Assignment 2

July 11, 2021

Student Name: Shirley Qi

Student ID: A15818203

1 CSE 190 Assignment 2

1.0.1 Markov Chain, LZify, and Speech Formants with Linear Predictive Coding

Instructions:

- This notebook is an interactive assignment; please read and follow the instructions in each cell.
- Cells that require your input (in the form of code or written response) will have 'Question #' above.
- After completing the assignment, please submit this notebook as a PDF and a WAV file of your Fur Elise variation.

1.1 Markov-Based Chord Prediction

In music, certain chord transitions are more likely than others. This idea can be applied as Markov Models, where the first-order temporal relationships between the various chords are captured by the transition probability matrix A.

If we consider only major and minor chords, there are a total of 24 chords in this model (12 major, from C through B, and 12 minor, from C through B), formalized as:

$$\mathcal{A} = \{\mathbf{C}, \mathbf{C}^\sharp, \ldots, \mathbf{B}, \mathbf{Cm}, \mathbf{Cm}^\sharp, \ldots, \mathbf{Bm}\} \tag{1}$$

We use the notation $\alpha i \rightarrow \alpha j$ referring to the transition from state αi to state αj, $i, j \in [1:24]$. For example, the coefficient $a_{1,2}$ expresses the probability for the transition $\alpha_1 \rightarrow \alpha_2$ (corresponding to $\mathbf{C} \rightarrow \mathbf{C}^\sharp$), whereas $a_{1,8}$ expresses the probability for $\alpha_1 \rightarrow \alpha_8$ (corresponding to $\mathbf{C} \rightarrow \mathbf{G}$). In real music, the change from a tonic to the dominant is much more likely than transposing by one semitone, so that the probability $a_{1,8}$ should be much larger than $a_{1,2}$. The coefficients $a_{i,i}$ express the probability of staying in state α_i (i.e., $\alpha_i \rightarrow \alpha_i$), $i \in [1:24]$. These coefficients are also referred to as **self-transition** probabilities.

A transition probability matrix can be specified in many ways. For example, the matrix may be defined manually by a music expert based on rules from harmony theory. The most common

approach is to generate such a matrix automatically by estimating the transition probabilities from labeled data.

In the following exercise, you will create a Markov model by determining transition probabilities found in the music of the Beatles using bigrams (pairs of adjacent elements) in labeled frame sequences from a subset of the Beatles Corpus.

For this assignment, assume each row in the dataset represents a chord that has followed the chord on the previous row in a Beatles song. When parsing the file for your model, you may discard any chord references beyond key and major/minor quality. For example, if a row reads 'Bbm7', you would parse for the key (B-flat) and quality (minor), but discard the extra information that it is a 7th chord.

[2]:
```
pip install jchord
```

```
Collecting jchord
  Downloading jchord-2.0.0-py3-none-any.whl (26 kB)
Installing collected packages: jchord
Successfully installed jchord-2.0.0
Note: you may need to restart the kernel to use updated packages.
```

[46]:
```
pip install mido
```

```
Collecting mido
  Downloading mido-1.2.10-py2.py3-none-any.whl (51 kB)
     |                  | 51 kB 3.0 MB/s eta 0:00:011
Installing collected packages: mido
Successfully installed mido-1.2.10
Note: you may need to restart the kernel to use updated packages.
```

[47]:
```
import numpy as np
import pandas as pd
from collections import Counter
from numpy.random import multinomial as randm
from numpy import where
import scipy.signal as si
import IPython.display as ipd
import librosa
import scipy
from matplotlib import patches
import librosa.display as ld
import music21
from music21 import midi as midi21
from music21 import stream
from jchord.progressions import ChordProgression, MidiConversionSettings
import copy
import matplotlib.pyplot as plt
```

```
np.random.seed(42)

data = pd.read_csv('Liverpool_band_chord_sequence.csv')

def preprocess(df):
    df[df['chords'] == 'Bb'] = 'A#'
    chords = ' '.join(map(str, df['chords']))
    return chords

data = preprocess(data)


def play(x):
    """Returns nothing. Outputs a midi realization of x, a note or stream.
    Primarily for use in notebooks and web environments.
    """
    prog = ChordProgression.from_string(x)
    prog.to_midi(MidiConversionSettings(filename="example.midi", tempo=100,␣
 ↪beats_per_chord=4, instrument=4))
    mf = midi21.MidiFile()
    mf.open("example.midi")
    mf.read()
    mf.close()
    s = midi21.translate.midiFileToStream(mf)
    myStream = stream.Stream()

    myStream.append(s)
    x = myStream

    if isinstance(x, stream.Stream):
        x = copy.deepcopy(x)
        for subStream in x.recurse(streamsOnly=True, includeSelf=True):
            mss = subStream.getElementsByClass(stream.Measure)
            for ms in mss:
                ms.offset += 1.0
    if isinstance(x, music21.note.Note):
        s = stream.Stream()
        s.append(music21.note.Rest(1))
        s.append(x)
        x = s
    x.show('midi')
```

