
    plt.colorbar()
    return plt

# Music
musicSpectrogramPlot = plotSpectrogram(musicSoundData, musicSoundSampleRate, "Music Spectrogram")
ax = musicSpectrogramPlot.gca()
ax.annotate('Harmonics', xy=(0.25, 750),  xycoords='data',
            xytext=(0.25, 0.5), textcoords='axes fraction',
            arrowprops=dict(color='black', arrowstyle="->")
            )
ax.annotate('', xy=(0.25, 1000),  xycoords='data',
            xytext=(0.26, 0.5), textcoords='axes fraction',
            arrowprops=dict(color='black', arrowstyle="->")
            )
ax.annotate('', xy=(0.25, 1400),  xycoords='data',
            xytext=(0.27, 0.5), textcoords='axes fraction',
            arrowprops=dict(color='black', arrowstyle="->")
            )
musicSpectrogramPlot.show()

# Female
femaleSoundSpectrogramPlot = plotSpectrogram(femaleSoundData, femaleSoundSampleRate, "Female Sound Spectrogram")
ax = femaleSoundSpectrogramPlot.gca()
ax.annotate('Voiced Speech', xy=(0.35, 600),  xycoords='data',
            xytext=(0.4, 0.3), textcoords='axes fraction',
            arrowprops=dict(color='black', arrowstyle="->")
            )
ax.annotate('Unvoiced Speech', xy=(0.22, 4000),  xycoords='data',
            xytext=(0.35, 0.6), textcoords='axes fraction',
            arrowprops=dict(color='black', arrowstyle="->")
            )
femaleSoundSpectrogramPlot.show()

