University of Technology
الجامعة التكنولوجية

Computer Science Department
قسم علوم الحاسوب

Speech Recognition Lab
تمييز الكلام- عملي

Dr. Khitam A. Salman
د. ختام عبدالنبي سلمان

cs.uotechnology.edu.iq

Lab 1:

Visualizing Audio Signals –

Reading from a File and Working on it

This is the first step in building speech recognition system as it gives an understanding of how an audio signal is structured.

We should perform sampling at a certain frequency and convert the signal into the discrete numerical form. Choosing the high frequency for sampling implies that when humans listen to the signal, they feel it as a continuous audio signal.

Example

The following example shows a stepwise approach to analyse an audio signal, using Python, which is stored in a file. The frequency of this audio signal is 44,100 HZ.

Import the necessary packages as shown here

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile
```

Now, read the stored audio file. It will return two values: the sampling frequency and the audio signal. Provide the path of the audio file where it is stored, as shown here –

```
frequency_sampling, audio_signal = wavfile.read("/Users/admin/audio_file.wav")
```

Display the parameters like sampling frequency of the audio signal, data type of signal and its duration, using the commands shown –

```
print('\nSignal shape:', audio_signal.shape)

print('Signal Datatype:', audio_signal.dtype)

print('Signal duration:', round(audio_signal.shape[0] /

float(frequency_sampling), 2), 'seconds')
```

This step involves normalizing the signal as shown below –

```
audio_signal = audio_signal / np.power(2, 15)
```

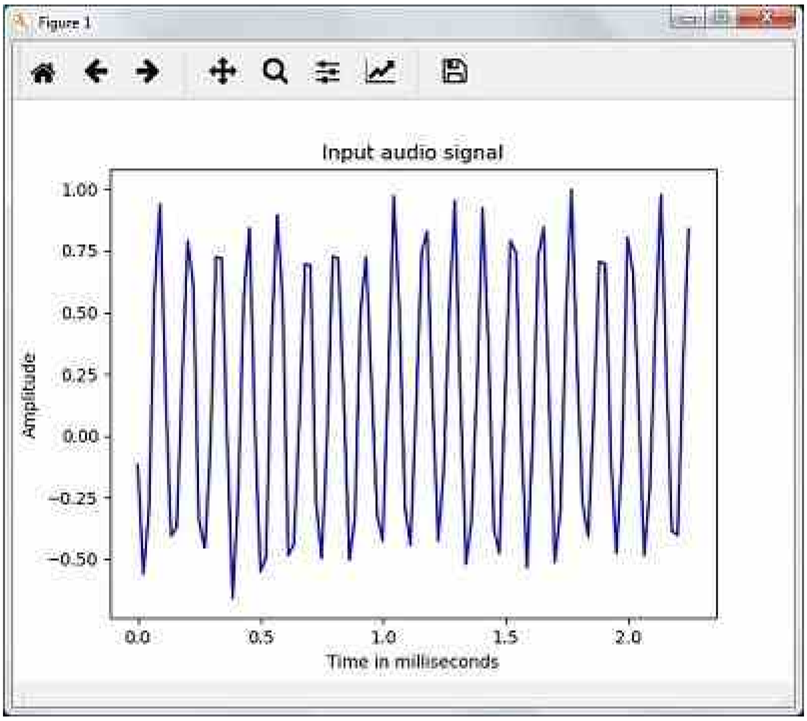
In this step, we are extracting the first 100 values from this signal to visualize. Use the following commands for this purpose −

```
audio_signal = audio_signal [:100]
time_axis = 1000 * np.arange(0, len(audio_signal), 1) / float(frequency_sampling)
```

Now, visualize the signal using the commands given below −

```
plt.plot(time_axis, audio_signal, color='blue')

plt.xlabel('Time (milliseconds)')

plt.ylabel('Amplitude')

plt.title('Input audio signal')

plt.show()
```

You would be able to see an output graph and data extracted for the above audio signal as shown in the image here



Speech Recognition Lab

Lab 2:

Characterizing the Audio Signal: Transforming From Time Domain to Frequency Domain.

Lecturer: Dr. Asia Ali

Tip 1: Characterizing an audio signal involves converting the time domain signal into frequency domain.

Tip 2: understanding its frequency components. This is an important step because it gives a lot of information about the signal.

Tip 3: You can use a mathematical tool like Fourier Transform to perform this transformation.

Example

The following example shows, step-by-step, how to characterize the signal, using Python, which is stored in a file. Note that here we are using Fourier Transform mathematical tool to convert it into frequency domain.

Import the necessary packages, as shown here –

```
import numpy as np

import matplotlib.pyplot as plt
from scipy.io import wavfile
```

Now, read the stored audio file. It will return two values: the sampling frequency and the the audio signal. Provide the path of the audio file where it is stored as shown in the command here

```
frequency_sampling, audio_signal = wavfile.read("/Users/admin/sample.wav")
```

In this step, we will display the parameters like sampling frequency of the audio signal, data type of signal and its duration, using the commands given below –

```
print('\nSignal shape:', audio_signal.shape)

print('Signal Datatype:', audio_signal.dtype)

print('Signal duration:', round(audio_signal.shape[0] /

float(frequency_sampling), 2), 'seconds')
```

In this step, we need to normalize the signal, as shown in the following command –

```
audio_signal = audio_signal / np.power(2, 15)
```

This step involves extracting the length and half length of the signal. Use the following commands for this purpose .

Rounded the values using np.ceil() function.