[48]:
```
play(data)
```

```
<IPython.core.display.HTML object>
```

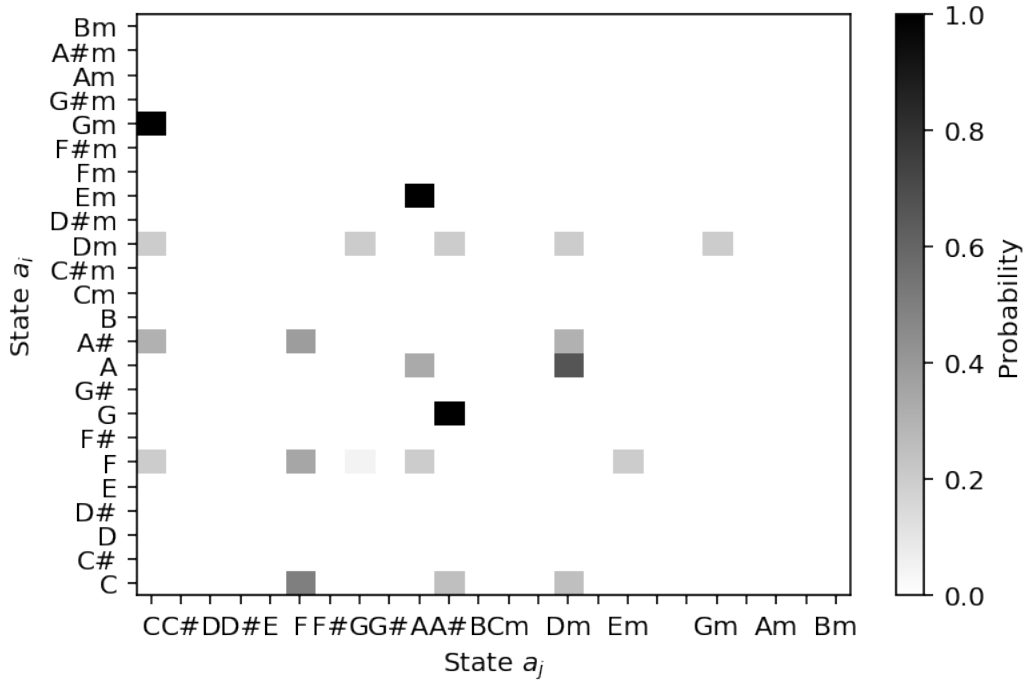
Question 1 (20 points) Using the above data, generate a 24x24 matrix, where each matrix element (i,j) is the transition probability from chord i to chord j.

[49]:
```
HMM = np.zeros((24,24))

### Your code here:
### C C# D D# E F F# G G# A A# B Cm C#m Dm D#m Em Fm F#m Gm G#m Am A#m Bm
chords_list = ["C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B",␣
 ↪"Cm", "C#m", "Dm", "D#m", "Em", "Fm", "F#m", "Gm", "G#m", "Am", "A#m", "Bm"]
chords_dict = {
    "C": 0,
    "C#": 1,
    "D": 2,
    "D#": 3,
    "E": 4,
    "F": 5,
    "F#": 6,
    "G": 7,
    "G#": 8,
    "A": 9,
    "A#": 10,
    "B": 11,
    "Cm": 12,
    "C#m": 13,
    "Dm": 14,
    "D#m": 15,
    "Em": 16,
    "Fm": 17,
    "F#m": 18,
    "Gm": 19,
    "G#m": 20,
    "Am": 21,
    "A#m": 22,
    "Bm": 23
}
chords_arr = data.split()
new_chords = []

### Process chords
for i in range(len(chords_arr)):
    curr = chords_arr[i]
    if (len(curr) == 2):
        if (curr[1] != "#" and curr[1] != "m"):
            curr = curr[0]
    elif (len(curr) > 2):
        if (curr[1] != "#" and curr[1] != "m"):
            curr = curr[0]
        elif (curr[1] == "#"):
            if (curr[2] == "m"):
                curr = curr[:3]
```

```
            else:
                curr = curr[:2]
        elif (curr[1] == "m"):
            curr = curr[:2]
    new_chords.append(curr)

### Construct transition matrix
for i in range(len(new_chords)-1):
    curr = new_chords[i]
    next_chord = new_chords[i+1]
    ind1 = chords_dict[curr]
    ind2 = chords_dict[next_chord]
    HMM[ind1, ind2] += 1

for i in range(len(HMM)):
    total = sum(HMM[i])
    if (total != 0):
        HMM[i] /= total
```

[50]:
```
def plot_transition_matrix(A, ax=None, xlabel='State $a_j$', ylabel='State␣
↪$a_i$', title='', clim=[-6, 0], cmap='gray_r'):

    im = plt.imshow(A, origin='lower', aspect='auto', cmap=cmap)
    im.set_clim([0, 1])
    cbar = plt.colorbar(im)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    cbar.ax.set_ylabel('Probability')

    chroma_label = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#',␣
↪'B']
    chord_label_maj = chroma_label
    chord_label_min = [s + 'm' for s in chroma_label]
    chord_labels = chord_label_maj + chord_label_min
    chord_labels_squeezed = chord_labels.copy()
    for k in [13, 15, 17, 18, 20, 22]:
        chord_labels_squeezed[k] = ''

    plt.xticks(np.arange(24), labels=chord_labels_squeezed )
    plt.yticks(np.arange(24), labels=chord_labels)

    return im

plot_transition_matrix(HMM)
```

[50]: <matplotlib.image.AxesImage at 0x7fdd88e3ae20>



Question 2 (10 points) Using your HMM, you will create your own 16-measure Beatles hits! For your first song, beginning on C major, select each next chord by choosing the chord with the largest transition probability from the current chord.

Make sure your chord progression string is formatted like this: 'C Dm G C'

Otherwise, it may not play in the in-browser MIDI player.

[51]:
```
my_first_beatles_hit = ''
my_first_beatles_hit_arr = []

### Your code here:
curr = 'C'
for i in range(16):
    my_first_beatles_hit_arr.append(curr)
    ind = chords_dict[curr]
    row = HMM[ind]
    curr = chords_list[np.argmax(row)]

my_first_beatles_hit = " ".join(my_first_beatles_hit_arr)

print(my_first_beatles_hit)
play(my_first_beatles_hit)
```

```
C F F F F F F F F F F F F F F
```

```
<IPython.core.display.HTML object>
```

Question 3 (10 points) For your second song, beginning on C major, select each next chord at random according to the probabilities of your HMM. For example, if C major transitions to G major with probability .5, F major with probability .25, and D minor with probability .25, then your next chord should be selected randomly from (G, F, Dm) with probability of selection (.5, .25, .25) respectively.