# Male
maleSoundSpectrogramPlot = plotSpectrogram(maleSoundData, maleSoundSampleRate, "Male Sound Spectrogram")
maleSoundSpectrogramPlot.show()
```

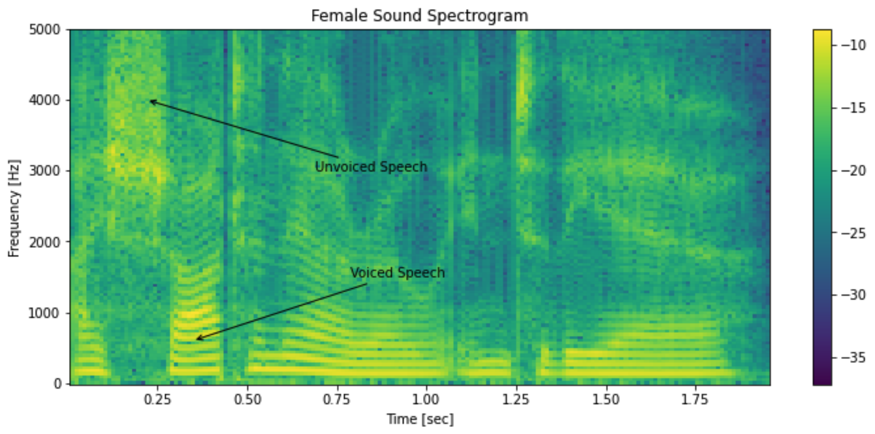
2.1 Female and Music Spectrograms

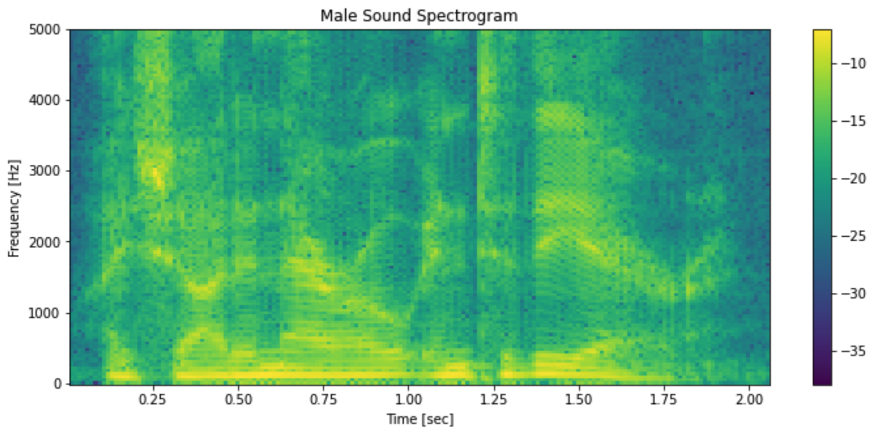




2.2 Female and Male Spectrograms

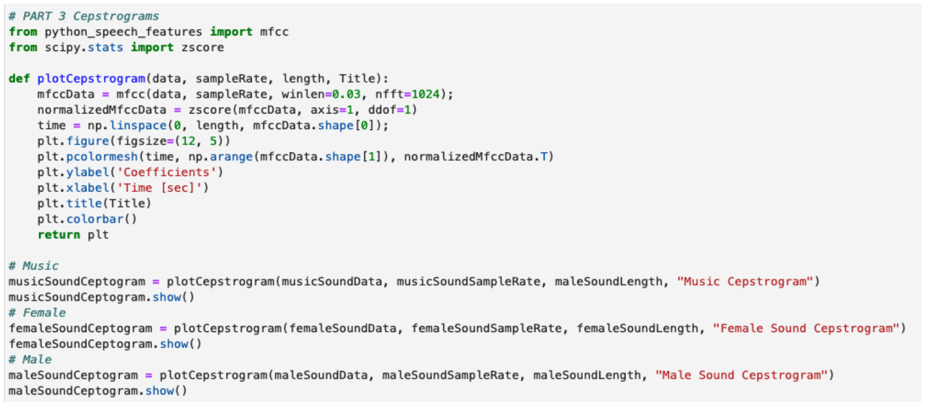
We can see that there is somewhat of a similarity between these two plots compared to the previous two at least. I believe it would be difficult for a human to identify that they are the same spoken phrases but not impossible. It would be difficult for a computer to identify that they are the same phrase also.





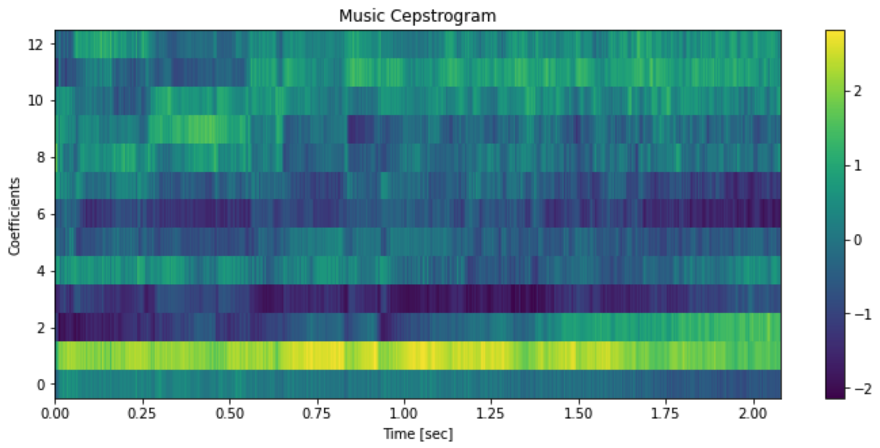
3 Cepstrograms

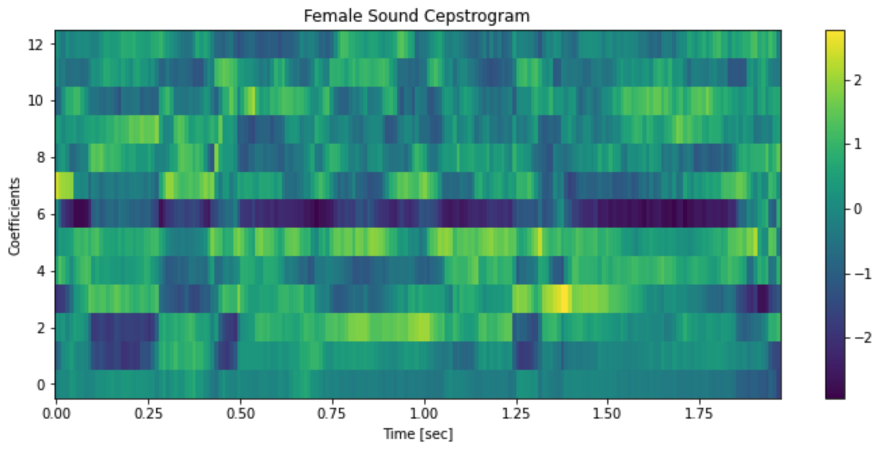
The code below was used to plot cepstrogram for all three sounds.