```
length_signal = len(audio_signal)
half_length = np.ceil((length_signal + 1) / 2.0).astype(np.int)
```

Now, we need to apply mathematics tools for transforming from time domain into frequency domain. Here we are using the Fourier Transform.

This function (np.fft.fft) computes the *N*-dimensional discrete Fourier Transform over dimensional array by means of the Fast Fourier Transform (FFT).

```
signal_frequency = np.fft.fft(audio_signal)
```

Now, do the normalization of frequency domain signal and square it –

```
signal_frequency = abs(signal_frequency[0:half_length]) / length_signal

signal_frequency **= 2
```

Next, extract the length and half length of the frequency transformed signal

```
len_fts = len(signal_frequency)
```

Note that the Fourier transformed signal must be adjusted for even as well as odd case.

```
if length_signal % 2:

   signal_frequency[1:len_fts] *= 2

else:

   signal_frequency[1:len_fts-1] *= 2
```

an even case is one that's "symmetric" when you reflect it about the y-axis; that is its right half looks exactly like its left.

an odd case is one that is "antisymmetric" when reflected about the vertical, its right half is an "upside-down" copy of its left half.

Now, extract the power, to get the power values we need to compute the logarithmic value for each value in (signal_frequency) to change them to linear values.

```
signal_power = 10 * np.log10(signal_frequency)
```

Signal_power : A location into which the result is stored. a freshly-allocated array is returned

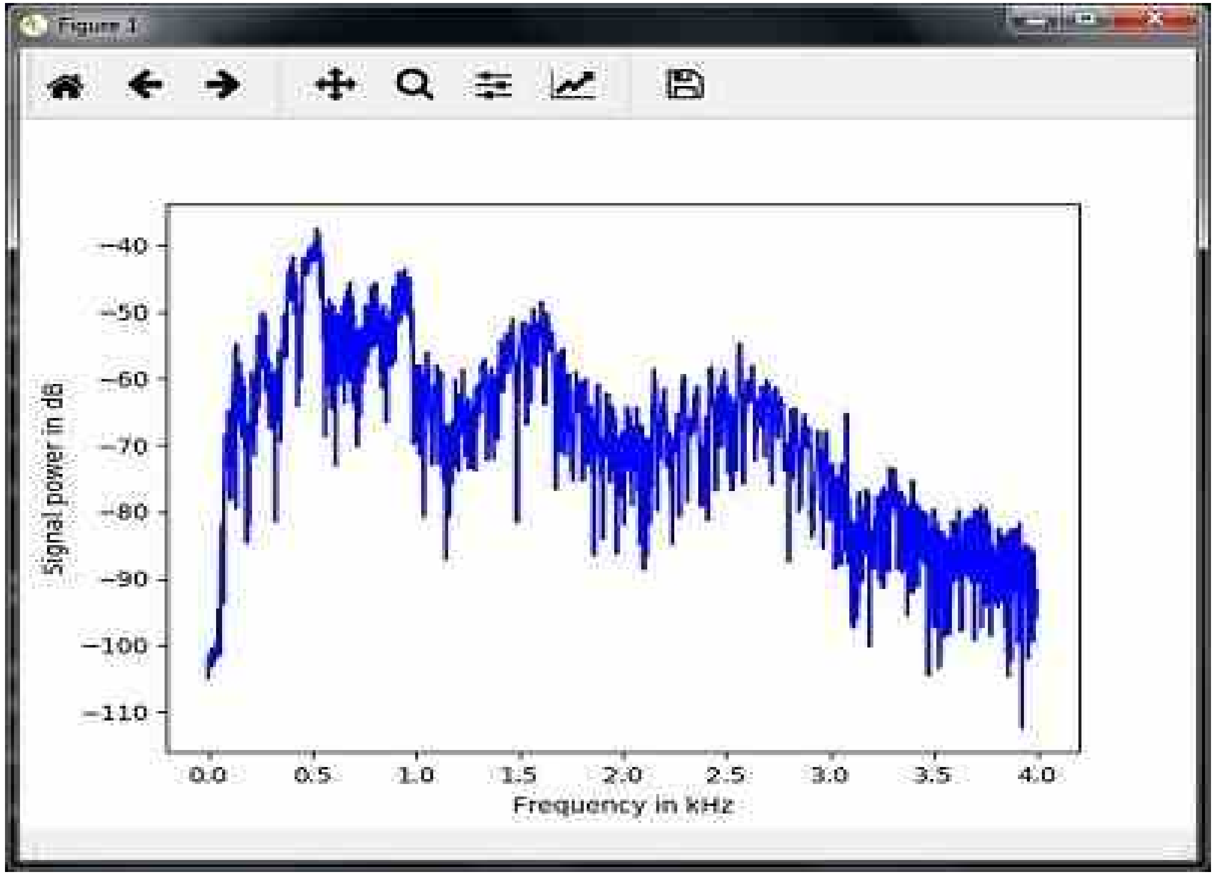
Adjust the frequency in kHz for X-axis –