[54]:
```
my_second_beatles_hit = ''
my_second_beatles_hit_arr = []

### Your code here:
curr = 'C'
for i in range(16):
    my_second_beatles_hit_arr.append(curr)
    ind = chords_dict[curr]
    row = HMM[ind]
    rand = np.random.choice(24, 24, p=row)
    curr = chords_list[np.bincount(rand).argmax()]

my_second_beatles_hit = " ".join(my_second_beatles_hit_arr)

print(my_second_beatles_hit)
play(my_second_beatles_hit)
```

```
C F Em A Dm C F F C F F Em A Dm Gm C
```

```
<IPython.core.display.HTML object>
```

1.2 LZify: Applying Universal Prediction to Musical Style

In this section, you will implement the Incremental Parsing (IP) from the Lempel Ziv LZ78 method for creating a dictionary of motifs. These motifs are later used to generate new sequences resembling the input sequence.

Please read the algorithm in Assayag, Dubnov, and Delerue's "Guessing the Composer's Mind" (available at https://pdfs.semanticscholar.org/0181/236e1b417c8dd5dddd1f919583893f7a9026.pdf).

The IPMotif function should compute the motif dictionary discovered in the text. It uses Incremental Parsing method to parse the text into unseen motifs.

Question 4 (15 points)

[55]:
```
def IPMotif(text):
    """Compute an associative dictionary (the motif dictionary)."""

    dictionary = {}
    motif = ''
    ### Your Code Here:
```

```
    for i in range(len(text)):
        motif = motif + text[i]
        if motif in dictionary:
            dictionary[motif] += 1
        else:
            dictionary[motif] = 1
            motif = ''

    return dictionary
```

The text below contains an excerpt of Beethoven's Fur Elise, written as MML (Music Macro Language). MML is used to represent musical melodies as text.

You can read more about MML syntax here: https://en.wikipedia.org/wiki/Music_Macro_Language

You can play with MML in this webapp: http://benjaminsoule.fr/tools/vmml/

Try playing the text below in the webapp to hear sample output.

[56]:
```
text = ␣
 ↪'o6ed+ed+ec-dc>aceabeg+bb+e<ed+ed+ec-dc>aceabe<c>bab<cde>g<fed>f<edc>e<dc>be<eeed+ed+ec-dc>
dict1 = IPMotif(text)
print(dict1)
```

```
{'o': 1, '6': 1, 'e': 13, 'd': 5, '+': 7, 'ed': 6, '+e': 4, 'c': 9, '-': 3,
'dc': 3, '>': 3, 'a': 4, 'ce': 3, 'ab': 1, 'eg': 1, '+b': 2, 'b': 5, '+e<': 2,
'ed+': 4, 'ed+e': 3, 'c-': 1, 'dc>': 2, 'ac': 1, 'ea': 1, 'be': 2, '<': 4, 'c>':
3, 'ba': 1, 'b<': 1, 'cd': 1, 'e>': 1, 'g': 1, '<f': 1, 'ed>': 1, 'f': 1, '<e':
1, 'dc>e': 1, '<d': 1, 'c>b': 1, 'e<': 2, 'ee': 1, 'ed+ed': 1, '+ec': 1, '-d':
2, 'c>a': 1, 'cea': 2, 'beg': 1, '+bb': 1, '+e<e': 1, 'd+': 1, 'ed+ec': 1,
'-dc': 1, '>a': 1, 'ceab': 1, 'e<c': 1, '>b': 1}
```

Next, implement the IPContinuation and Normalize functions. The IPContinuation function transforms the IPMotif dictionary into a tree-like representation to allow finding continuations for new motifs. The Normalize function turns the counters in every element of the IPContinuation dictionary into probabilities.

Question 5 (10 points)

[122]:
```
def IPContinuation(dict1):
    """Compute continuation dictionary from a motif dictionary"""

    dict2 = {}

    ### Your Code Here:
    for key in dict1.keys():
        if (len(key) == 1):
            W = ""
            k = key
        else:
            W = key[:len(key)-1]
```

```
            k = key[len(key)-1]
        counter = str(dict1[key])
        lst = [k, counter]
        if W in dict2:
            dict2[W].append(lst)
        else:
            dict2[W] = [lst]

    Normalize(dict2)

    return dict2
```

Question 6 (10 points)

[131]:
```
def Normalize(dict2):
    """Turns the counters in every element of dict2 to probabilities"""

    ### Your Code Here:
    for key in dict2.keys():
        total = 0
        value_list = dict2[key]

        for lst in value_list:
            total += int(lst[1])

        for lst in value_list:
            lst[1] = int(lst[1])/total

    return dict2
```

[132]:
```
dict2 = IPContinuation(dict1)
print(dict2)
```

```
{'': [['o', 0.017543859649122806], ['6', 0.017543859649122806], ['e',
0.22807017543859648], ['d', 0.08771929824561403], ['+', 0.12280701754385964],
['c', 0.15789473684210525], ['-', 0.05263157894736842], ['>',
0.05263157894736842], ['a', 0.07017543859649122], ['b', 0.08771929824561403],
['<', 0.07017543859649122], ['g', 0.017543859649122806], ['f',
0.017543859649122806]], 'e': [['d', 0.5], ['g', 0.08333333333333333], ['a',
0.08333333333333333], ['>', 0.08333333333333333], ['<', 0.16666666666666666],
['e', 0.08333333333333333]], '+': [['e', 0.6666666666666666], ['b',
0.3333333333333333]], 'd': [['c', 0.75], ['+', 0.25]], 'c': [['e', 0.375], ['-',
0.125], ['>', 0.375], ['d', 0.125]], 'a': [['b', 0.5], ['c', 0.5]], '+e': [['<',
0.6666666666666666], ['c', 0.3333333333333333]], 'ed': [['+', 0.8], ['>', 0.2]],
'ed+': [['e', 1.0]], 'dc': [['>', 1.0]], 'b': [['e', 0.5], ['a', 0.25], ['<',
0.25]], '<': [['f', 0.3333333333333333], ['e', 0.3333333333333333], ['d',
0.3333333333333333]], 'dc>': [['e', 1.0]], 'c>': [['b', 0.5], ['a', 0.5]],
'ed+e': [['d', 0.5], ['c', 0.5]], '-': [['d', 1.0]], 'ce': [['a', 1.0]], 'be':
```

```
[['g', 1.0]], '+b': [['b', 1.0]], '+e<': [['e', 1.0]], '-d': [['c', 1.0]], '>':
[['a', 0.5], ['b', 0.5]], 'cea': [['b', 1.0]], 'e<': [['c', 1.0]]}
```

