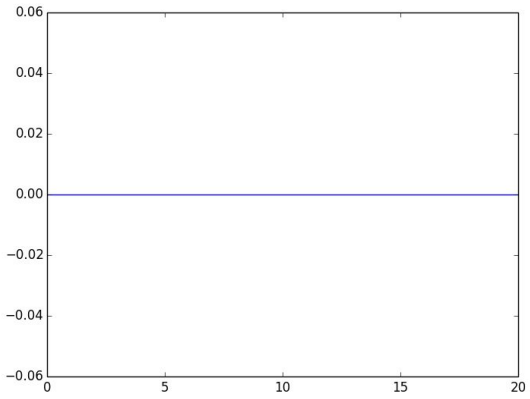
CSC 475 assignment2

V00825808 XinYue Liu

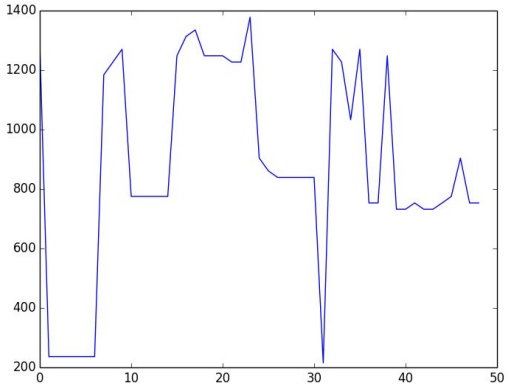
Q1.1

```
import numpy as np
import scipy.io.wavfile as sc
import matplotlib.pyplot as plt
from scipy.fftpack import fft
import math
import wave

rate, a = sc.read("output.wav")
rate, b = sc.read("Ah.wav")
rate, c = sc.read("qbhexamples.wav")

types = {1: np.int8, 2: np.int16, 4: np.int32}

s = 2048
sr = 44100
window = np.hamming(2048)

wave_a = wave.open('output.wav')
#nframe_a = wave_a.getnframes()
a_nchannels, a_sampwidth, a_framerate, a_nframes, a_comptype, a_compname = wave_a.getparams()

wave_b = wave.open('Ah.wav')
#nframe_a = wave_a.getnframes()
b_nchannels, b_sampwidth, b_framerate, b_nframes, b_comptype, b_compname = wave_b.getparams()

wave_c = wave.open('qbhexamples.wav')
#nframe_a = wave_a.getnframes()
c_nchannels, c_sampwidth, c_framerate, c_nframes, c_comptype, c_compname = wave_c.getparams()

fft_aout = abs(fft(a))
fft_bout = abs(fft(b))
fft_cout = abs(fft(c))

freqa = []
freqb = []
freqc = []

for i in range(a_nframes // s):
    content_a = wave_a.readframes(s)
    sam = np.fromstring(content_a, dtype=types[a_sampwidth])[0::a_nchannels]
    sam = sam * window
    fft_aout = abs(fft(sam,2048))
    fund_freq_a = sr * np.argmax(fft_aout) / len(fft_aout)
    plt.figure()
    freqa.append(fund_freq_a)
    plt.plot(freqa)
    plt.savefig("nq1_1A.png")

for i in range(b_nframes // s):
    content_b = wave_b.readframes(s)
    sbm = np.fromstring(content_b, dtype=types[b_sampwidth])[0::b_nchannels]
    sbm = sbm * window
    fft_bout = abs(fft(sbm,2048))
    fund_freq_b = sr * np.argmax(fft_bout) / len(fft_bout)
    freqb.append(fund_freq_b)
    plt.figure()
    plt.plot(freqb)
    plt.savefig("nq1_1B.png")

for i in range(c_nframes // s):
    content_c = wave_c.readframes(s)
    scm = np.fromstring(content_c, dtype=types[c_sampwidth])[0::c_nchannels]
    scm = scm * window
    fft_cout = abs(fft(scm,2048))
    fund_freq_c = sr * np.argmax(fft_cout) / len(fft_cout)
    freqc.append(fund_freq_c)
    plt.figure()
    plt.plot(freqc)
    plt.savefig("nq1_1C.png")
```