```
x_axis = np.arange(0, len_half, 1) * (frequency_sampling / length_signal) / 
1000.0
```

Now, visualize the characterization of signal as follows –

```
plt.figure()

plt.plot(x_axis, signal_power, color='black')

plt.xlabel('Frequency (kHz)')

plt.ylabel('Signal power (dB)')

plt.show()
```

You can observe the output graph of the above code as shown in the image below −



Speech Recognition Lab

Lab 3:

Lab Assignment.

Lecturer: Dr. Asia Ali

Q1- Why do we need python language to build speech recognitions system?

Q2- Why we should perform sampling at a certain frequency ?

Q3- What is the sampling rate? What is the standard value that we can use in our programs?

Q4- Write the code that we need to open and read an audio file.

Q5- We are extracting the first 100 values from an audio signal to visualize them in lab1 program. Can we change the command to read more or less values?

Q6- What is the command we have used to plot the audio signal, which library we have used?

Q7- Which function we have used in lab2 to change from time domain to the frequency domain?

Q8- What is the function of (np.Fft.Fft)? And from which library we have import it?

Q9- The fourier transformed signal must be adjusted for even as well as odd case, show the commands that can do that.

Q10- How can we adjust the frequency in khz for x-axis ?

Speech Recognition Lab

Lab 4:

Feature Extraction methods:

Zero Crossing Rate.

Lecturer: Dr. Asia Ali

Zero Crossing Rate Method.

- The input audio stream is broken into Number of frames . Each of specific size.
- We calculate ZCR for number of frams.
- We need to calculate the standard and mean of ZCR for all the frams.
- To find the correlation-based feature we used only the standard deviation and mean.

- import numpy
- import scipy.io.wavfile

These are the imported library files that we need in this program

```
def main():
  frameSizeInMs = 0.01      # the file is small so we use 0.01
  frequency         = 44100  # Frequency of the input data(constant)
  numSamplesPerFrame = int(frequency * frameSizeInMs)

  data       = scipy.io.wavfile.read( "d:\\speech\\download.wav" )       The path of the wave file in your drive
 Zer_Fst_list=list(data[1])            #sampling rate data[0], data[1]

  chunkedData = chunks(zer_fst_list, numSamplesPerFrame)  #slicing the data
  List_ chunkedData = list(chunkedData )                               #organize them in separated lists
  zcr = rateSampleByCrossingRate(List_ chunkedData )           # call the function rateSampleByCrossingRate
  print("Standard deviation of ZCR = %f\n", zcr)

if zcr >= 0.05:
   print("Standard deviation of ZCR suggests that the sample contains speech")