Genering a new sequence is done by traversing the IPContinuation tree and selecting possible branches according to their weights. If motif is not found, its last symbol is removed and the process is repeated for a shorter motif.

[136]:
```
def IPGenerate(n,dict2):
    p = 0
    out = ""
    for k in range(n):
        while True:
            context = out[-p:]
            if context in dict2:
                prob = [tup[1] for tup in dict2[context]]
                conti = where(randm(1,prob))[0][0]
                cont = dict2[context][conti][0]
                out = out + cont
                break
            else:
                p = p-1
    return out
out = IPGenerate(92,dict2)
print(out)
```

```
eabegdc>ed+ecdc>ed+ed+eceababegd+ec>babegc>abeged+ed+ed+ed>ab<dc>eg+e<ed+ed+ec>a
begb<feac>ab
```

Question 7 (5 points) Paste your output in the online MML player, and listen to your piece. Do you hear elements of Fur Elise in your composition? What are some of the differences in the output from the original?

Please export the WAV file of your output from the webapp, and submit the WAV file with your assignment.

There were some elements of Fur Elise in the composition. For example, the repeating "ed+" or the "begbb", which comes up quite frequently in Fur Elise. For differences, the generated composition had very random octaves, and would at some points just go lower and lower in octave. While the notes were similar, the octaves generated were not.

A few important points: 1. The method captures the "texture" of the language but not it's meaning. 2. We could parse a new text using IPMotifs from two languages, then count the length and number of motifs in order to decide what was the language of the new text. 3. In order to use this method with musical information, we need first to translate audio to features, or in case of polyphonic midi change this into some proper representation. One possibility is using virtual fundamental or chroma for harmony, or some other specialized representation to capture repetition in terms of other specific musical properties.

1.3 Speech Formants and LPC

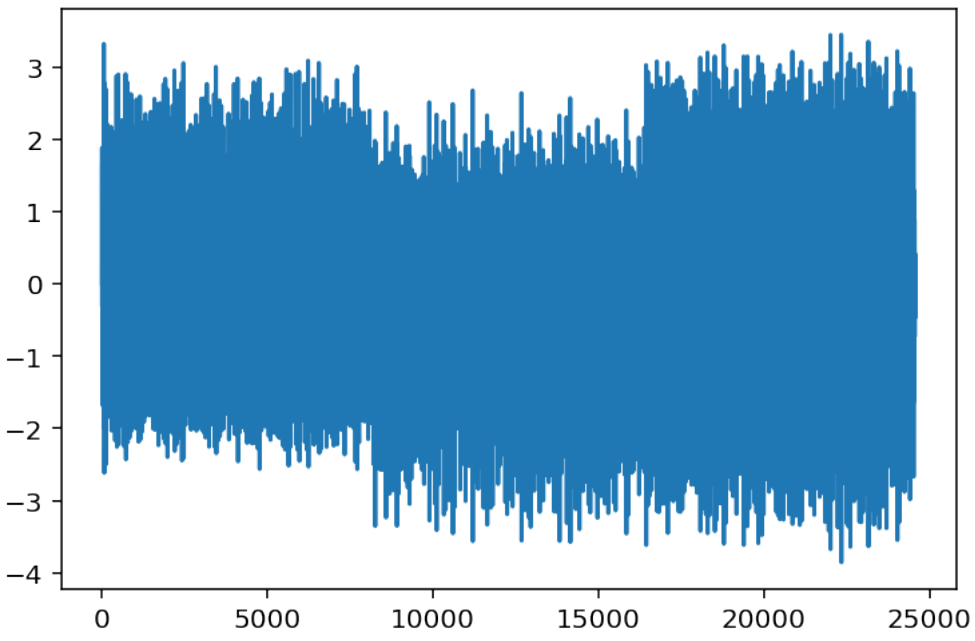
In this section, you will synthesize vowel sounds, and investigate the frequencies in vowels from your own voice.

