## Problem 1

In this section, we're gonna apply what we learned about DFT in class to classify the given chord audio files.

We're given training data of 10 chordtypes, and each type contains about 200 audio files. In part(a), we will randomly choose one file from each chordtype and set it as the model wavelet which would later on be used to determine the similarity of other unknown files.

Here's the code for the plots:

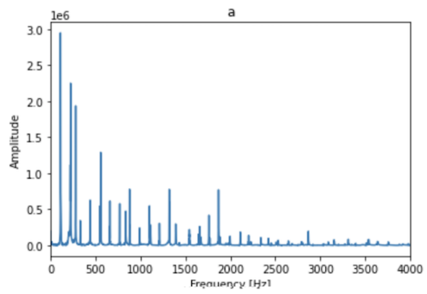
```python
import matplotlib.pyplot as plt
import random

Fs=44100
training_chord=[] #create an array to store all the wavelets


for i in range(10):
  ctype_index=np.squeeze(np.where(y==i)) #get all the index for one chordtype
  n=random.choice(ctype_index) #randomly choose one audio file from a type
  chord=X[n] #get the chord data
  training_chord.append(chord) #add to the storing array


  N=len(chord)
  X_k = np.fft.fft(chord)[0:int(N/2)] #take DFT of the chord data and take half of the signal due
      to symmetry
  X_k=np.abs(X_k)
  X_k=np.trim_zeros(X_k,'b') #get rid of the zeros at the end of the DFT


  f = Fs*np.arange(len(X_k))/N #the x-axis
  delta_f=Fs/N #the sample spacing of the x-axis (frequency resolution)
  delta_f=np.around(delta_f,2)


  title=df_train.iloc[n, 1] #get the chord name instead of the number from encoder


  print("For chord "+ title + " the frequency resolution (sample spacing) is:" + str(delta_f))
  plt.plot(f,X_k)
  plt.ylabel('Amplitude')
  plt.xlabel('Frequency [Hz]')
  plt.xlim(0,4000) #zoom in a region
  plt.title(title)
  plt.show()
```

As we encoded "y" with number values, in part(a) we first found all the indices of y for one class, then randomly choose one index, the corresponding data in X would be our wavelet data of this chordtype. Then we use the fft() function in numpy. Our frequency resolution, as well as the sampling space, is Fs/N. We also zoomed in on the region where we have a better vision of the plots.

Here's the output:

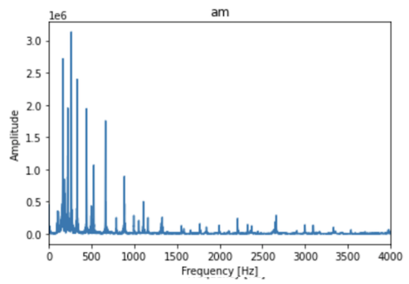
(Because the wavelets are randomly generated, if we re-run the code, we'll get different results. Here's just one possibility.)

```
For chord a the frequency resolution (sample spacing) is:0.59
```



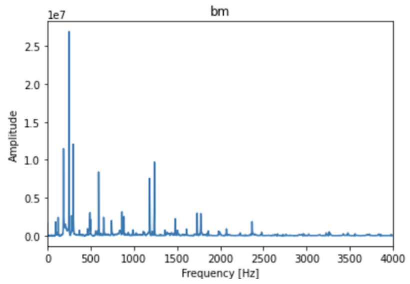
Chord a has many peaks gradually reducing as frequency increases, the distribution is quite even

```
For chord am the frequency resolution (sample spacing) is:0.43
```



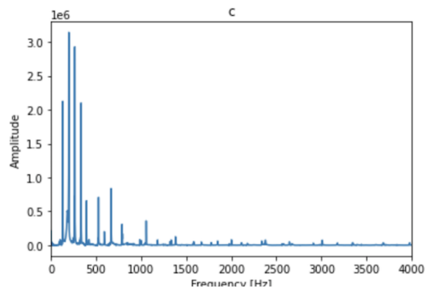
Chord am has more dense high peaks between 0-500Hz than chord a, not much peaks on higher frequencies

```
For chord bm the frequency resolution (sample spacing) is:0.81
```