main() # the program starts from here
```

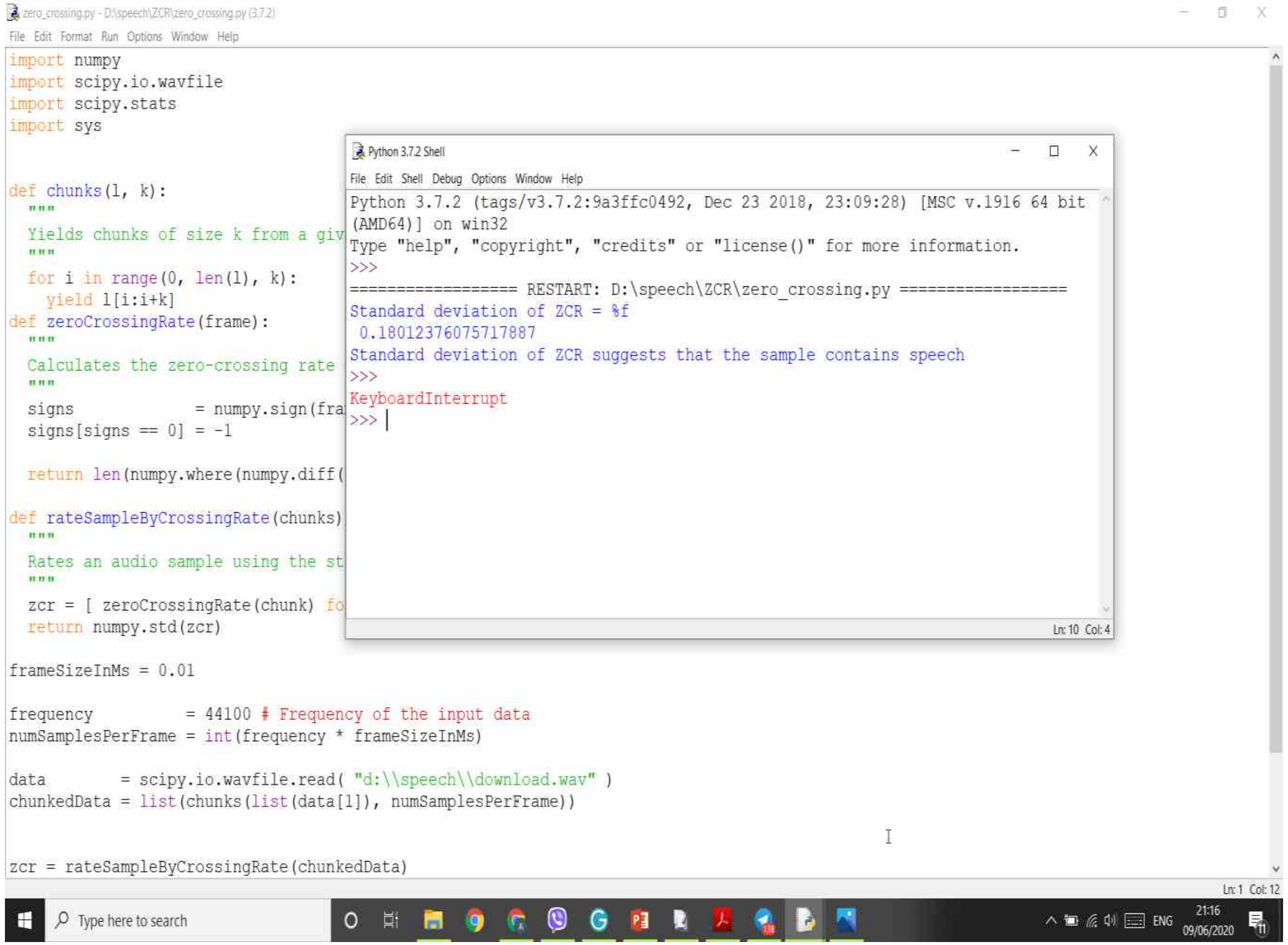
```python
def chunks(l, k):
  """
  Yields(return only a chunks of size k from a given list. it is called slicing
  """
  for i in range(0, len(l), k):
    yield l[i:i+k]
```

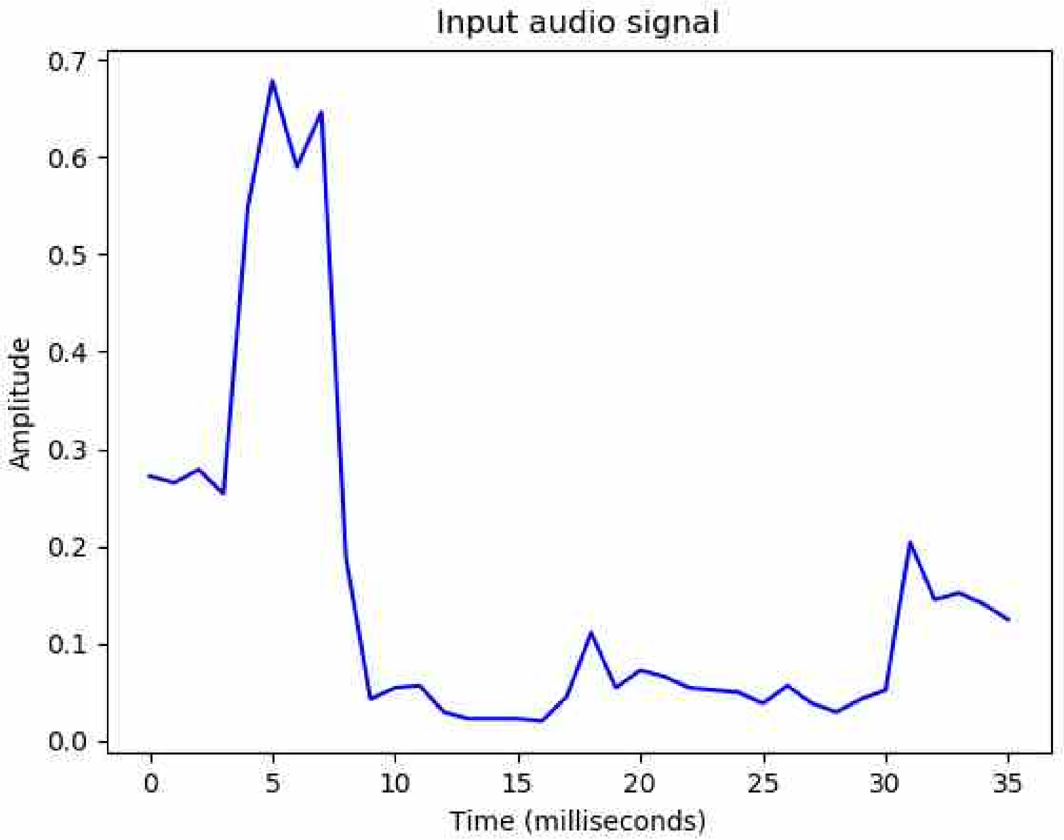
#ZERO CROSSING : number of times unvoiced speech crosses the zero line is significantly higher than that of voiced speech