[137]:
```
Fdict = {
    'mystery_1':[[328, 2208, 2885],[27,80,575]],
    'mystery_2':[[504, 868, 2654],[62,   108,  299]],
    'mystery_3':[[700, 1220, 2600],[130,   70,  160]]
    } # Formant frequencies in Hz

def excitation(f0,jitt,dur,nharm=None,unvoiced=False):
    w0T = 2*np.pi*f0/fs

    if nharm == None:
        nharm = int((fs/2)/f0) # number of harmonics
    nsamps = fs*dur
    sig = np.zeros(nsamps)
    ph = np.random.uniform(size=nsamps)*2*np.pi
    n = np.arange(nsamps)

    if unvoiced:
        sig = np.random.normal(size=nsamps)
    else:
    # Synthesize bandlimited impulse train
        for i in range(1,nharm):
            sig = sig + np.cos(i*w0T*n + jitt*ph)

    sig = sig/max(sig)
    return sig

def voca(sig,F,Fb):
    R = np.exp(-np.pi*Fb/fs);      # Pole radii
    theta = 2*np.pi*F/fs;      # Pole angles
    poles = R * np.exp(1j*theta) # Poles[B,A] = zpk2tf(0,np.concatenate((poles,␣
 ↪np.conj(poles))),1) # Filter from zeros and poles

    [B,A] = si.zpk2tf(0,np.concatenate((poles, np.conj(poles))),1) # Filter␣
 ↪from zeros and poles

    speech = si.lfilter(B, A, sig)
    speech = speech/np.std(speech)
    return speech,B,A

fs = 8192 # 22050  % Sampling rate in Hz ("telephone quality" for speed)

vowels = list(Fdict.keys())
f0 = 150 # Pitch in Hz
```

```
dur = 1 #one second in duration
ji = 0.1 #0.1
ex = excitation(f0,ji,dur)

text = ['mystery_1','mystery_2','mystery_3']

speech = np.zeros(1)
for t in text:
    F = np.array(Fdict[t][0])
    Fb = np.array(Fdict[t][1])
    print(t)

    vow,B,A = voca(ex,F,Fb)
    speech = np.concatenate((speech,vow))

speech = speech/np.std(speech)
plt.plot(speech)
```

```
mystery_1
mystery_2
mystery_3
```

[137]: [<matplotlib.lines.Line2D at 0x7fdd9a6b3040>]



[138]:
```
ipd.Audio(speech, rate=fs)
```

[138]: <IPython.lib.display.Audio object>

Question 8 (6 points) Based on the audio output, what vowels were synthesized as mystery_1, mystery_2, and mystery_3? Please specify using a word; for example, if you heard an 'oo' sound as in 'hoot', you may answer with the word "hoot".

mystery_1: heat

mystery_2: hope

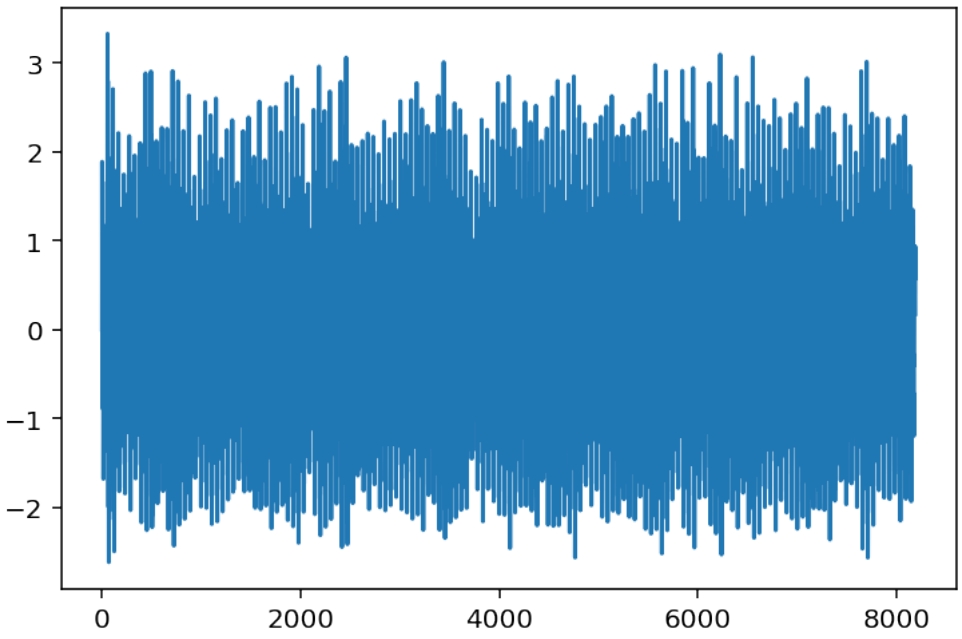
mystery_3: ha

Now we will examine just one vowel in greater detail. Select one mystery vowel to analyze below:

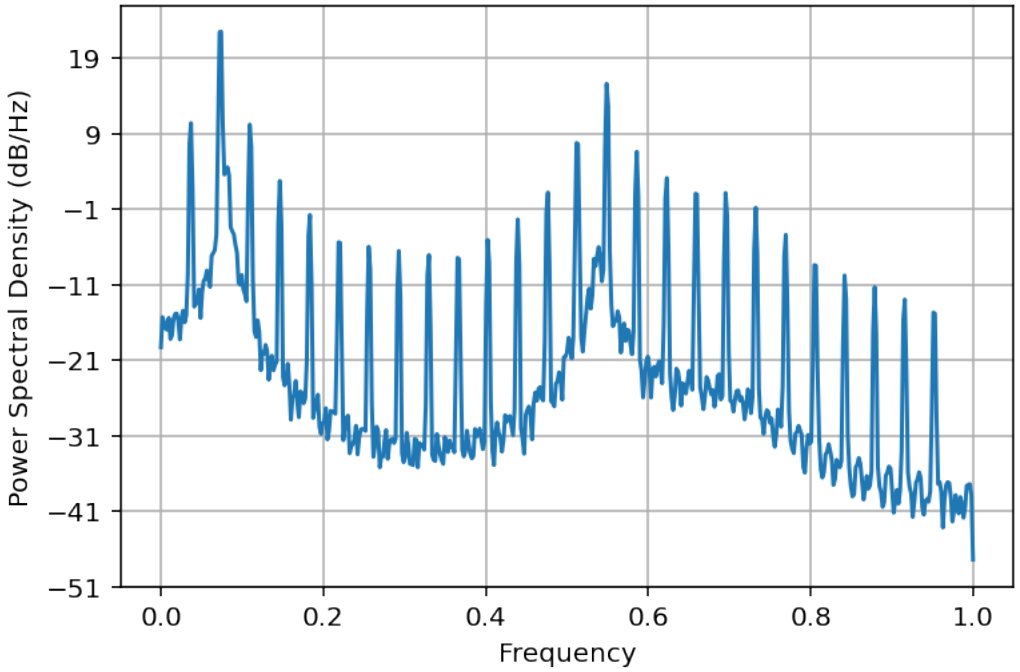
[142]:
```
%matplotlib inline

### Modify the line below:
text = ['mystery_1']

speech = np.zeros(1)
for t in text:
    F = np.array(Fdict[t][0])
    Fb = np.array(Fdict[t][1])
    print(t)

    vow,B,A = voca(ex,F,Fb)
    speech = np.concatenate((speech,vow))

speech = speech/np.std(speech)
plt.plot(speech)
```

```
mystery_1
```