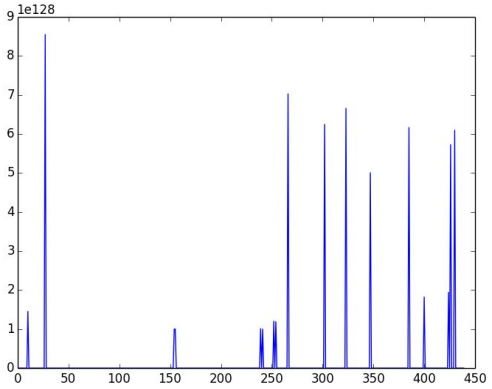
The little melody you created in assignment 1:



The same melody hummed by you and recorded in Audacity. Use a single vowel like Ah for your singing:



The qbhexamples.wav available under Resources in Connex:



Q1.2

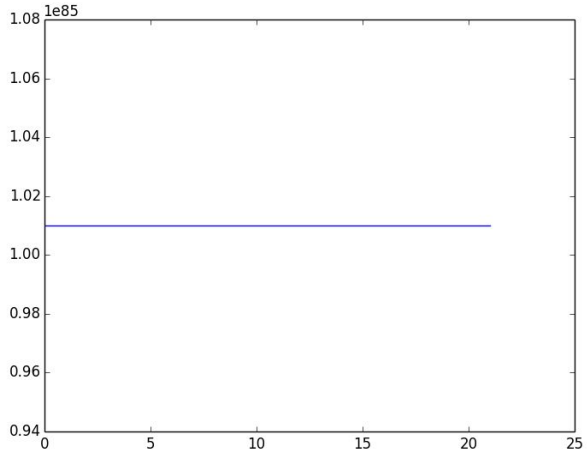
```
Series {
    inSamples = 2048
    -> input: SoundFileSource { filename = "output.wav" }
    -> Windowing { size = 2048 }

    -> AutoCorrelation
    -> Peaker
    -> MaxArgMax

    -> Transposer
    -> selection: Selector { disable = 0 }

    -> sink: CsvSink { filename = "q1.2.1.csv" }
    + done = (input/hasData == false)
}
```

```
Series {
    inSamples = 2048
    -> input: SoundFileSource { filename = "Ah.wav" }
    -> Windowing { size = 2048 }

    -> AutoCorrelation
    -> Peaker
    -> MaxArgMax

    -> Transposer
    -> selection: Selector { disable = 0 }

    -> sink: CsvSink { filename = "q1.2.2.csv" }
    + done = (input/hasData == false)
}
```

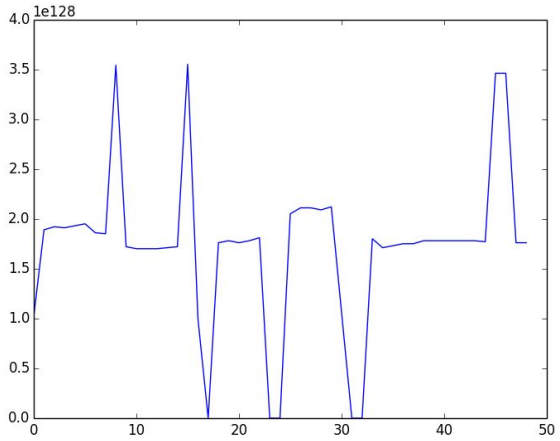
```
Series {
    inSamples = 2048
    -> input: SoundFileSource { filename = "qbhexamples.wav" }
    -> Windowing { size = 2048 }

    -> AutoCorrelation
    -> Peaker
    -> MaxArgMax

    -> Transposer
    -> selection: Selector { disable = 0 }

    -> sink: CsvSink { filename = "q1.2.3.csv" }
    + done = (input/hasData == false)
}
```

```
import csv
import matplotlib
matplotlib.use('AGG')
import matplotlib.pyplot as plt

y_data1 = []
y_data2 = []
y_data3 = []
with open('q1.2.1.csv') as csvDataFile:
        csvReader = csv.reader(csvDataFile)
        for row in csvReader:
                y_data1.append(row[0])

with open('q1.2.2.csv') as csvDataFile:
        csvReader = csv.reader(csvDataFile)
        for row in csvReader:
                y_data2.append(row[0])

with open('q1.2.3.csv') as csvDataFile:
        csvReader = csv.reader(csvDataFile)
        for row in csvReader:
                y_data3.append(row[0])

y_data_result1=[]
y_data_result2=[]
y_data_result3=[]
for i in y_data1 :
        y_data_result1.append(i * (44100/1024)) # Sampling Rate / FFT Size
for i in y_data2 :
        y_data_result2.append(i * (44100/1024)) # Sampling Rate / FFT Size
for i in y_data3 :
        y_data_result3.append(i * (44100/1024)) # Sampling Rate / FFT Size


plt.figure()
plt.plot(range(0, len(y_data1)), y_data_result1)
plt.savefig('q1_2_1.png')

plt.figure()
plt.plot(range(0, len(y_data2)), y_data_result2)
plt.savefig('q1_2_2.png')

plt.figure()
plt.plot(range(0, len(y_data3)), y_data_result3)
plt.savefig('q1_2_3.png')
```