```
def rateSampleByCrossingRate(chunks):
  """
  Rates an audio sample using the standard deviation of its zero-crossing rate.
  """
zcr=list()
  i=0
    for chunk in chunks:
      zcr.append(zeroCrossingRate(chunk,i))
      print(zcr)
      i+=1
      z=numpy.std(zcr , axis=0)
  print (z)
return z
```

We need to compute the standard deviation and mean of zcr all the frames

```
def zeroCrossingRate(frame):
  """Calculates the zero-crossing rate of an audio frame.  """
 # numpy.sign =Returns an element-wise indication of the sign of a number. The sign function returns -1 if x < 0,
0 if x==0, 1 if x > 0. nan is returned for nan inputs. #using already existed function in the numpy
  signs   = numpy.sign(frame)
# numpy.diff:The first difference is given by signs[n] = signs[n+1] - signs[n] a long the given
axis, higher differences are calculated by using diff recursively.
  aa1=numpy.diff(signs)
#where in the aa1 array we have [0] zero
  aa=numpy.where(aa1)[0])
  print("aa")
  print(aa)
  len1=len(aa)/len(frame)
  print ("len",len1)
  return len1
```





The output of the ZCR program: the signal and the value of the ZCR method

Speech Recognition Lab

Lab 5:

Feature Extraction methods:

Short Time Energy (STE)

Lecturer: Dr. Asia Ali

Short Time Energy Method (STE).

- The input audio stream is broken into a number of frames . Each of specific size.
- Rates an audio sample using the coefficient of variation of its short-term energy.
- Speech tends to have higher values here than non-speech.
- Some experimentation with a few speech/music files shows that a threshold of 1.0.

- import numpy
- import scipy.io.wavfile
- import scipy.stats
- import sys
- import matplotlib.pyplot as plt

These are the imported library files that we need in this program

```
def main():
  frameSizeInMs = 0.001     # the file is  big we took a small frame size= 0.001
  frequency         = 44100  # Frequency of the input data(constant)
  numSamplesPerFrame = int(frequency * frameSizeInMs)

  data       = scipy.io.wavfile.read( "d:\\speech\\FDHH_Sa.wav" )
 Zer_Fst_list=list(data[1])              #sampling rate data[0], data[1]

  chunkedData = chunks(zer_fst_list, numSamplesPerFrame)  #slicing the data
  List_ chunkedData = list(chunkedData )                      #organize them in separated lists
  variation = rateSampleByVariation(chunkedData)

print("Coefficient of variation  = %f\n",variation)

if variation >= 1.0:
  print("the value of variation means that the sample contains speech")