[142]: [<matplotlib.lines.Line2D at 0x7fdd88e20310>]

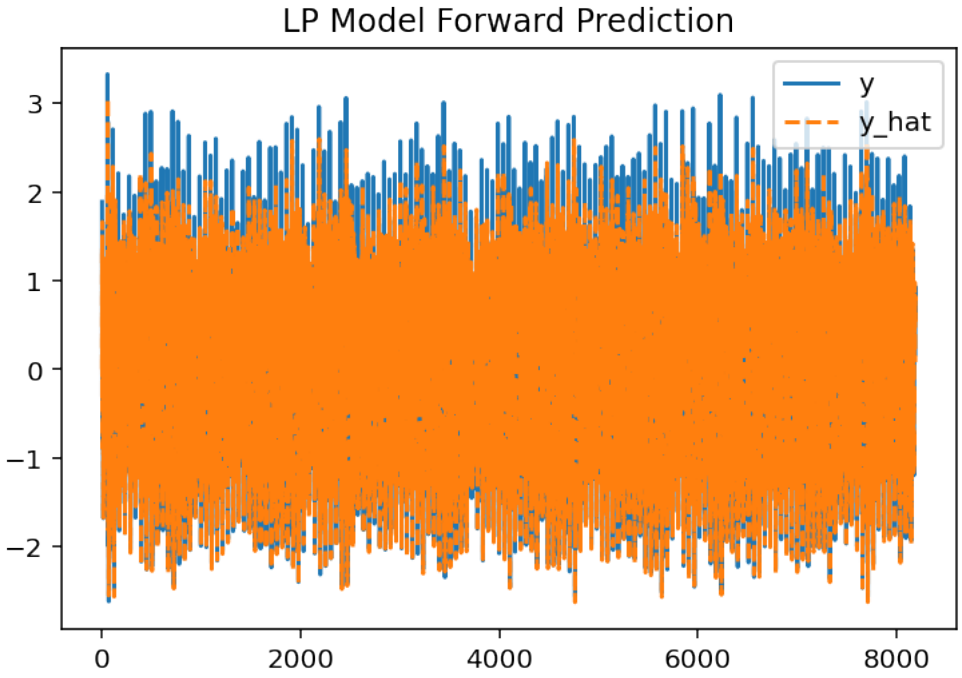


[143]:
```
plt.psd(speech, 1024)
plt.show()
```

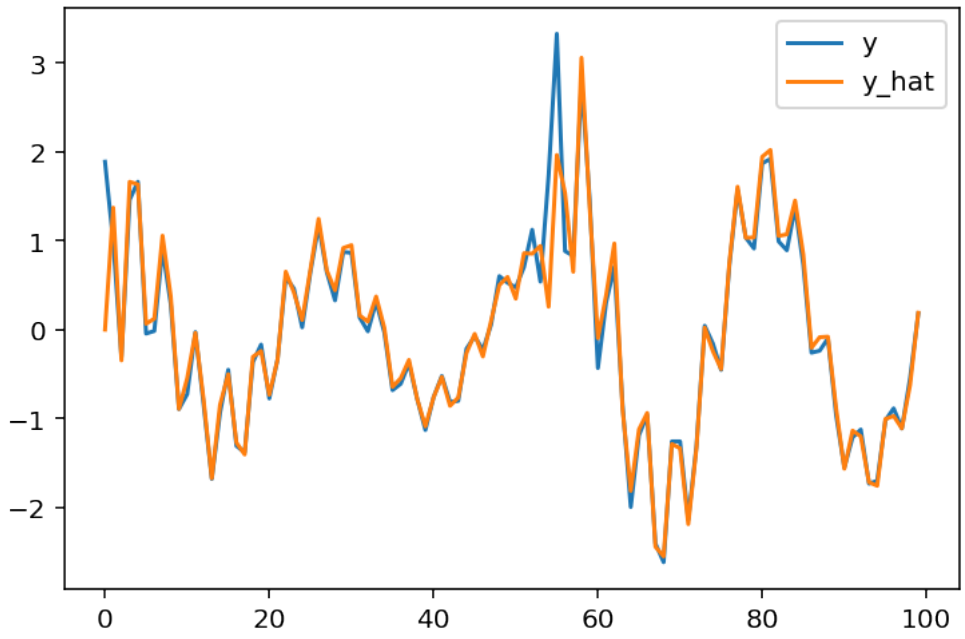


[144]:
```
lpc_order = 10
s = speech

a = librosa.core.lpc(s, lpc_order)
print(a)
s_hat = scipy.signal.lfilter([0] + -1*a[1:], [1], s)
s_err = s[1:] - s_hat[:-1]
plt.plot(s[1:])
plt.plot(s_hat[:-1], linestyle='--')
plt.legend(['y', 'y_hat'])
plt.title('LP Model Forward Prediction')
plt.show()
```

```
[ 1.         -0.72650243  0.56145505 -1.29063747  0.45925372 -0.27741644
  0.63971058 -0.148379    0.11613571 -0.02152859  0.06008777]
```



[145]:
```
plt.plot(s[1:101])
plt.plot(s_hat[:100])
plt.legend(['y', 'y_hat'])
plt.show()
```