```python
# PART 3 Cepstrograms
from python_speech_features import mfcc
from scipy.stats import zscore

def plotCepstrogram(data, sampleRate, length, Title):
    mfccData = mfcc(data, sampleRate, winlen=0.03, nfft=1024);
    normalizedMfccData = zscore(mfccData, axis=1, ddof=1)
    time = np.linspace(0, length, mfccData.shape[0]);
    plt.figure(figsize=(12, 5))
    plt.pcolormesh(time, np.arange(mfccData.shape[1]), normalizedMfccData.T)
    plt.ylabel('Coefficients')
    plt.xlabel('Time [sec]')
    plt.title(Title)
    plt.colorbar()
    return plt

# Music
musicSoundCeptogram = plotCepstrogram(musicSoundData, musicSoundSampleRate, maleSoundLength, "Music Cepstrogram")
musicSoundCeptogram.show()
# Female
femaleSoundCeptogram = plotCepstrogram(femaleSoundData, femaleSoundSampleRate, femaleSoundLength, "Female Sound Cepstrogram")
femaleSoundCeptogram.show()
# Male
maleSoundCeptogram = plotCepstrogram(maleSoundData, maleSoundSampleRate, maleSoundLength, "Male Sound Cepstrogram")
maleSoundCeptogram.show()
```

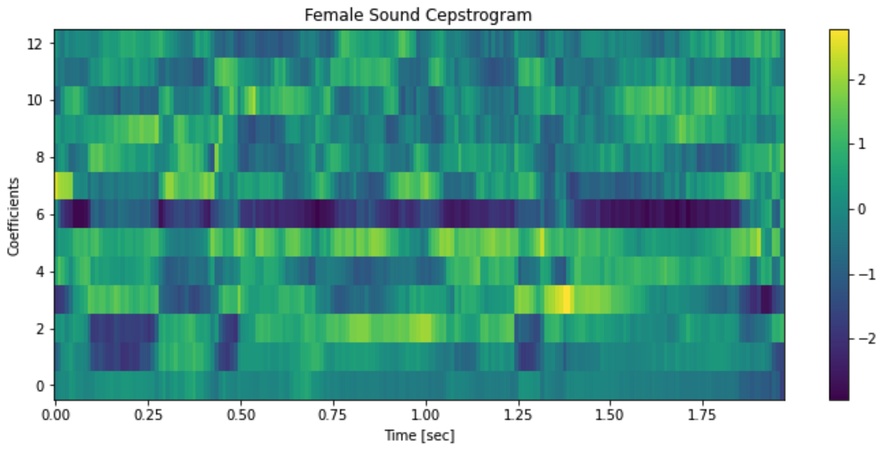
3.1 Female and Music Cepstrograms

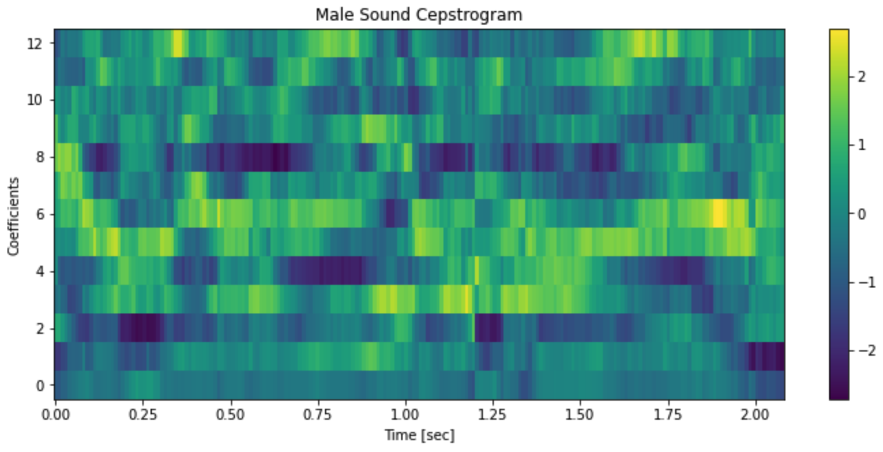




3.2 Female and Male Cepstrograms

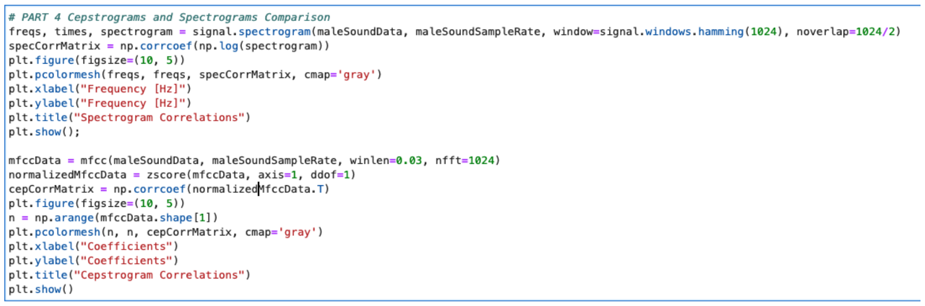
I believe it would still be difficult for a human to identify that they are the same spoken phrases but easier for a computer than previously.





4 Comparing Correlation Matrices

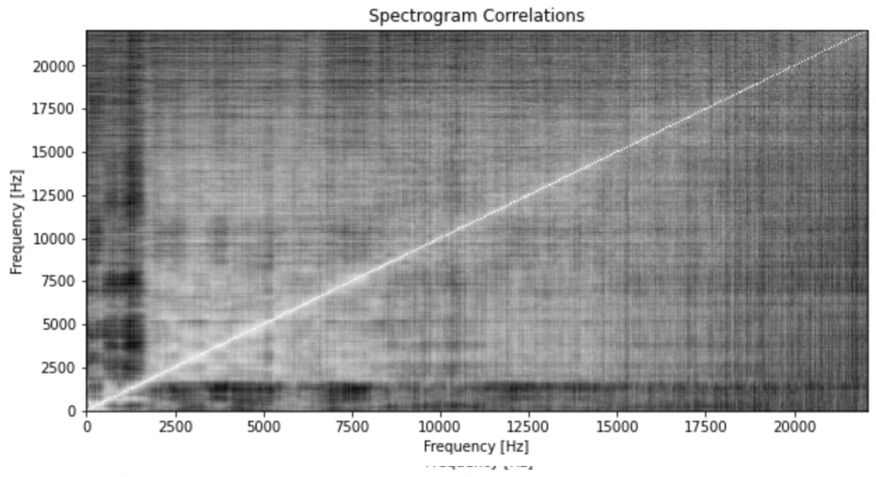
The code below was used to get the correlation matrices of the two sets of feature data and plot them to see which one seems more diagonal and easier to compute.

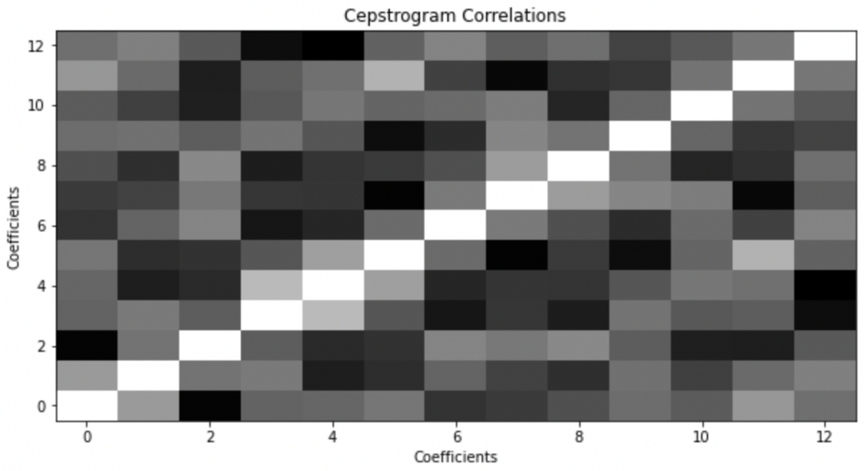


```python
# PART 4 Cepstrograms and Spectrograms Comparison
freqs, times, spectrogram = signal.spectrogram(maleSoundData, maleSoundSampleRate, window=signal.windows.hamming(1024), noverlap=1024/2)
specCorrMatrix = np.corrcoef(np.log(spectrogram))
plt.figure(figsize=(10, 5))
plt.pcolormesh(freqs, freqs, specCorrMatrix, cmap='gray')
plt.xlabel("Frequency [Hz]")
plt.ylabel("Frequency [Hz]")
plt.title("Spectrogram Correlations")
plt.show();

mfccData = mfcc(maleSoundData, maleSoundSampleRate, winlen=0.03, nfft=1024)
normalizedMfccData = zscore(mfccData, axis=1, ddof=1)
cepCorrMatrix = np.corrcoef(normalizedMfccData.T)
plt.figure(figsize=(10, 5))
n = np.arange(mfccData.shape[1])
plt.pcolormesh(n, n, cepCorrMatrix, cmap='gray')
plt.xlabel("Coefficients")
plt.ylabel("Coefficients")
plt.title("Cepstrogram Correlations")
plt.show()
```

4.1 Results

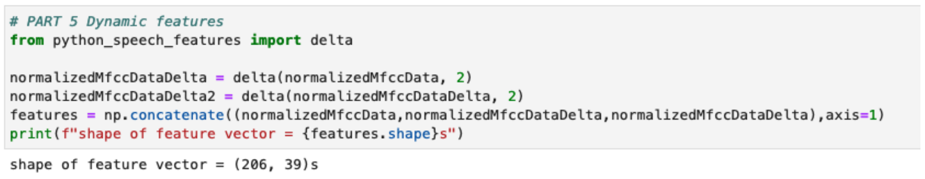
The results are seen below and we can clearly see the the cepstral coefficient series has a more obviously diagonal and easier to compute correlation matrix.





5 Dynamic features

The code below was used to compute the dynamic features and combine them with out static ones to produce our final feature series.



```
# PART 5 Dynamic features
from python_speech_features import delta

normalizedMfccDataDelta = delta(normalizedMfccData, 2)
normalizedMfccDataDelta2 = delta(normalizedMfccDataDelta, 2)
features = np.concatenate((normalizedMfccData,normalizedMfccDataDelta,normalizedMfccDataDelta),axis=1)
print(f"shape of feature vector = {features.shape}s")
```

```
shape of feature vector = (206, 39)s
```

One shortfall of MFCCs could be that they lack robustness for noise signals where one error could change everything.

6 Notes

Code used to answer all questions can be found in its respective section in the jupyter notebook.