main() # the program starts from here
```

The path of the wave file in your drive

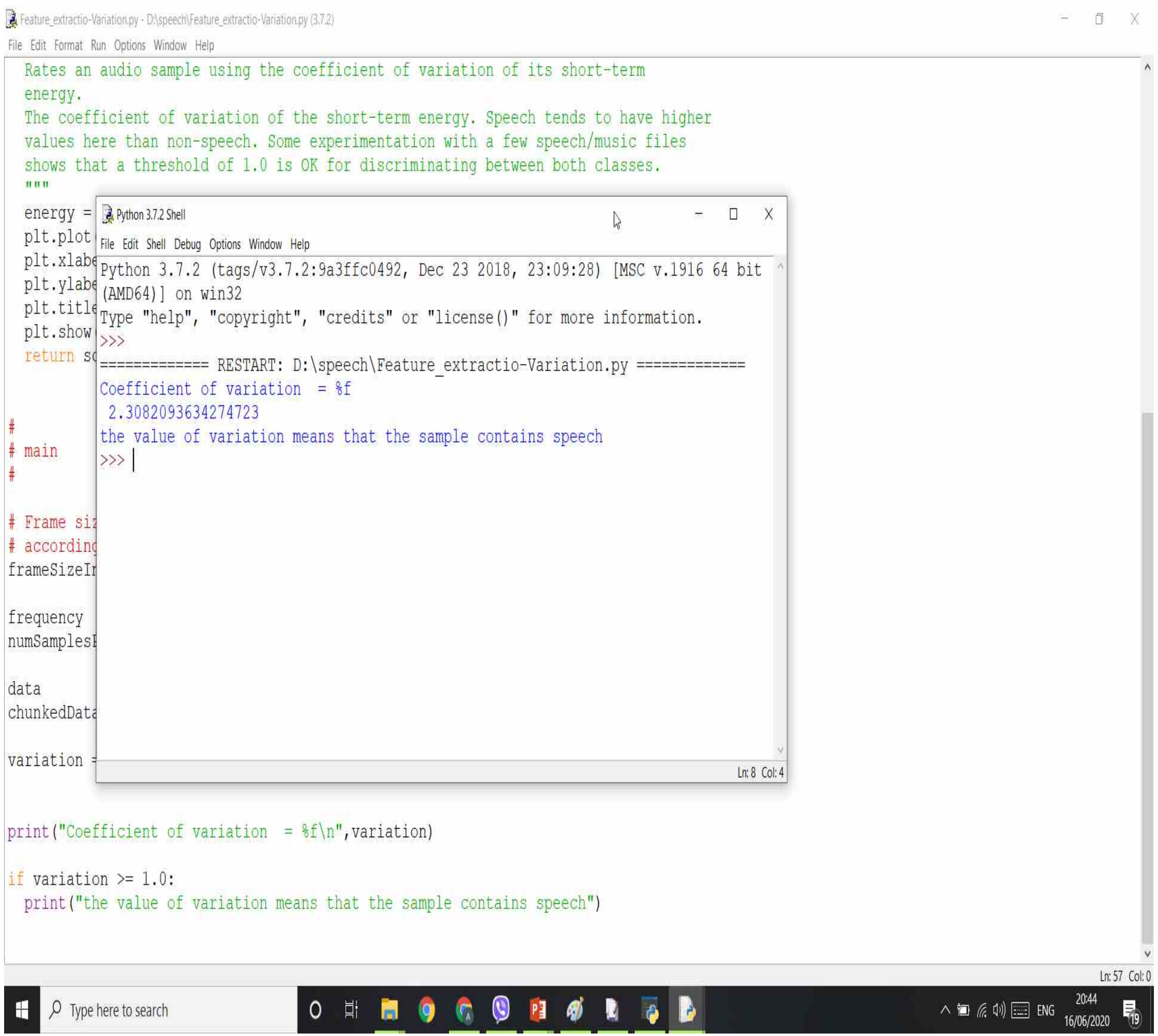
```python
def chunks(l, k):
  """
  Yields(return only a chunks of size k from a given list. it is called slicing
  """
  for i in range(0, len(l), k):
    yield l[i:i+k]
```

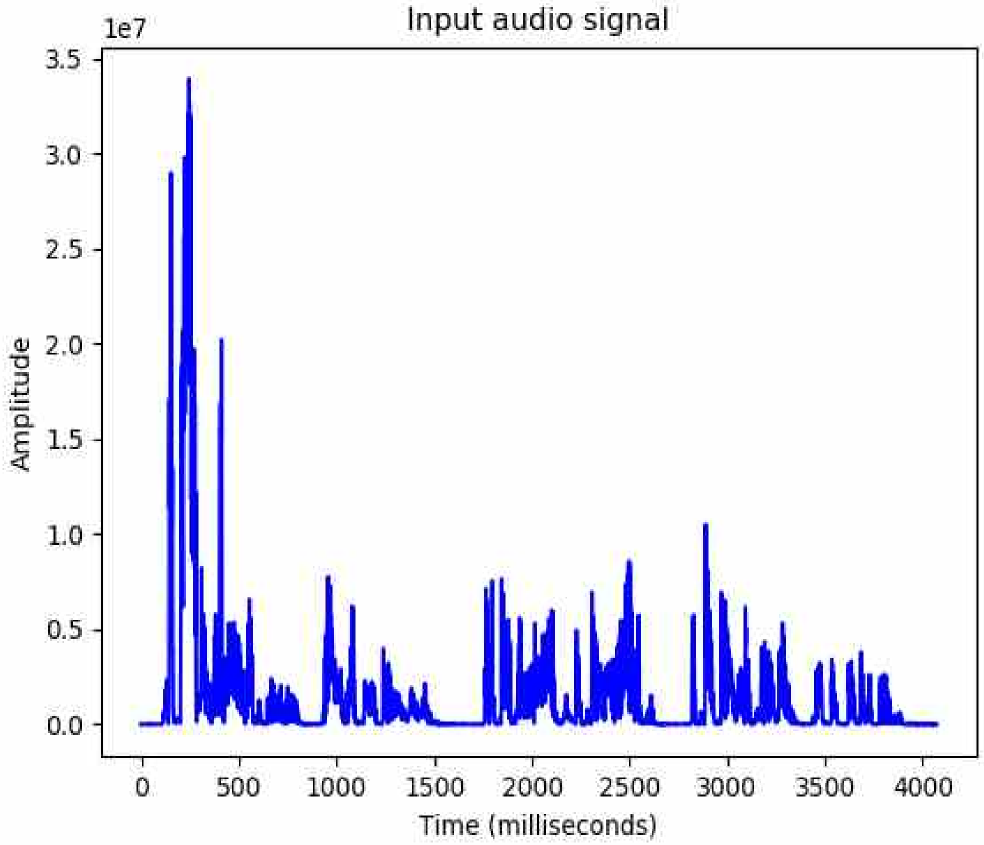
```
def shortTermEnergy(frame):
  """
  Calculates the short-term energy of an audio frame. The energy value is
  normalized using the length of the frame to make it independent of said
  quantity.
  """
  return sum( [ abs(x)**2 for x in frame ] ) / len(frame)
```

```
def rateSampleByVariation(chunks):
  """
  Rates an audio sample using the coefficient of variation of its short-term energy.
  The coefficient of variation of the short-term energy. Speech tends to have higher   values here than non-speech. Some experimentation with a few speech/music files  shows that a threshold of 1.0 is OK for discriminating between both classes.
  ""“

  energy = [ shortTermEnergy(chunk) for chunk in chunks ]
  plt.plot( energy, color='blue')
  plt.xlabel('Time (milliseconds)')
  plt.ylabel('Amplitude')
  plt.title('Input audio signal')
  plt.show()

  return scipy.stats.variation(energy)
```

variation

- Work out the Mean (the simple average of the numbers)
- Then for each number: subtract the Mean and square the result (the *squared difference*).
- Then work out the average of those squared differences.





The output of the STE program: the signal and the value of the STE method

Speech Recognition Lab

Lab 6:

Feature Extraction methods:

Entropy Of Energy

Lecturer: Dr. Asia Ali

Entropy Of Energy

- The input audio stream is broken into Number of frames . Each of specific size.
- Computing the minimum entropy of energy (Entropy is a measure of the energy spread in the system).
- This is where speech usually has lower values than music.
- A threshold of 2.5 to decide whether an audio file contains speech.

consider an audio file with a computed total energy (use the same function in lab5) and a fixed total number of frames N. Each frame of the file can be in one out of M discrete energy states m {M1,M2; :::;Mg} and the particles can exchange energy without loss.

- import numpy
- import scipy.io.wavfile
- import scipy.stats
- import sys
- import matplotlib.pyplot as plt

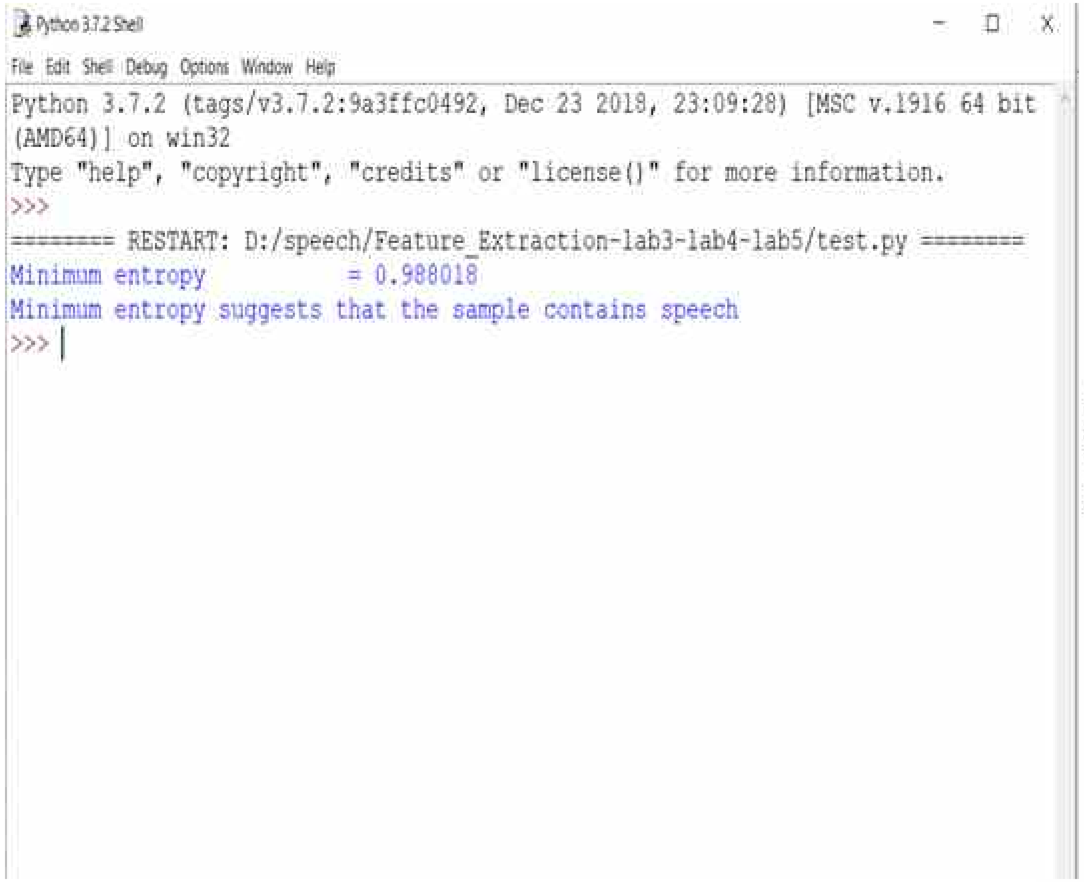
These are the imported library files that we need in this program

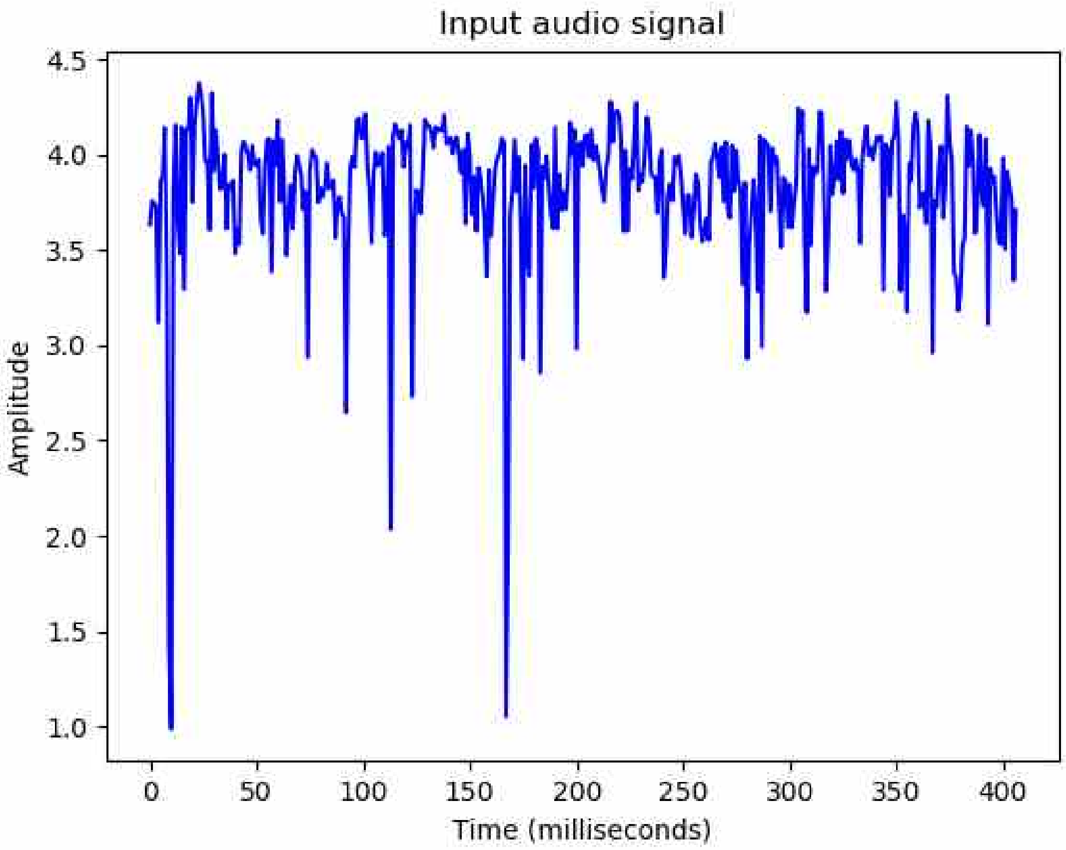
```
def main():
  frameSizeInMs = 0.01      # the  small frame size
  frequency         = 44100  # Frequency of the input data(constant)
  numSamplesPerFrame = int(frequency * frameSizeInMs)

  data        = scipy.io.wavfile.read( "d:\\speech\\FDHH_Sa.wav" )        The path of the wave file in your drive
 Zer_Fst_list=list(data[1])              #sampling rate data[0], data[1]

  chunkedData = chunks(zer_fst_list, numSamplesPerFrame)  #slicing the data
  List_ chunkedData = list(chunkedData )                                   #organize them in separated lists
   entropy   = rateSampleByEntropy(List_ chunkedData )

print( "Minimum entropy          = %f" %  entropy )

if entropy < 2.5:
  print("Minimum entropy suggests that the sample contains speech")

main() # the program starts from here
```

```
def chunks(l, k):
  """
  Yields(return only a chunks of size k from a given list. it is called slicing
  """
  for i in range(0, len(l), k):
    yield l[i:i+k]
```

```
def entropyOfEnergy(frame, numSubFrames):
  """
  Calculates the entropy of energy of an audio frame. For this, the frame is
  partitioned into a number of sub-frames.
  """
  lenSubFrame = int(numpy.floor(len(frame) / numSubFrames))
  shortFrames = list(chunks(frame, lenSubFrame))
  energy      = [ shortTermEnergy(s) for s in shortFrames ]
  totalEnergy = sum(energy)  # The items of the iterable should be numbers. start (optional) - this
                             # value is added to the sum of items of the iterable.
  energy      = [ e / totalEnergy for e in energy ]

  entropy = 0.0
  for e in energy:   #energy is an array
    if e != 0:
      entropy = entropy - e * numpy.log2(e)

  return entropy
```

```
def rateSampleByEntropy(chunks):
  """
  Rates an audio sample using its minimum entropy.
  """
  entropy = [ entropyOfEnergy(chunk, 20) for chunk in chunks ]
  return numpy.min(entropy)
```





The output of the Entropy: the signal and the value of the minimum entropy.