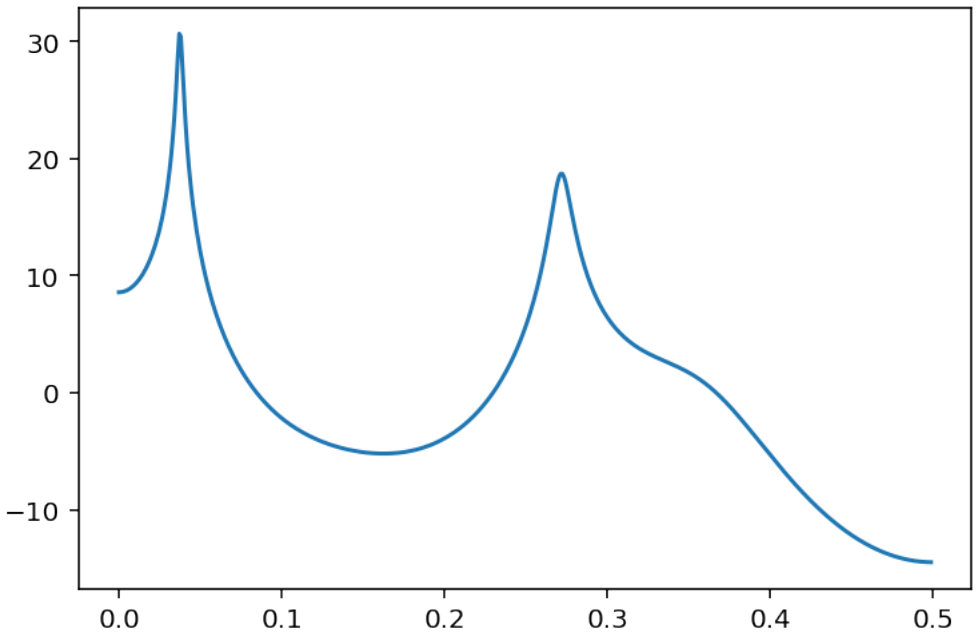
Question 9 (4 points) What is being visualized as y and y_hat on the above plot?

y is the actual speech data, and y_hat is the data generated with LPC.

[146]:
```
w,h = si.freqz(b=1,a = a, fs=1)
plt.plot(w,20*np.log10(h))
```

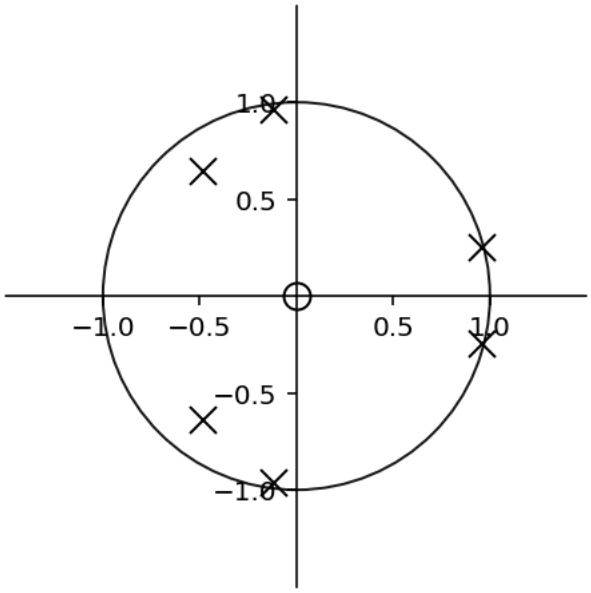
```
/Users/shirleyqi/opt/anaconda3/lib/python3.8/site-
packages/numpy/core/_asarray.py:102: ComplexWarning: Casting complex values to
real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

[146]:
```
[<matplotlib.lines.Line2D at 0x7fdd88ace220>]
```



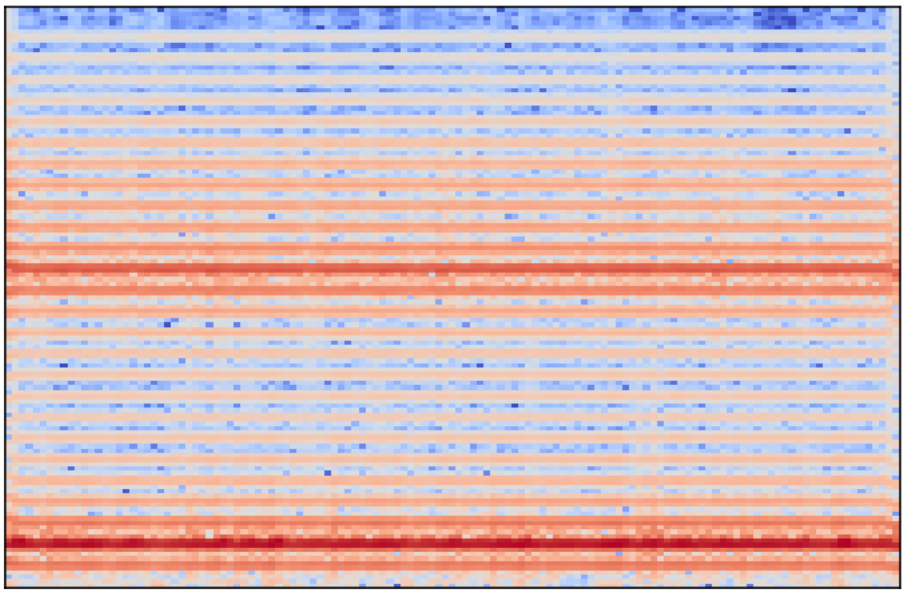
[147]:
```
z,p,k = si.tf2zpk(B,A)