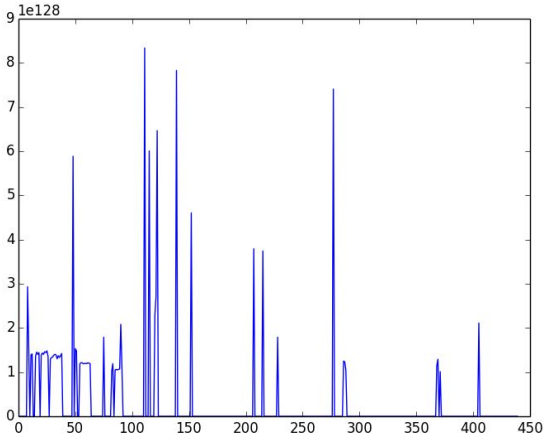
The little melody you created in assignment 1:



The same melody hummed by you and recorded in Audacity. Use a single vowel like Ah for your singing:



The qbhexamples.wav available under Resources in Connex:



Q1.3

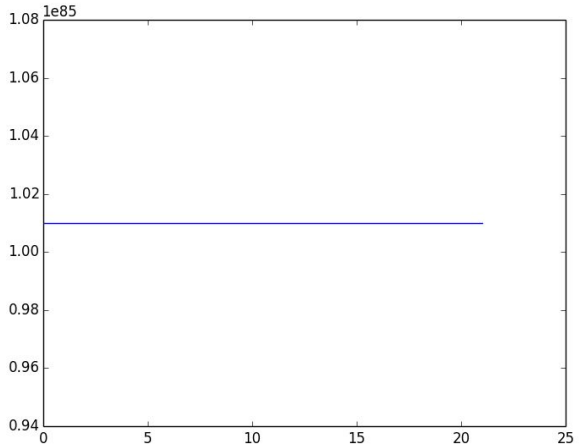
```
Series {
    inSamples = 2048
    -> input: SoundFileSource { filename = "output.wav" }
    -> Windowing { size = 2048 }
    -> Fanout {

        -> Series {
            -> Spectrum
            -> PowerSpectrum { spectrumType = "magnitude" }
            -> Transposer
            -> max: MaxArgMax
            -> Transposer
            -> selection_1: Selector { disable = 0 }
        }
        -> Series {
            -> AutoCorrelation
            -> Peaker
            -> MaxArgMax
            -> Transposer
            -> selection_2: Selector { disable = 0 }
        }
    }
    -> summer: Sum
    -> sink: CsvSink { filename = "q1.3.1.csv" }
    + done = (input/hasData == false)
}
```

```
Series {
    inSamples = 2048
    -> input: SoundFileSource { filename = "Ah.wav" }
    -> Windowing { size = 2048 }
    -> Fanout {

        -> Series {
            -> Spectrum
            -> PowerSpectrum { spectrumType = "magnitude" }
            -> Transposer
            -> max: MaxArgMax
            -> Transposer
            -> selection_1: Selector { disable = 0 }
        }
        -> Series {
            -> AutoCorrelation
            -> Peaker
            -> MaxArgMax
            -> Transposer
            -> selection_2: Selector { disable = 0 }
        }
    }
    -> summer: Sum
    -> sink: CsvSink { filename = "q1.3.2.csv" }
    + done = (input/hasData == false)
}
```

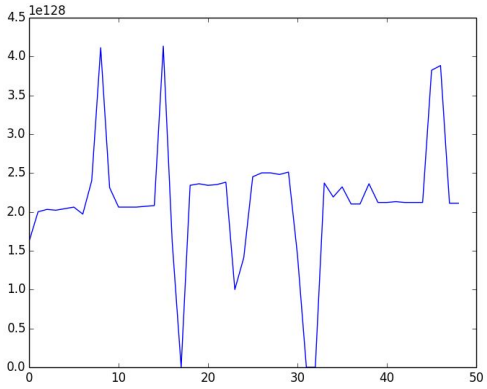
```
Series {
    inSamples = 2048
    -> input: SoundFileSource { filename = "qbhexamples.wav" }
    -> Windowing { size = 2048 }
    -> Fanout {

        -> Series {
            -> Spectrum
            -> PowerSpectrum { spectrumType = "magnitude" }
            -> Transposer
            -> max: MaxArgMax
            -> Transposer
            -> selection_1: Selector { disable = 0 }
        }
        -> Series {
            -> AutoCorrelation
            -> Peaker
            -> MaxArgMax
            -> Transposer
            -> selection_2: Selector { disable = 0 }
        }
    }
    -> summer: Sum
    -> sink: CsvSink { filename = "q1.3.3.csv" }
    + done = (input/hasData == false)
}
```

```
import csv
import matplotlib
matplotlib.use('AGG')
import matplotlib.pyplot as plt

y_data1 = []
y_data2 = []
y_data3 = []
with open('q1.3.1.csv') as csvDataFile:
        csvReader = csv.reader(csvDataFile)
        for row in csvReader:
                y_data1.append(row[0])

with open('q1.3.2.csv') as csvDataFile:
        csvReader = csv.reader(csvDataFile)
        for row in csvReader:
                y_data2.append(row[0])

with open('q1.3.3.csv') as csvDataFile:
        csvReader = csv.reader(csvDataFile)
        for row in csvReader:
                y_data3.append(row[0])

y_data_result1=[]
y_data_result2=[]
y_data_result3=[]
for i in y_data1 :
        y_data_result1.append(i * (44100/1024)) # Sampling Rate / FFT Size
for i in y_data2 :
        y_data_result2.append(i * (44100/1024)) # Sampling Rate / FFT Size
for i in y_data3 :
        y_data_result3.append(i * (44100/1024)) # Sampling Rate / FFT Size


plt.figure()
plt.plot(range(0, len(y_data1)), y_data_result1)
plt.savefig('q1_3_1.png')

plt.figure()
plt.plot(range(0, len(y_data2)), y_data_result2)
plt.savefig('q1_3_2.png')

plt.figure()
plt.plot(range(0, len(y_data3)), y_data_result3)
plt.savefig('q1_3_3.png')
```

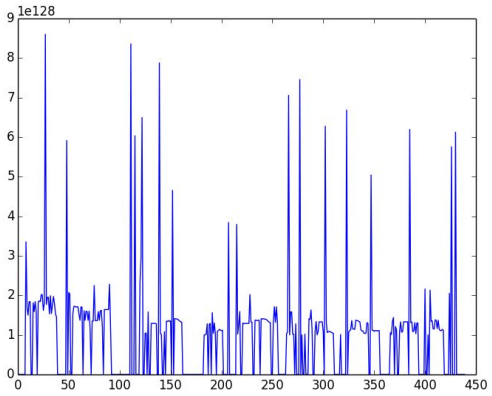
The little melody you created in assignment 1:



The same melody hummed by you and recorded in Audacity. Use a single vowel like Ah for your singing:



The qbhexamples.wav available under Resources in Connex:



I think the AutoCorrelation approach is better. The DFT approach may cannot detect the low frequency.

Q2.1

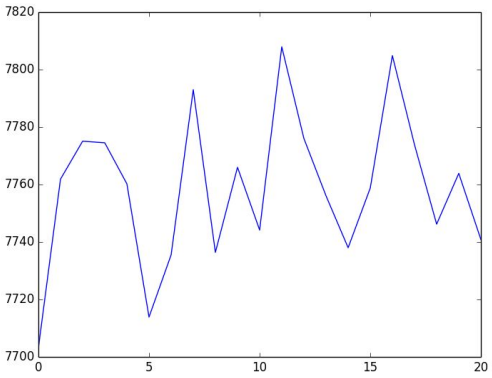
same for three inputs. I just changed the name of input file for three output.

```
import numpy as np
import scipy.io.wavfile as sc
import scipy.fftpack as fft
import matplotlib.pyplot as plt

def spectral_centroid(x, samplerate=44100):
    magnitudes = np.abs(np.fft.rfft(x))
    length = len(x)
    freqs = np.abs(np.fft.fftfreq(length, 1.0/samplerate)[:length//2+1])
    return np.sum(magnitudes*freqs) / np.sum(magnitudes)
sr = 44100
rate, data = sc.read('qbhexamples.wav')
count = 0
maxN = []
chunkF = []
nframe = len(data)
while count < nframe / 2048:
        for x in range(2048):
                if x + count * 2048 < nframe:
                        if data[x + count * 2048] < 0:
                                chunkF.append(0)
                        else:
                                chunkF.append(data[x + count * 2048])
        maxN.append(spectral_centroid(chunkF, sr))
        count += 1
        del chunkF[:]

plt.figure()
plt.plot(range(0, len(maxN)), maxN)
plt.savefig('q2_1_3.png')
```

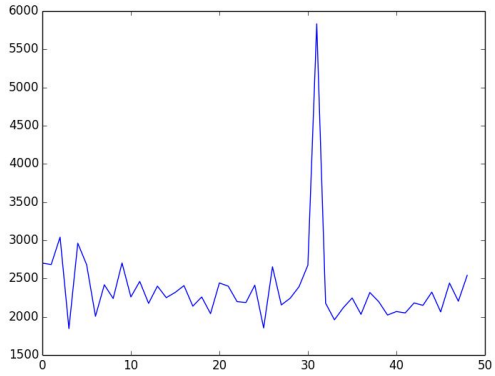
```
def generate_sin(freq, duration, srate=44100.0, amp=1.0,phase=0):
    t = np.linspace(0,duration,int(srate*duration))
    data = amp * np.sin(2*np.pi*freq *t + phase)
    return data

freq = 440
srate = 44100
duration = 5
amp = 2.0
phase = 5
time_space = np.linspace(0, 1000/44100,1000)
data = amp *np.sin(2*np.pi*freq*time_space+phase)
audio_file = wave.open('result.wav', 'w')
audio_file.setparams((1, 2, 44100, 1000, "NONE", "Uncompressed"))
for i in data:
    packed_value = struct.pack('h', np.int16(i))
    audio_file.writeframes(packed_value)
audio_file.close()
```

generate sin wave() from assignment 01. don’t know how to generate sine wave controlled by the centroid.

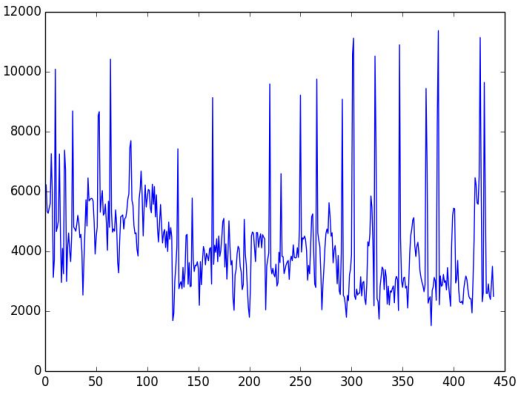
The little melody you created in assignment 1:



The same melody hummed by you and recorded in Audacity. Use a single vowel like Ah for your singing:



The qbhexamples.wav available under Resources in Connex:



Q2.2

When I used sfplay to play these downloaded music, they are sound strange. But with Audacity they are good. I can tell the similarities between classical and classical, metal and metal. Also, the differences between classical and metal is clear. If I comment out the MarSystems Memory and Mean, the sound will become less smooth.

Q2.3

```
Series {
    inSamples = 1024
    -> input: SoundFileSource { filename = "qbhexamples.wav" onSamples = 1024 inSamples = 1024 }
    -> MixToMono
    -> ShiftInput { winSize = 2048 }
    -> Fanout {
        -> Series {
            -> Spectrum
            -> PowerSpectrum
            -> Centroid
            -> Memory { memSize = 20 }
            -> Mean
            -> centroid: FlowToControl
        }

    }
    + done = (input/hasData == false)
}
```

```
Series {
    inSamples = 1024
    -> input: SoundFileSource { filename = "qbh_examples.wav" onSamples = 1024 inSamples = 1024}
    -> MixToMono
    -> ShiftInput { winSize = 2048 }
    -> Fanout {
        -> Series {
            -> Spectrum
            -> PowerSpectrum
            -> Centroid
            -> Memory { memSize = 20 }
            -> Mean
            -> centroid: FlowToControl
        }
        -> Series {
            -> SineSource { frequency = (centroid/value * (input/israte / 2)) }
            -> AudioSink
        }
    }
    + done = (input/hasData == false)
}
```