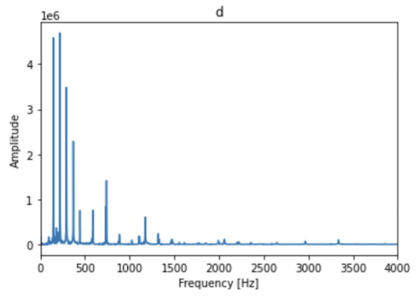
The peaks in chord b looks thin, around 250Hz appears the highest peak, and also around 1200Hz

```
For chord c the frequency resolution (sample spacing) is:0.46
```



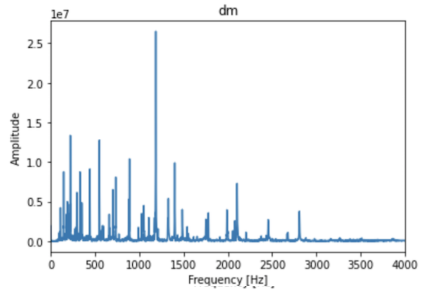
Chord c's peaks gathers within 500Hz, difernt from chord a is that there's no other significant peaks on higher frequencies while chord a still has

```
For chord d the frequency resolution (sample spacing) is:0.54
```



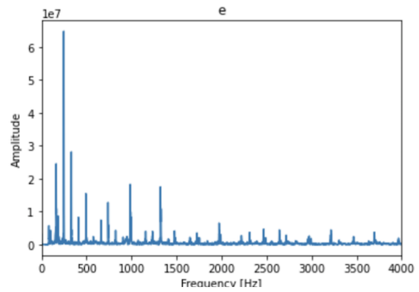
Chord d looks a bit like chord c but we can see that their peaks not not at the same frequencies

```
For chord dm the frequency resolution (sample spacing) is:0.7
```



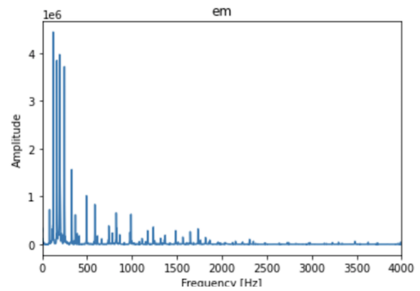
Highest peak of chord dm is at around 1200Hz, on lower frequencies there're some fairly distributed lower peaks

```
For chord e the frequency resolution (sample spacing) is:0.52
```



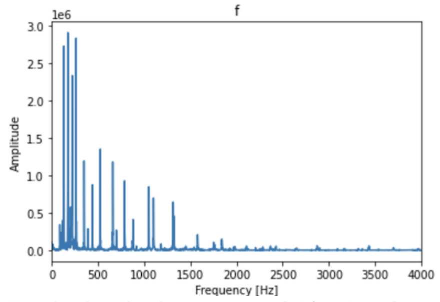
Chord e reaches an amplitude of $6*10^7, thehighestamongall, peaksatlowfrequencytoo$

```
For chord em the frequency resolution (sample spacing) is:0.58
```



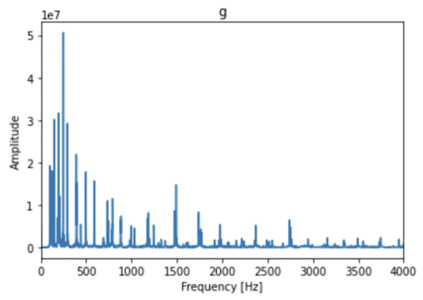
The peaks in em is narrow and mainly on low frequencies

```
For chord f the frequency resolution (sample spacing) is:0.5
```



For chord f, a bit like c, but with more lower peaks between 500-1500Hz

```
For chord g the frequency resolution (sample spacing) is:0.42
```



For chord g, we can see the distribution from the shade of the color blue, density on lower frequencies, and highest peak at around 300Hz

We see the difference between the chords, and the parameters like the peak position, the amplitude would be used to help us tell the unknown chords using the function we create in part(b).
We create a function to find the chordtype of unknown audio files using the standard wavelets.

What we're doing is, we're given an unknown signal, and we're gonna determine which one it is from the known 10 types of standard signals. To achieve this, we can use cross correlation, which is a measure of how similar two signals are at different delays. To calculate it, it's like a convolution with one of the signal flipped in time domain. After we obtain the sequence, to compare which one resonates the most, we can calculate the energy. And our prediction would be the signal that achieves the maximum energy.
Now, to implement this method, to express it in the programming language, we can explore some quicker method using our knowledge of the fourier transform. According to Parseval's Theorem, we can use the fourier transform data to get the energy of an original signal. Meanwhile, the multiplication is significantly faster than the convolution function. Thus we'll find the same frequency bins between the two signals, then create a new array storing the corresponding multiplication of the DFT, then compare the energy.

```
def find_type(signal):
  U=[] #create an array to store all the energy of the cross-correlation with diffrent wavelets

  for i in range(10):
    h=np.flip(training_chord[i]) #time reversal of the wavelets
    h=h/np.linalg.norm(h) #normalize the reversed wavelets signal
    H = np.fft.fft(h) #DFT of the wavelets
    N1=len(H)
    H_k=np.abs(H[0:int(N1/2)]) #DFT of the wavelet
    #corresponding frequencies of the DFT:
    f_H=Fs*np.arange(N1)/N1
    f_H=f_H[0:int(N1/2)]
    f_H=np.around(f_H,2) #take decimal of 2 in order to get more same frequency points in the later
         steps

    S = np.fft.fft(signal) #do the same for the input signal
    N2=len(S)
    S_k=np.abs(S[0:int(N2/2)])
    f_S=Fs*np.arange(N2)/N2
    f_S=f_S[0:int(N2/2)]
    f_S=np.around(f_S,2)


    u=[] #create an array of the multiplication of the signals in same frequencies

    same_f=np.intersect1d(f_H, f_S) #find the same frequencies of the two signals

    for i in range(len(same_f)):
      indices1=np.where(f_H==same_f[i]) #find the indices of the same frequencies
      indices2=np.where(f_S==same_f[i])
      u.append(H_k[indices1]*S_k[indices2]) #multiply the corresponding amplitude


    energy=np.linalg.norm(u)**2/len(u)
    U.append(energy)

  index=np.argmax(U) #find the max energy

  return(index)
```

Then, apply this function, we'll construct a confusion matrix for all the training data. The principle is to predict the chordtype using our method, for chordtype i, if our prediction is j, then we'll add one to the value on (i,j) in the matrix. The confusion matrix would visualize how well we are at predicting the chordtypes. If our (i,i) has the largest value, then we could say that the method obtains a fine accuracy.

```python
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

matrix_data=np.zeros([10,10], dtype=int)
for i in range(10):
  ctype_index=np.squeeze(np.where(y==i))
  for j in ctype_index:  #for all the files in one chrodtype
    signal=X[j]
    name=find_type(signal) #find the predicted chordtype using created function
    print(name)
    matrix_data[i][name] += 1

df_cm = pd.DataFrame(matrix_data, index = ['a', 'am', 'bm', 'c', 'd', 'dm', 'e', 'em', 'f', 'g'],
                  columns = ['a', 'am', 'bm', 'c', 'd', 'dm', 'e', 'em', 'f', 'g'])
fig=sn.heatmap(df_cm, annot=True)
fig.set_xlabel('Predicted');fig.set_ylabel('Actual');
fig.set_title('Confusion Matrix For Training Data');
```

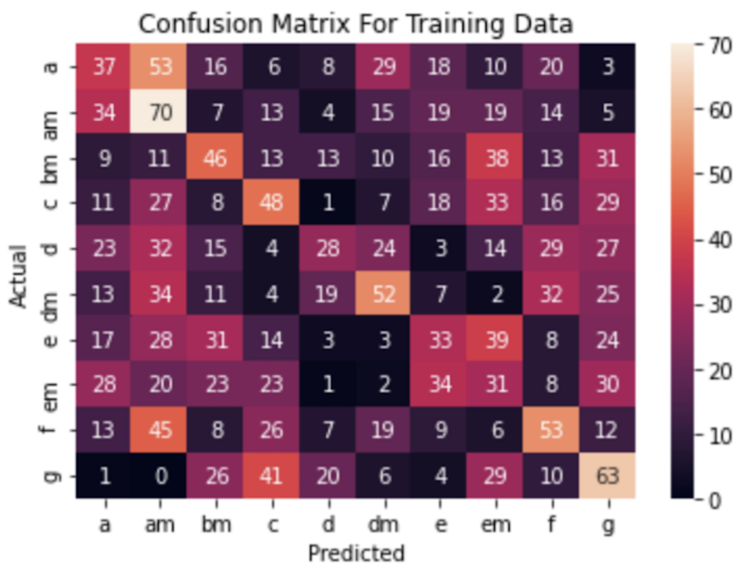
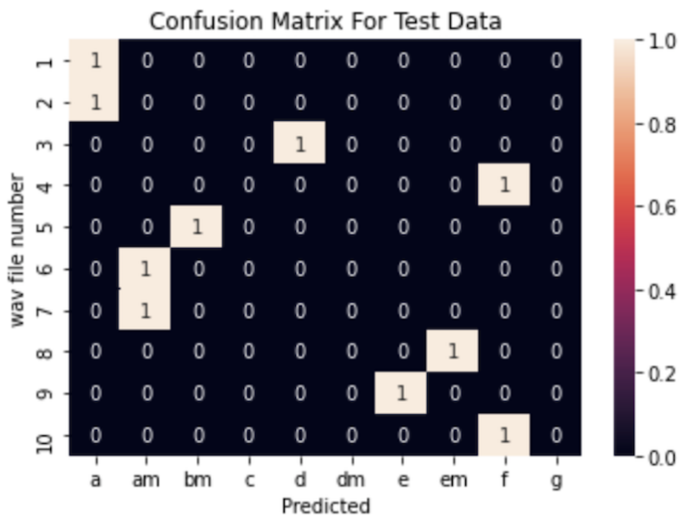


Figure 1: Traning Data Confusion Matrix

From the confusion matrix, we can see that our accuracy is fine, since the heatmap indicates larger number on lighter color, we expect to see a light y=1-x diagonal line among darker colors around. Our chord am, bm, c, dm, f, and g have great prediction result. Overall is fine.
Then we do the same to predict the testing data:

```
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

matrix_data=np.zeros([10,10], dtype=int)
for i in range(10):
  signal=X_test[i]
  name=find_type(signal)
  matrix_data[i][name] += 1


df_cm = pd.DataFrame(matrix_data, index = [1,2,3,4,5,6,7,8,9,10],
                  columns = ['a', 'am', 'bm', 'c', 'd', 'dm', 'e', 'em', 'f', 'g'])
fig=sn.heatmap(df_cm, annot=True)
fig.set_xlabel('Predicted');fig.set_ylabel('wav file number');
fig.set_title('Confusion Matrix For Test Data');
```



| | a | am | bm | c | d | dm | e | em | f | g |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Figure 2: Testing Data Confusion Matrix

To improve accuracy, we could implement more data into one chordtype to set up the model wavelet. For instance, as we are only using one set of standard wavelet that we randomly selected for now, we could generate more, like ten or a hundred sets of different standard wavelets and use the function that many times on an unknown file, sum up the results and take the average. As different a, am... could be different and in that way we'll resonate more, thus reduce the effect of the similarity between standard wavelet signals and decrease the error.