unit_circle = patches.Circle((0,0), radius=1, fill=False, color='black',␣
 ↪ls='solid', alpha=0.9)
ax = plt.subplot(111)
ax.add_patch(unit_circle)
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('center')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
r = 1.5; plt.axis('scaled'); plt.axis([-r, r, -r, r])
ticks = [-1, -.5, .5, 1]; plt.xticks(ticks); plt.yticks(ticks)

plt.plot(z.real, z.imag, 'ko', fillstyle='none', ms = 10)
plt.plot(p.real, p.imag, 'kx', fillstyle='none', ms = 10)
```

[147]:
```
[<matplotlib.lines.Line2D at 0x7fdd88c58eb0>]
```



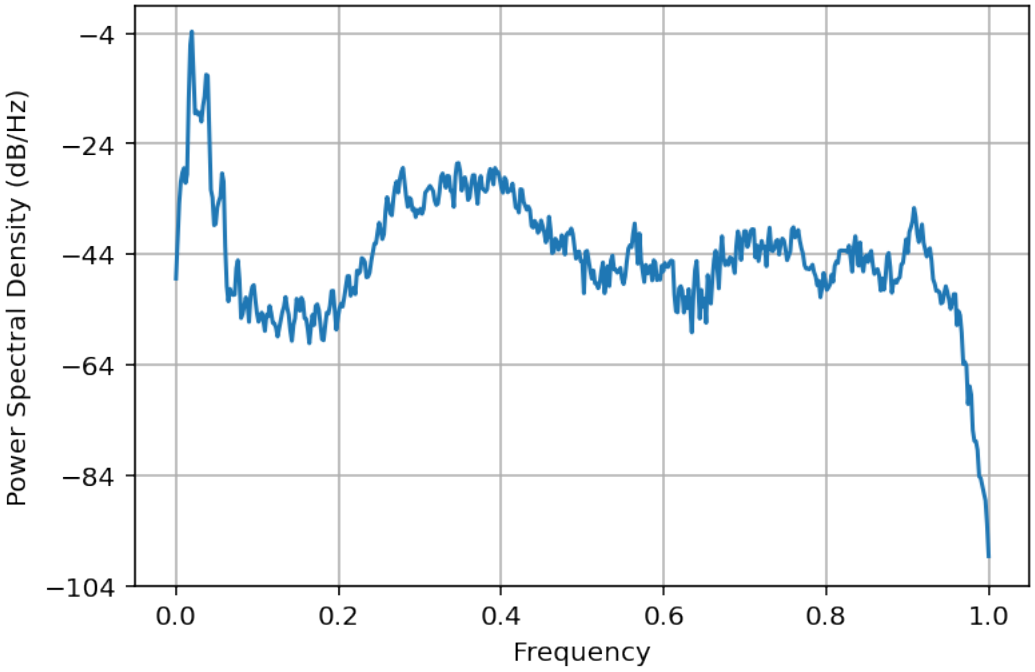
[148]:
```
D = np.abs(librosa.stft(s,n_fft=256,hop_length = 64))
ld.specshow(librosa.amplitude_to_db(D))
```

[148]: <matplotlib.collections.QuadMesh at 0x7fdd88c8bc70>

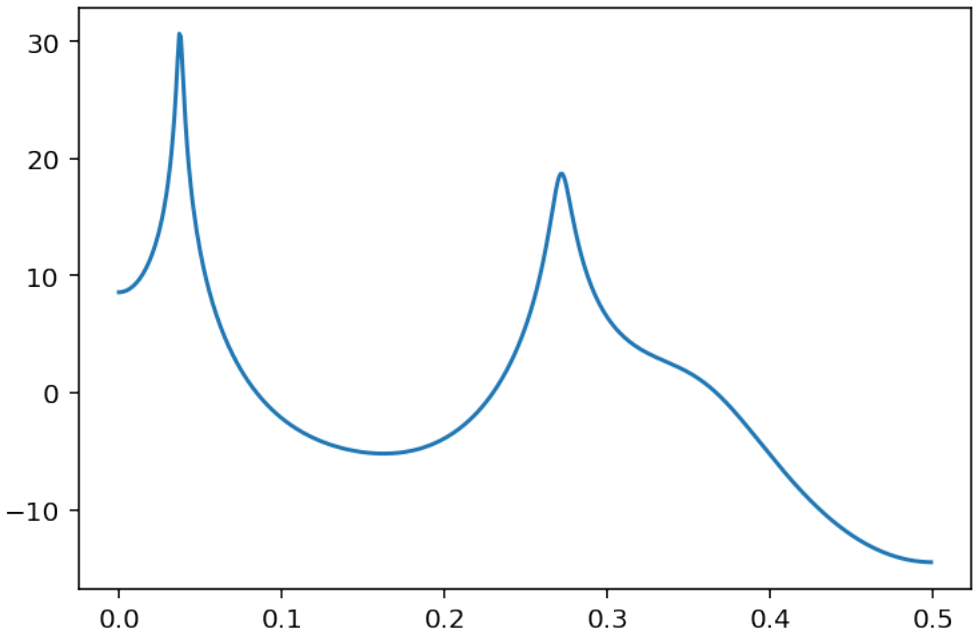


Question 10 (5 points) Record yourself speaking the same vowel sound you analyzed above. Graph the power spectral density of your recording alongside the LPC spectrum generated above.

[158]:
```
### Your Code Here
x, fs = librosa.load("he.wav")
plt.psd(x, 1024)
plt.show()

plt.plot(w,20*np.log10(h))
plt.show()
```




```
/Users/shirleyqi/opt/anaconda3/lib/python3.8/site-
packages/numpy/core/_asarray.py:102: ComplexWarning: Casting complex values to
real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



Question 11 (5 points) How does the power spectral density of your recorded signal compare to the LPC spectrum?

The shape of the two graphs are similar, with the peaks being at around the same area of the graph.

[ ]: