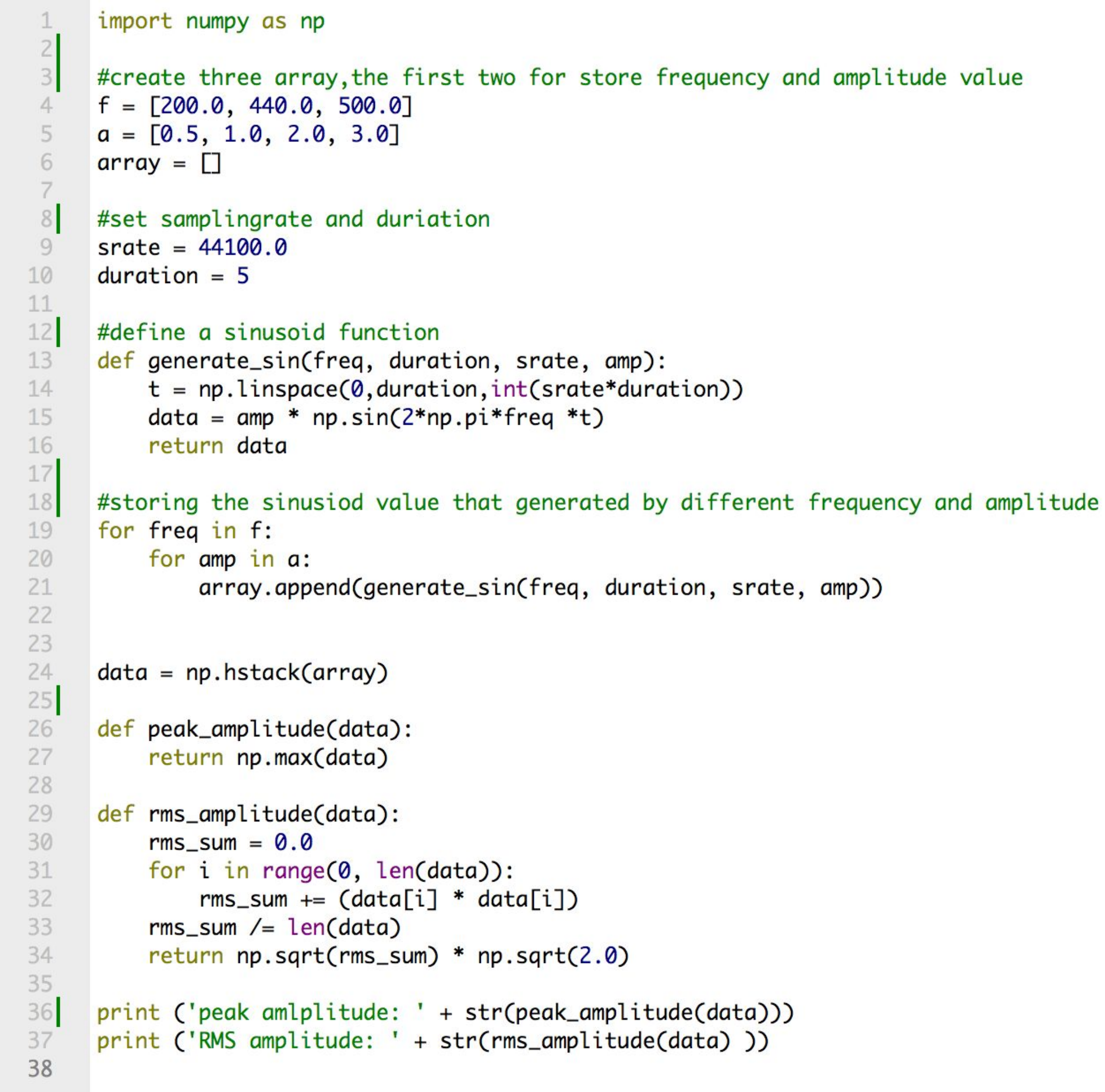
csc475 assignment_01

V00825808 XinYue Liu

Q1.1

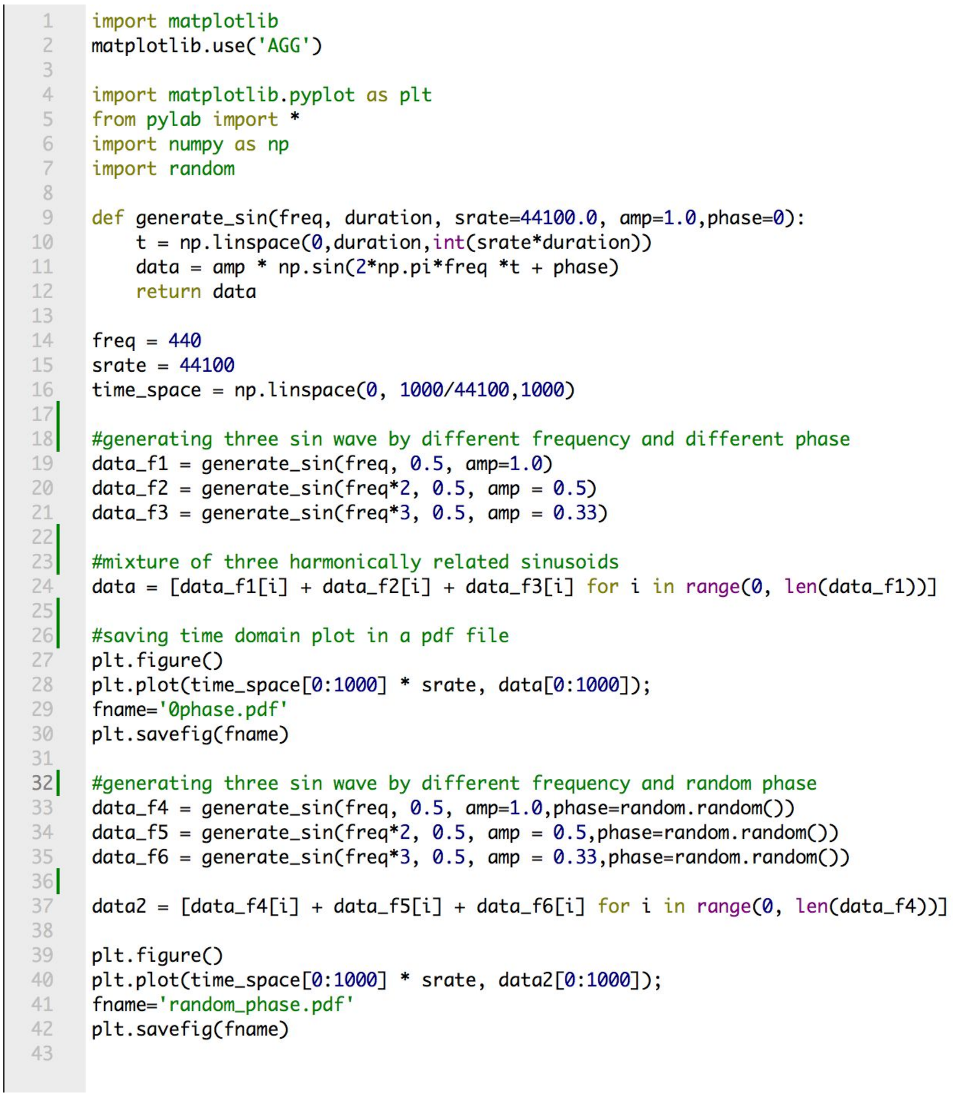


```
import numpy as np

#create three array,the first two for store frequency and amplitude value
f = [200.0, 440.0, 500.0]
a = [0.5, 1.0, 2.0, 3.0]
array = []

#set samplingrate and duriation
srate = 44100.0
duration = 5

#define a sinusoid function
def generate_sin(freq, duration, srate, amp):
    t = np.linspace(0,duration,int(srate*duration))
    data = amp * np.sin(2*np.pi*freq *t)
    return data

#storing the sinusiod value that generated by different frequency and amplitude
for freq in f:
    for amp in a:
        array.append(generate_sin(freq, duration, srate, amp))


data = np.hstack(array)

def peak_amplitude(data):
    return np.max(data)

def rms_amplitude(data):
    rms_sum = 0.0
    for i in range(0, len(data)):
        rms_sum += (data[i] * data[i])
    rms_sum /= len(data)
    return np.sqrt(rms_sum) * np.sqrt(2.0)

print ('peak amlplitude: ' + str(peak_amplitude(data)))
print ('RMS amplitude: ' + str(rms_amplitude(data) ))
```

output:

```
peak amlplitude: 2.99999999992
RMS amplitude: 1.88745432886
```

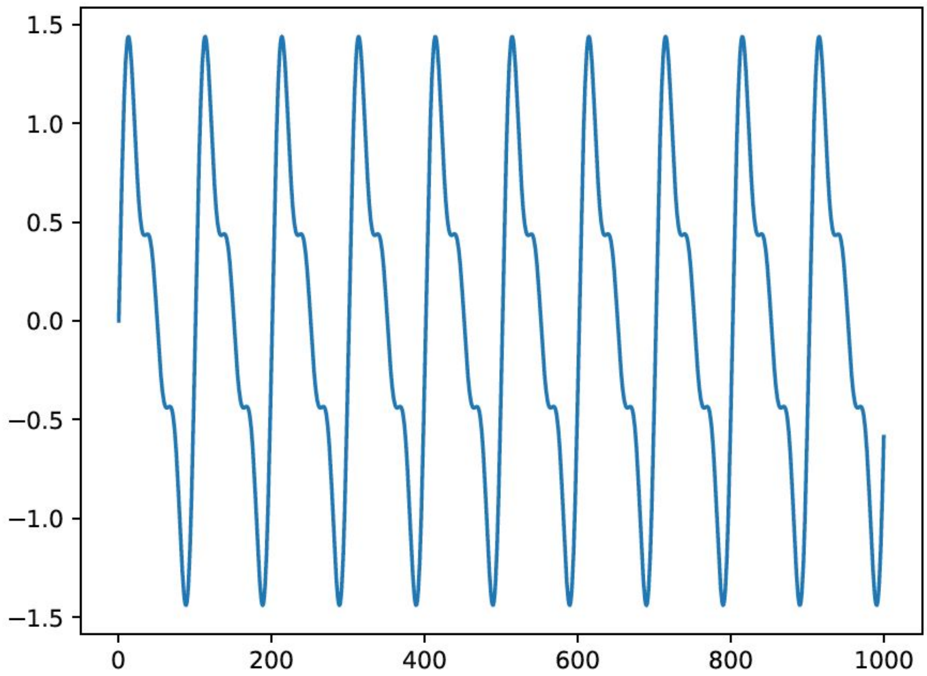
Q1.2



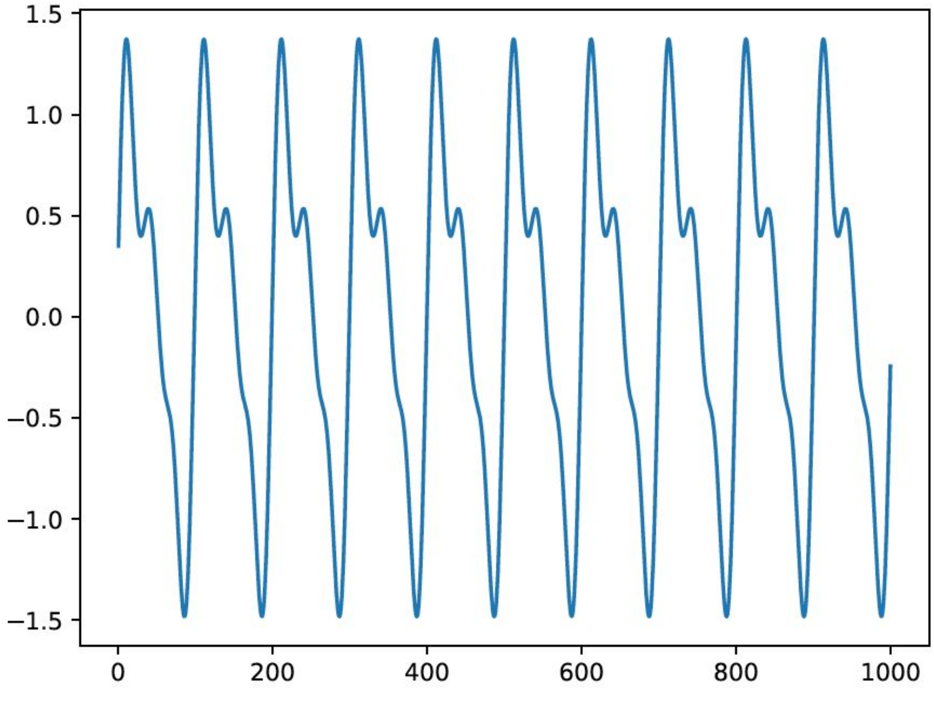
```python
import matplotlib
matplotlib.use('AGG')

import matplotlib.pyplot as plt
from pylab import *
import numpy as np
import random

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
time_space = np.linspace(0, 1000/44100,1000)

#generating three sin wave by different frequency and different phase
data_f1 = generate_sin(freq, 0.5, amp=1.0)
data_f2 = generate_sin(freq*2, 0.5, amp = 0.5)
data_f3 = generate_sin(freq*3, 0.5, amp = 0.33)

#mixture of three harmonically related sinusoids
data = [data_f1[i] + data_f2[i] + data_f3[i] for i in range(0, len(data_f1))]

#saving time domain plot in a pdf file
plt.figure()
plt.plot(time_space[0:1000] * srate, data[0:1000]);
fname='0phase.pdf'
plt.savefig(fname)

#generating three sin wave by different frequency and random phase
data_f4 = generate_sin(freq, 0.5, amp=1.0,phase=random.random())
data_f5 = generate_sin(freq*2, 0.5, amp = 0.5,phase=random.random())
data_f6 = generate_sin(freq*3, 0.5, amp = 0.33,phase=random.random())

data2 = [data_f4[i] + data_f5[i] + data_f6[i] for i in range(0, len(data_f4))]

plt.figure()
plt.plot(time_space[0:1000] * srate, data2[0:1000]);
fname='random_phase.pdf'
plt.savefig(fname)
```

output:

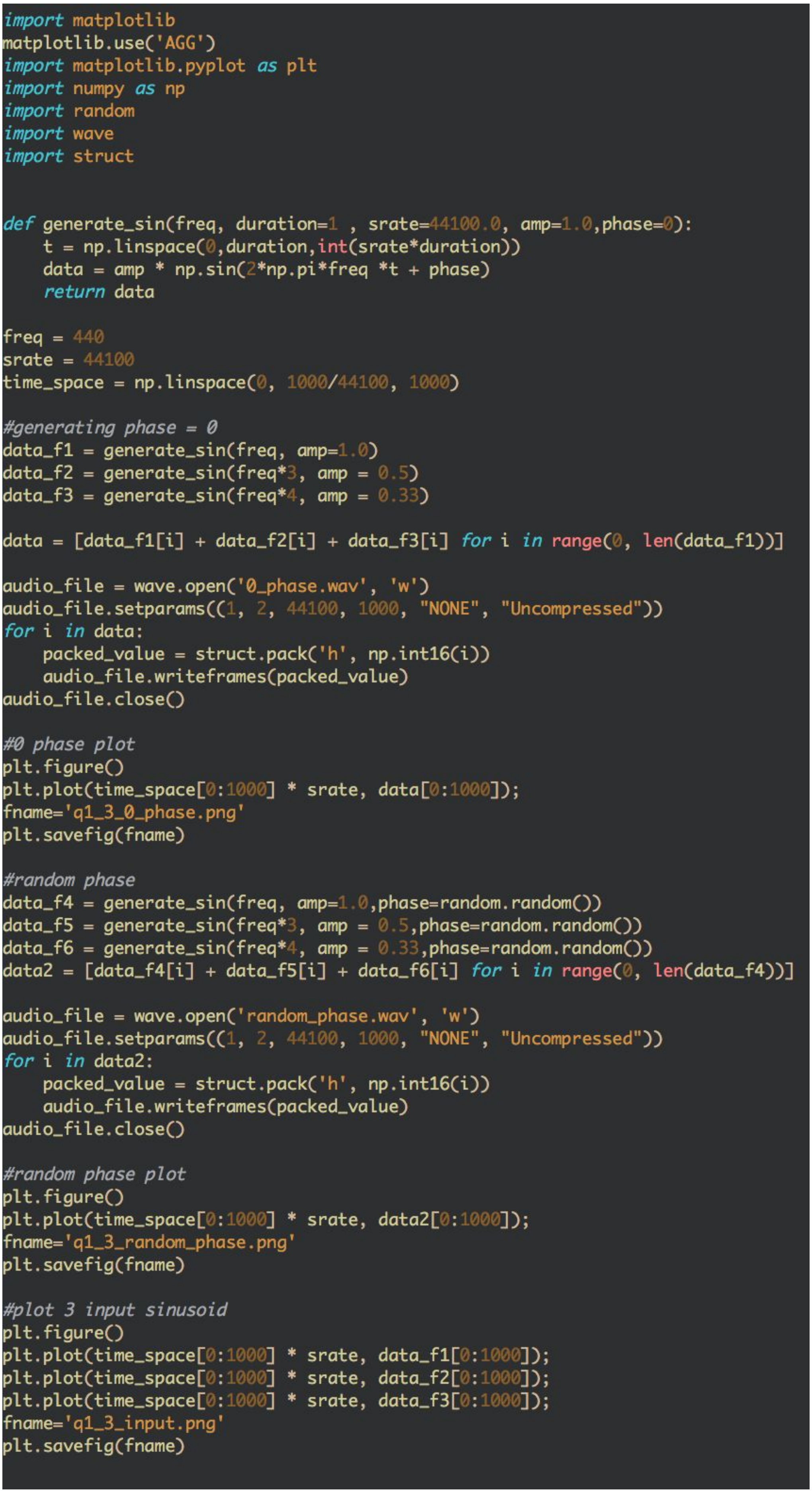
0 phase



random phase:

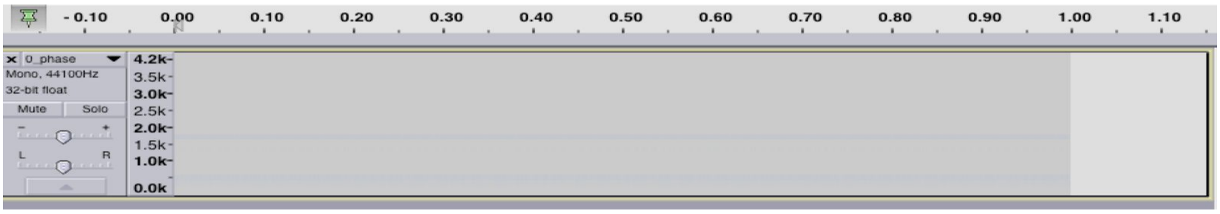


Q1.3

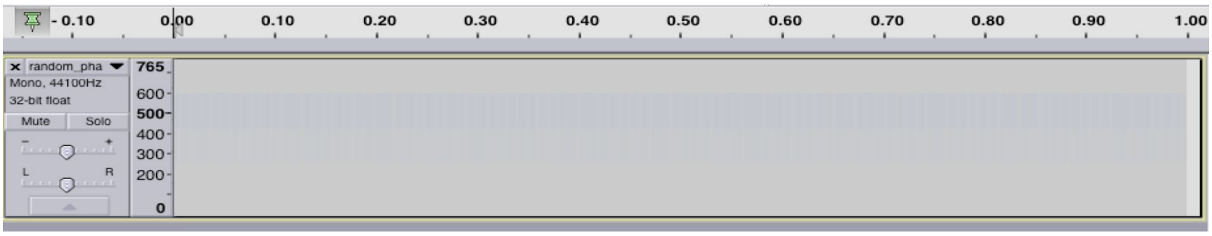


```python
import matplotlib
matplotlib.use('AGG')
import matplotlib.pyplot as plt
import numpy as np
import random
import wave
import struct


def generate_sin(freq, duration=1 , 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
time_space = np.linspace(0, 1000/44100, 1000)

#generating phase = 0
data_f1 = generate_sin(freq, amp=1.0)
data_f2 = generate_sin(freq*3, amp = 0.5)
data_f3 = generate_sin(freq*4, amp = 0.33)

data = [data_f1[i] + data_f2[i] + data_f3[i] for i in range(0, len(data_f1))]

audio_file = wave.open('0_phase.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()

#0 phase plot
plt.figure()
plt.plot(time_space[0:1000] * srate, data[0:1000]);
fname='q1_3_0_phase.png'
plt.savefig(fname)

#random phase
data_f4 = generate_sin(freq, amp=1.0,phase=random.random())
data_f5 = generate_sin(freq*3, amp = 0.5,phase=random.random())
data_f6 = generate_sin(freq*4, amp = 0.33,phase=random.random())
data2 = [data_f4[i] + data_f5[i] + data_f6[i] for i in range(0, len(data_f4))]

audio_file = wave.open('random_phase.wav', 'w')
audio_file.setparams((1, 2, 44100, 1000, "NONE", "Uncompressed"))
for i in data2:
    packed_value = struct.pack('h', np.int16(i))
    audio_file.writeframes(packed_value)
audio_file.close()

#random phase plot
plt.figure()
plt.plot(time_space[0:1000] * srate, data2[0:1000]);
fname='q1_3_random_phase.png'
plt.savefig(fname)

#plot 3 input sinusoid
plt.figure()
plt.plot(time_space[0:1000] * srate, data_f1[0:1000]);
plt.plot(time_space[0:1000] * srate, data_f2[0:1000]);
plt.plot(time_space[0:1000] * srate, data_f3[0:1000]);
fname='q1_3_input.png'
plt.savefig(fname)
```

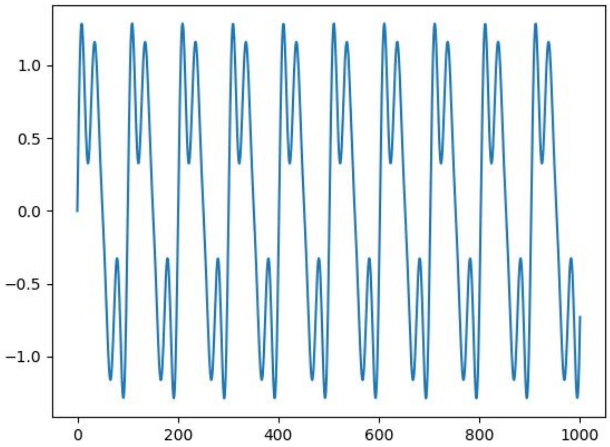
0 phase spectrogram:



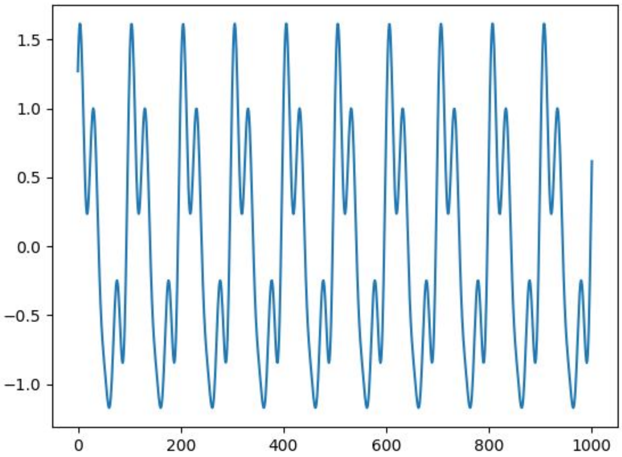
random phase spectrogram:



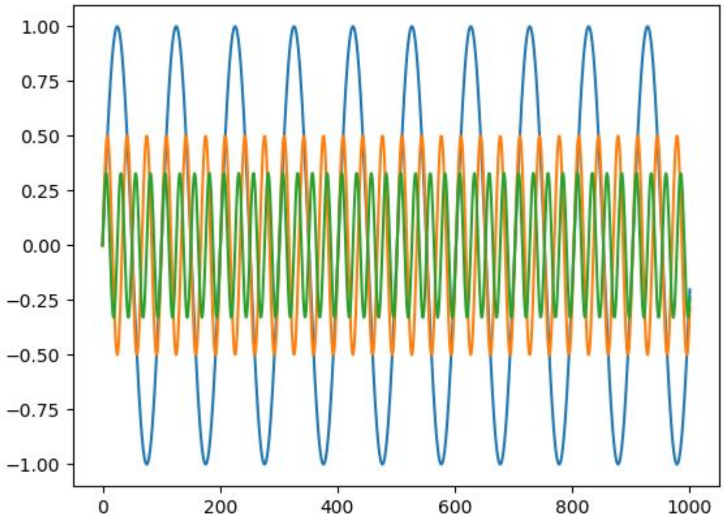
0 phase plot:



random phase plot:

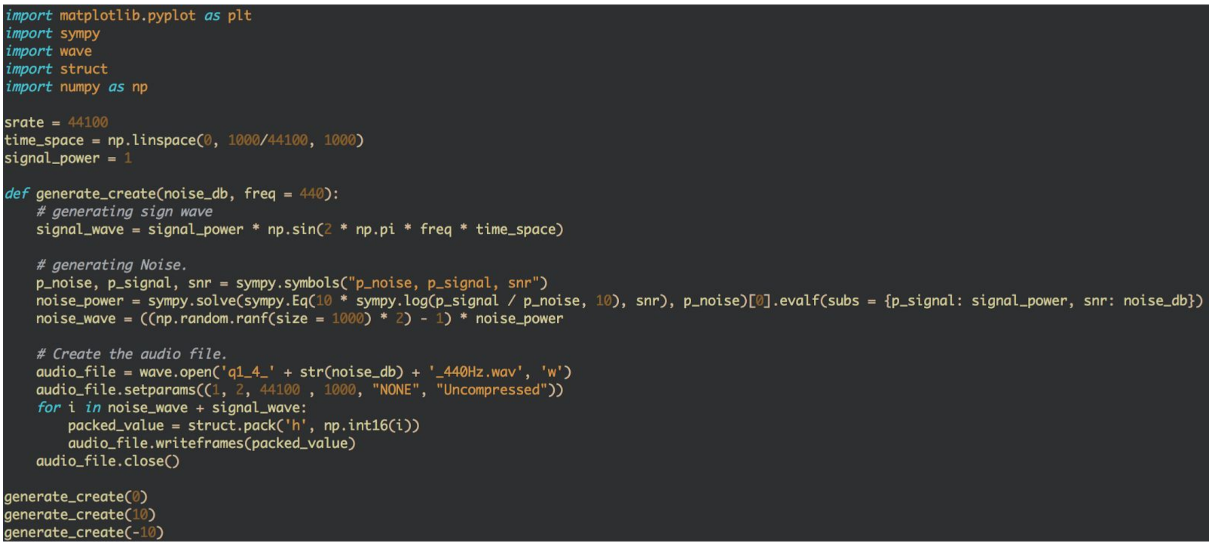


3 input plot:



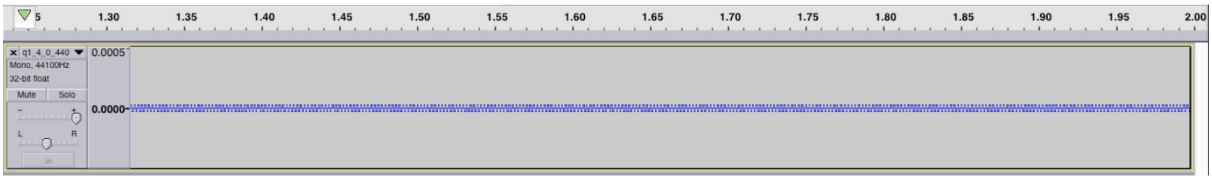
The plot figuer and spectrogram have apperentlly difference, but I canoot hear the difference between two audios.

Q1.4

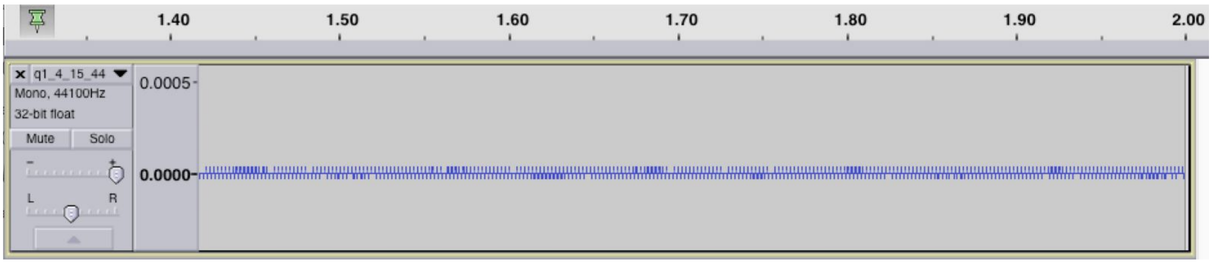


```
import matplotlib.pyplot as plt
import sympy
import wave
import struct
import numpy as np

srate = 44100
time_space = np.linspace(0, 1000/44100, 1000)
signal_power = 1

def generate_create(noise_db, freq = 440):
    # generating sign wave
    signal_wave = signal_power * np.sin(2 * np.pi * freq * time_space)

    # generating Noise.
    p_noise, p_signal, snr = sympy.symbols("p_noise, p_signal, snr")
    noise_power = sympy.solve(sympy.Eq(10 * sympy.log(p_signal / p_noise, 10), snr), p_noise)[0].evalf(subs = {p_signal: signal_power, snr: noise_db})
    noise_wave = ((np.random.ranf(size = 1000) * 2) - 1) * noise_power

    # Create the audio file.
    audio_file = wave.open('q1_4_' + str(noise_db) + '_440Hz.wav', 'w')
    audio_file.setparams((1, 2, 44100 , 1000, "NONE", "Uncompressed"))
    for i in noise_wave + signal_wave:
        packed_value = struct.pack('h', np.int16(i))
        audio_file.writeframes(packed_value)
    audio_file.close()

generate_create(0)
generate_create(10)
generate_create(-10)
```

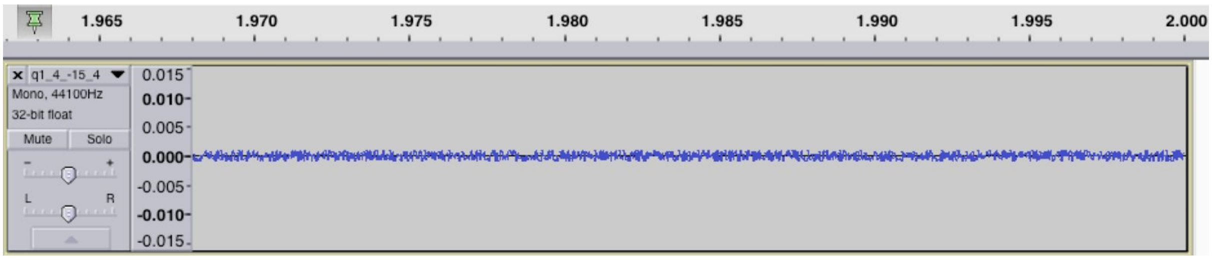
db=0:



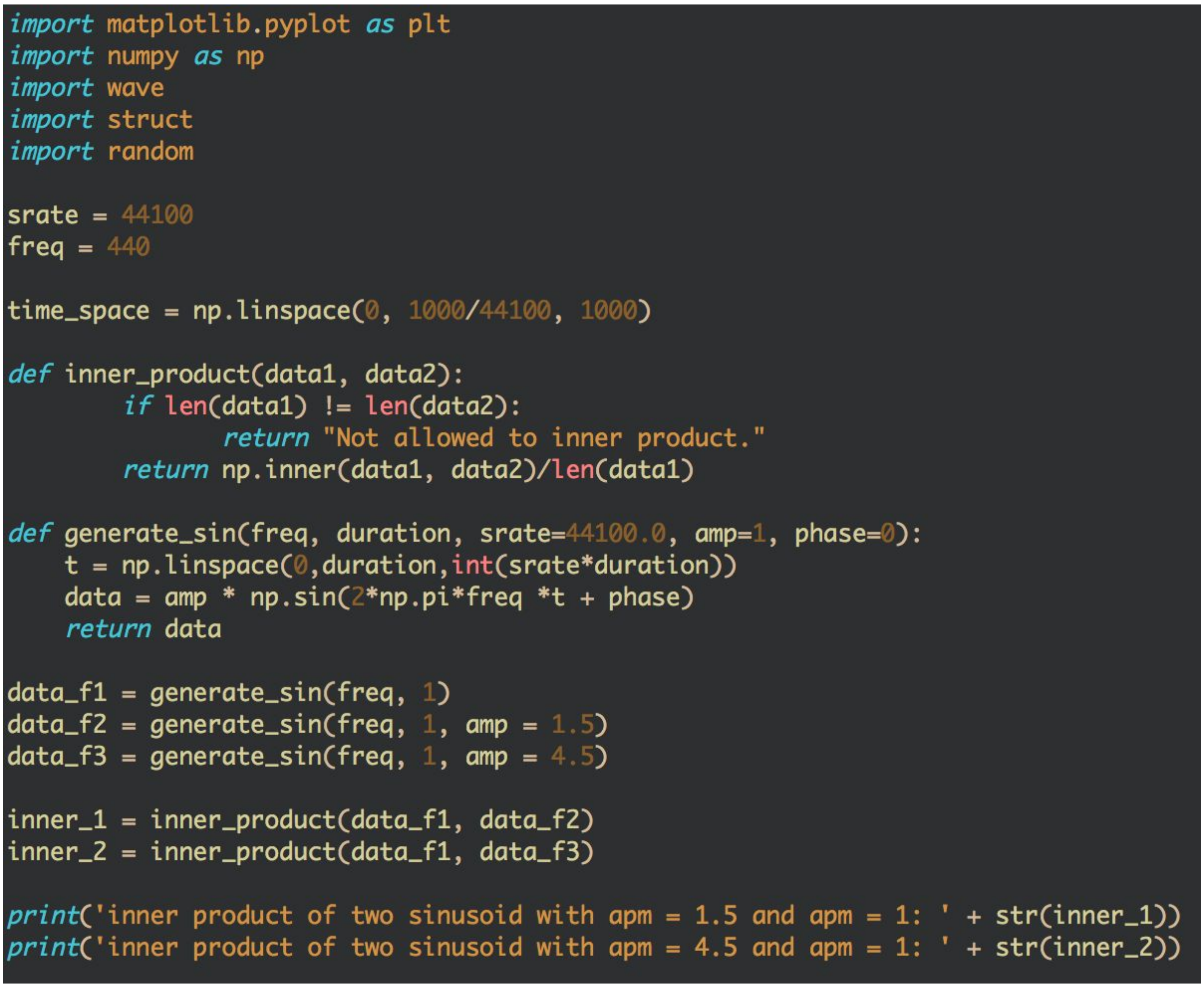
db = 15:



db = -15:



Q1.5



```
import matplotlib.pyplot as plt
import numpy as np
import wave
import struct
import random

srate = 44100
freq = 440

time_space = np.linspace(0, 1000/44100, 1000)

def inner_product(data1, data2):
        if len(data1) != len(data2):
               return "Not allowed to inner product."
        return np.inner(data1, data2)/len(data1)

def generate_sin(freq, duration, srate=44100.0, amp=1, phase=0):
    t = np.linspace(0,duration,int(srate*duration))
    data = amp * np.sin(2*np.pi*freq *t + phase)
    return data

data_f1 = generate_sin(freq, 1)
data_f2 = generate_sin(freq, 1, amp = 1.5)
data_f3 = generate_sin(freq, 1, amp = 4.5)

inner_1 = inner_product(data_f1, data_f2)
inner_2 = inner_product(data_f1, data_f3)

print('inner product of two sinusoid with apm = 1.5 and apm = 1: ' + str(inner_1))
print('inner product of two sinusoid with apm = 4.5 and apm = 1: ' + str(inner_2))
```

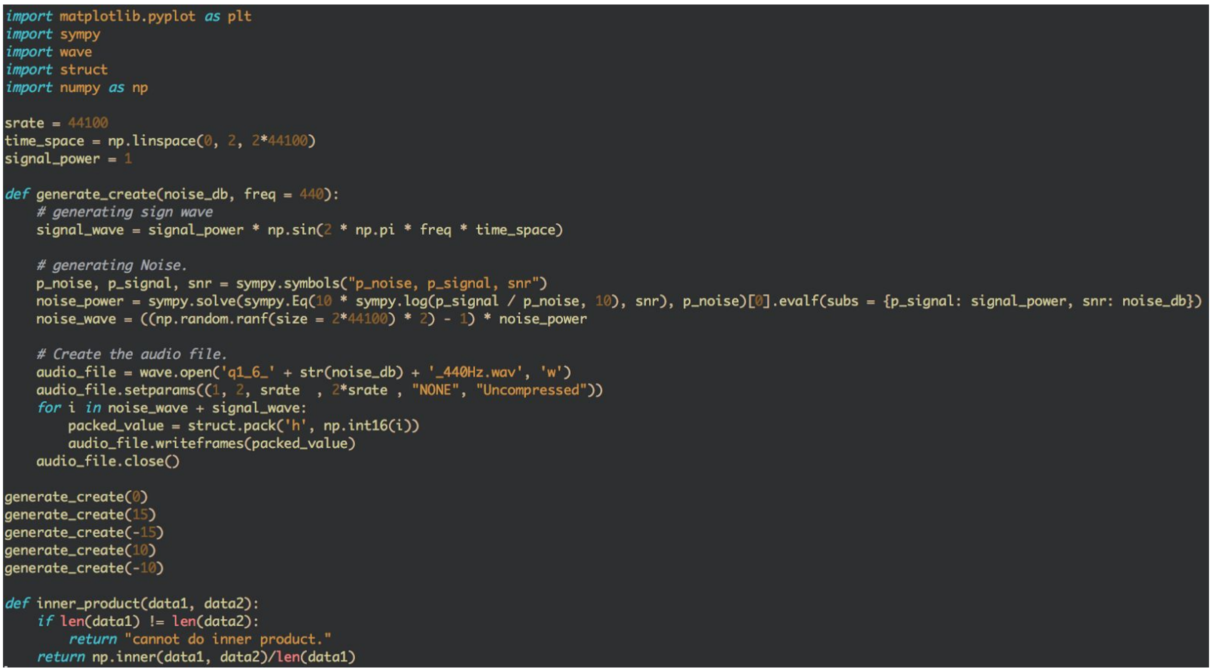
output:

```
inner product of two sinusoid with apm = 1.5 and apm = 1: 0.749982993197
inner product of two sinusoid with apm = 4.5 and apm = 1: 2.24994897959
```

When I change a to a bigger value, the value of inner product become larger.

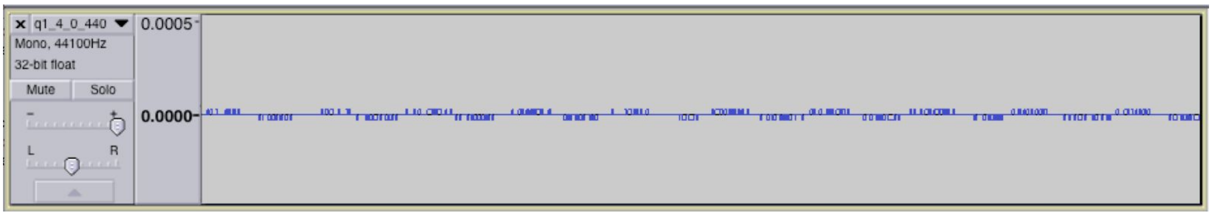
formula: amplitude = innerproduct*amplitude_unit/0.5

Q1.6

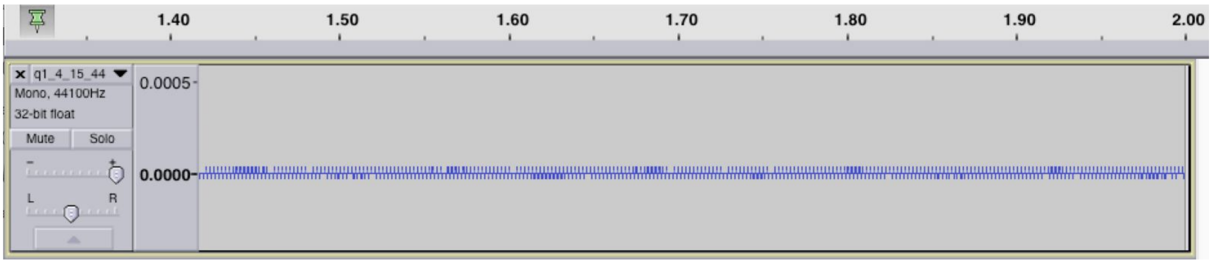


```python
import matplotlib.pyplot as plt
import sympy
import wave
import struct
import numpy as np

srate = 44100
time_space = np.linspace(0, 2, 2*44100)
signal_power = 1

def generate_create(noise_db, freq = 440):
    # generating sign wave
    signal_wave = signal_power * np.sin(2 * np.pi * freq * time_space)

    # generating Noise.
    p_noise, p_signal, snr = sympy.symbols("p_noise, p_signal, snr")
    noise_power = sympy.solve(sympy.Eq(10 * sympy.log(p_signal / p_noise, 10), snr), p_noise)[0].evalf(subs = {p_signal: signal_power, snr: noise_db})
    noise_wave = ((np.random.ranf(size = 2*44100) * 2) - 1) * noise_power

    # Create the audio file.
    audio_file = wave.open('q1_6_' + str(noise_db) + '_440Hz.wav', 'w')
    audio_file.setparams((1, 2, srate  , 2*srate , "NONE", "Uncompressed"))
    for i in noise_wave + signal_wave:
        packed_value = struct.pack('h', np.int16(i))
        audio_file.writeframes(packed_value)
    audio_file.close()

generate_create(0)
generate_create(15)
generate_create(-15)
generate_create(10)
generate_create(-10)

def inner_product(data1, data2):
    if len(data1) != len(data2):
        return "cannot do inner product."
    return np.inner(data1, data2)/len(data1)
```

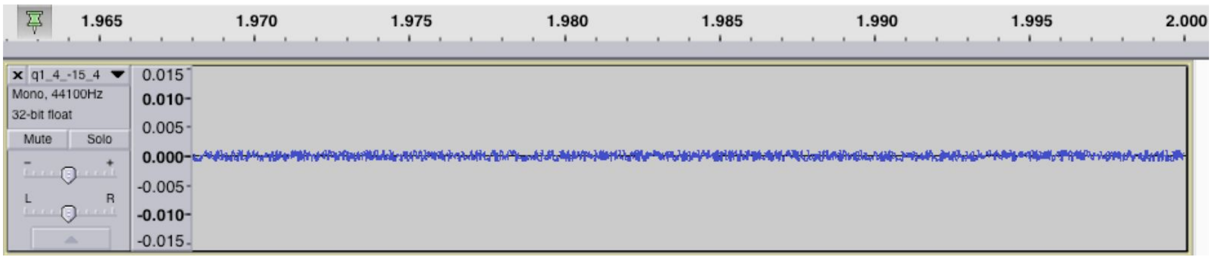
db = 0



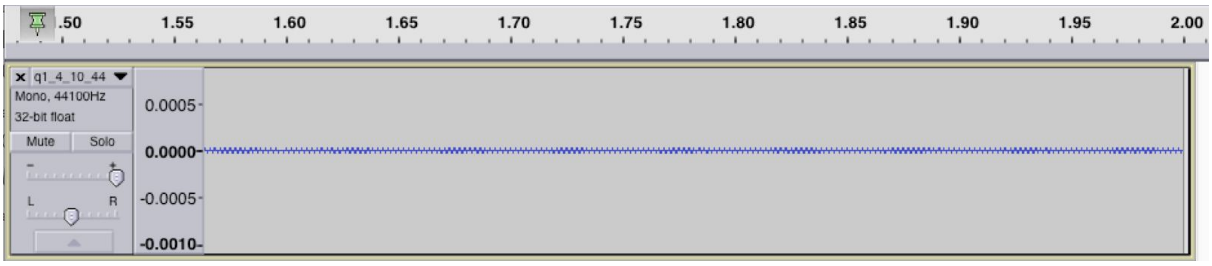
db = 15



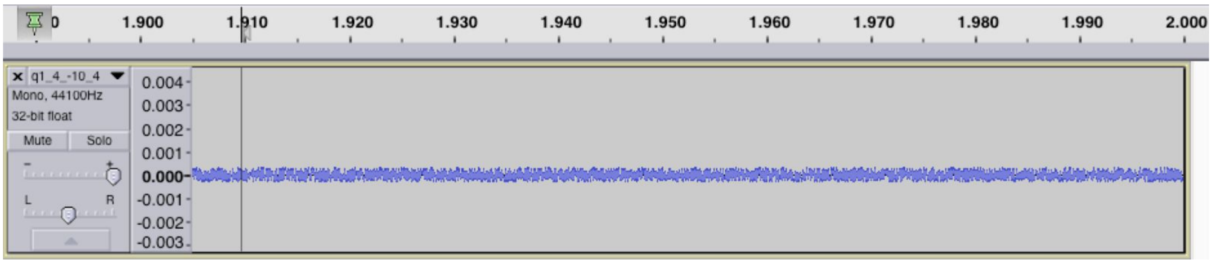
db = -15



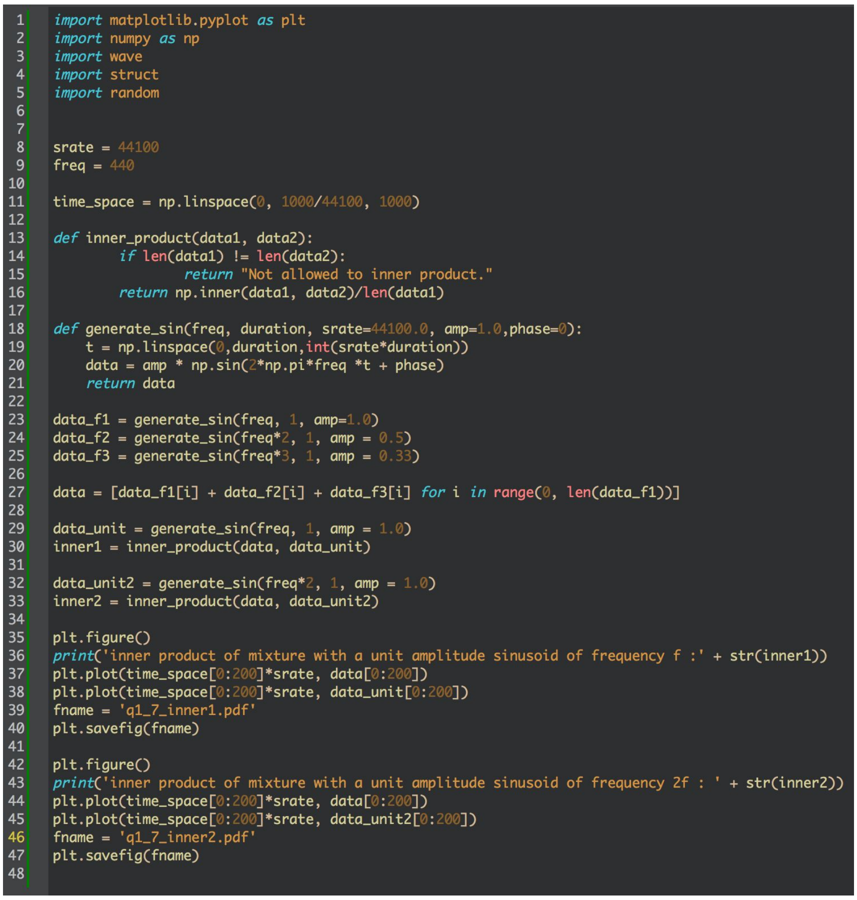
db = 10



db = -10



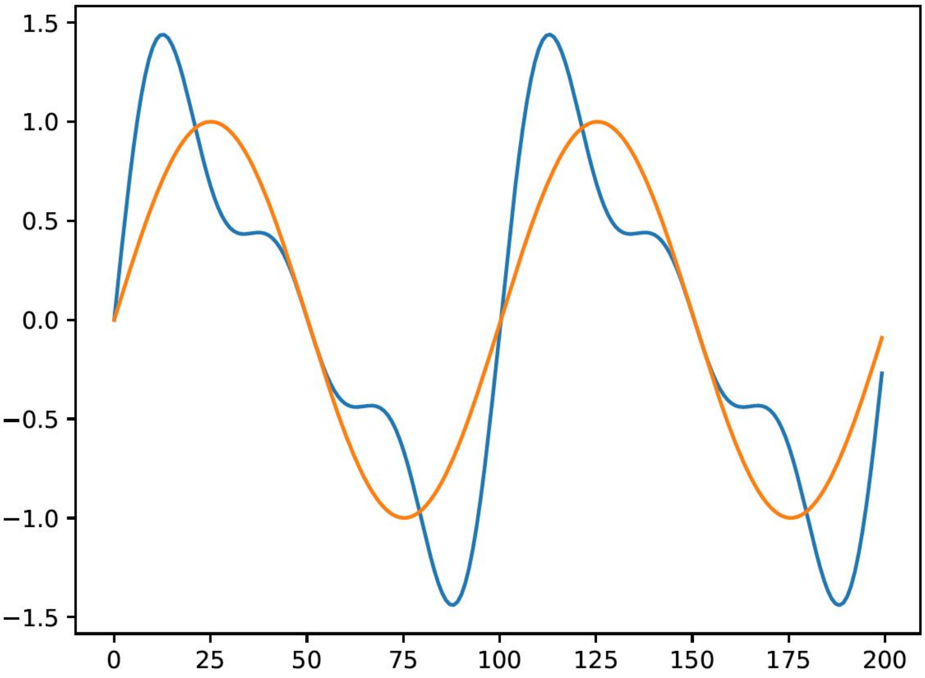
Q1.7



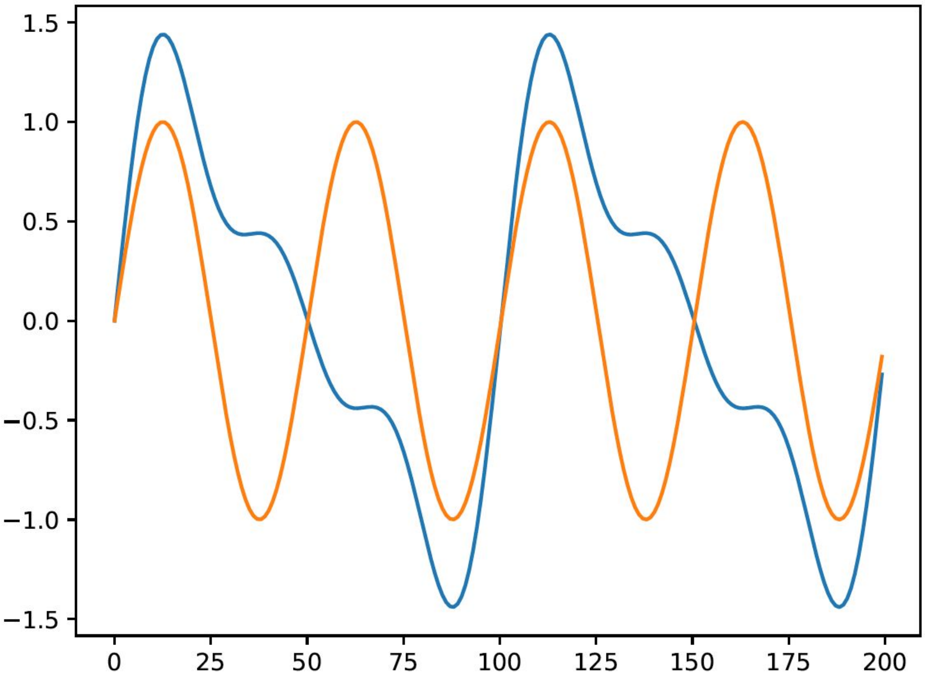
```
import matplotlib.pyplot as plt
import numpy as np
import wave
import struct
import random


srate = 44100
freq = 440

time_space = np.linspace(0, 1000/44100, 1000)

def inner_product(data1, data2):
        if len(data1) != len(data2):
                return "Not allowed to inner product."
        return np.inner(data1, data2)/len(data1)

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

data_f1 = generate_sin(freq, 1, amp=1.0)
data_f2 = generate_sin(freq*2, 1, amp = 0.5)
data_f3 = generate_sin(freq*3, 1, amp = 0.33)

data = [data_f1[i] + data_f2[i] + data_f3[i] for i in range(0, len(data_f1))]

data_unit = generate_sin(freq, 1, amp = 1.0)
inner1 = inner_product(data, data_unit)

data_unit2 = generate_sin(freq*2, 1, amp = 1.0)
inner2 = inner_product(data, data_unit2)

plt.figure()
print('inner product of mixture with a unit amplitude sinusoid of frequency f :' + str(inner1))
plt.plot(time_space[0:200]*srate, data[0:200])
plt.plot(time_space[0:200]*srate, data_unit[0:200])
fname = 'q1_7_inner1.pdf'
plt.savefig(fname)

plt.figure()
print('inner product of mixture with a unit amplitude sinusoid of frequency 2f : ' + str(inner2))
plt.plot(time_space[0:200]*srate, data[0:200])
plt.plot(time_space[0:200]*srate, data_unit2[0:200])
fname = 'q1_7_inner2.pdf'
plt.savefig(fname)

```

output:

inner product of mixture with a unit amplitude sinusoid of frequency f :0.499988662132

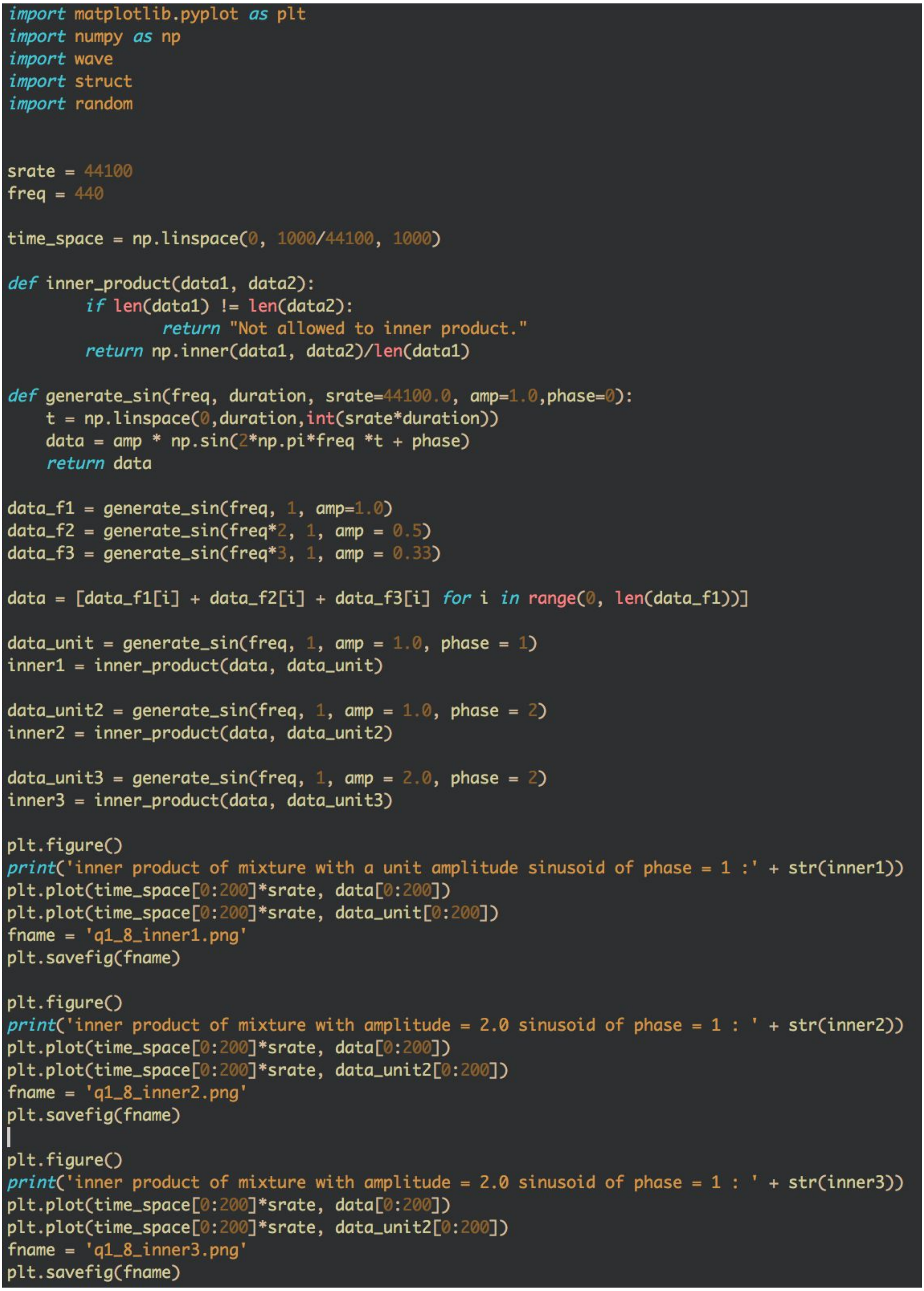


inner product of mixture with a unit amplitude sinusoid of frequency 2f : 0.249994331066



When adding a mixture with a sinusoid that has different frequency, if the frequency gets bigger, the value of inner product gets smaller.

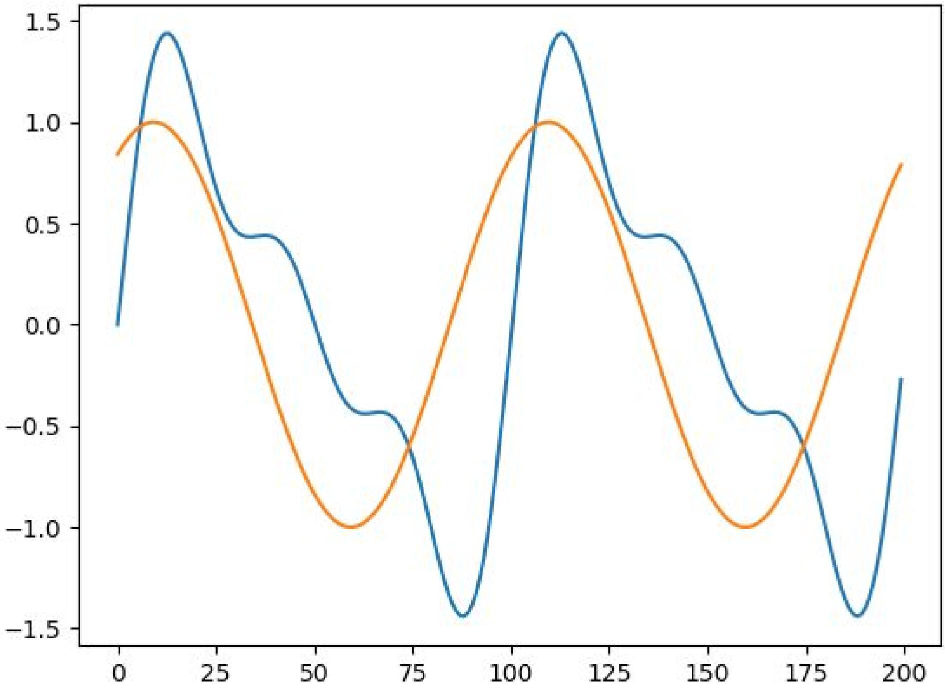
Q1.8



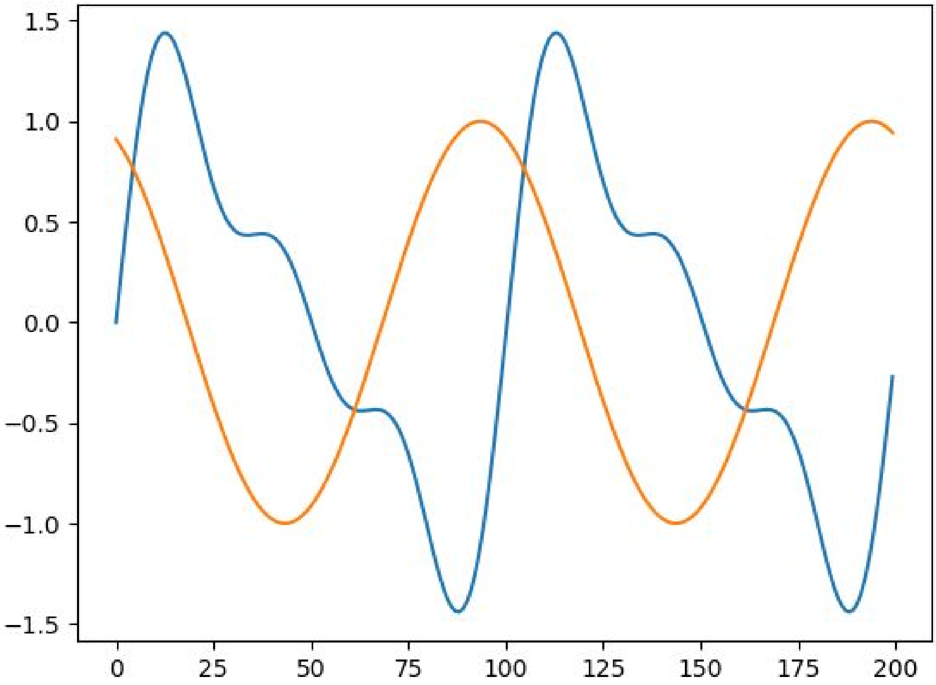
```python
import matplotlib.pyplot as plt
import numpy as np
import wave
import struct
import random


srate = 44100
freq = 440

time_space = np.linspace(0, 1000/44100, 1000)

def inner_product(data1, data2):
        if len(data1) != len(data2):
                return "Not allowed to inner product."
        return np.inner(data1, data2)/len(data1)

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

data_f1 = generate_sin(freq, 1, amp=1.0)
data_f2 = generate_sin(freq*2, 1, amp = 0.5)
data_f3 = generate_sin(freq*3, 1, amp = 0.33)

data = [data_f1[i] + data_f2[i] + data_f3[i] for i in range(0, len(data_f1))]

data_unit = generate_sin(freq, 1, amp = 1.0, phase = 1)
inner1 = inner_product(data, data_unit)

data_unit2 = generate_sin(freq, 1, amp = 1.0, phase = 2)
inner2 = inner_product(data, data_unit2)

data_unit3 = generate_sin(freq, 1, amp = 2.0, phase = 2)
inner3 = inner_product(data, data_unit3)

plt.figure()
print('inner product of mixture with a unit amplitude sinusoid of phase = 1 :' + str(inner1))
plt.plot(time_space[0:200]*srate, data[0:200])
plt.plot(time_space[0:200]*srate, data_unit[0:200])
fname = 'q1_8_inner1.png'
plt.savefig(fname)

plt.figure()
print('inner product of mixture with amplitude = 2.0 sinusoid of phase = 1 : ' + str(inner2))
plt.plot(time_space[0:200]*srate, data[0:200])
plt.plot(time_space[0:200]*srate, data_unit2[0:200])
fname = 'q1_8_inner2.png'
plt.savefig(fname)

plt.figure()
print('inner product of mixture with amplitude = 2.0 sinusoid of phase = 1 : ' + str(inner3))
plt.plot(time_space[0:200]*srate, data[0:200])
plt.plot(time_space[0:200]*srate, data_unit2[0:200])
fname = 'q1_8_inner3.png'
plt.savefig(fname)
```

output：

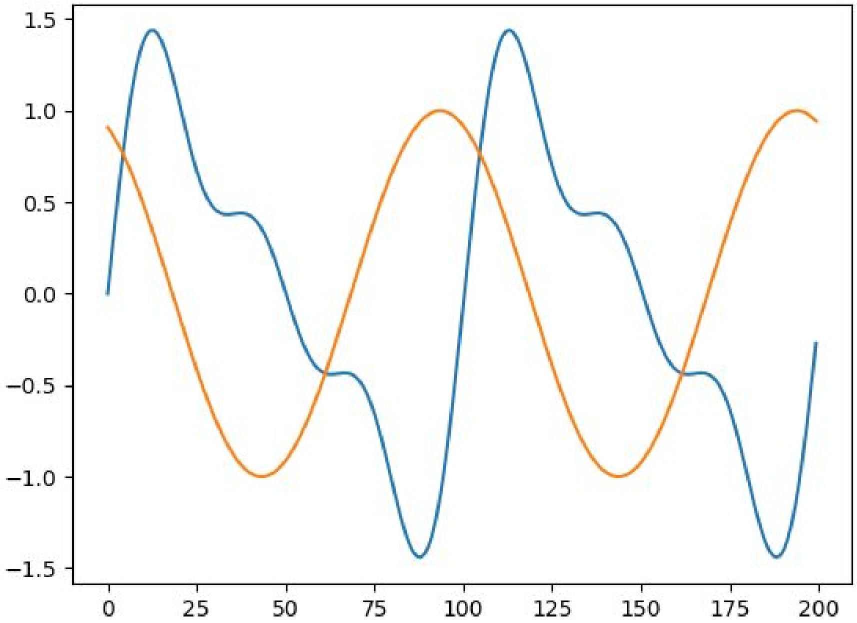
inner product of mixture with a unit amplitude sinusoid of phase = 1 :0.270145027058



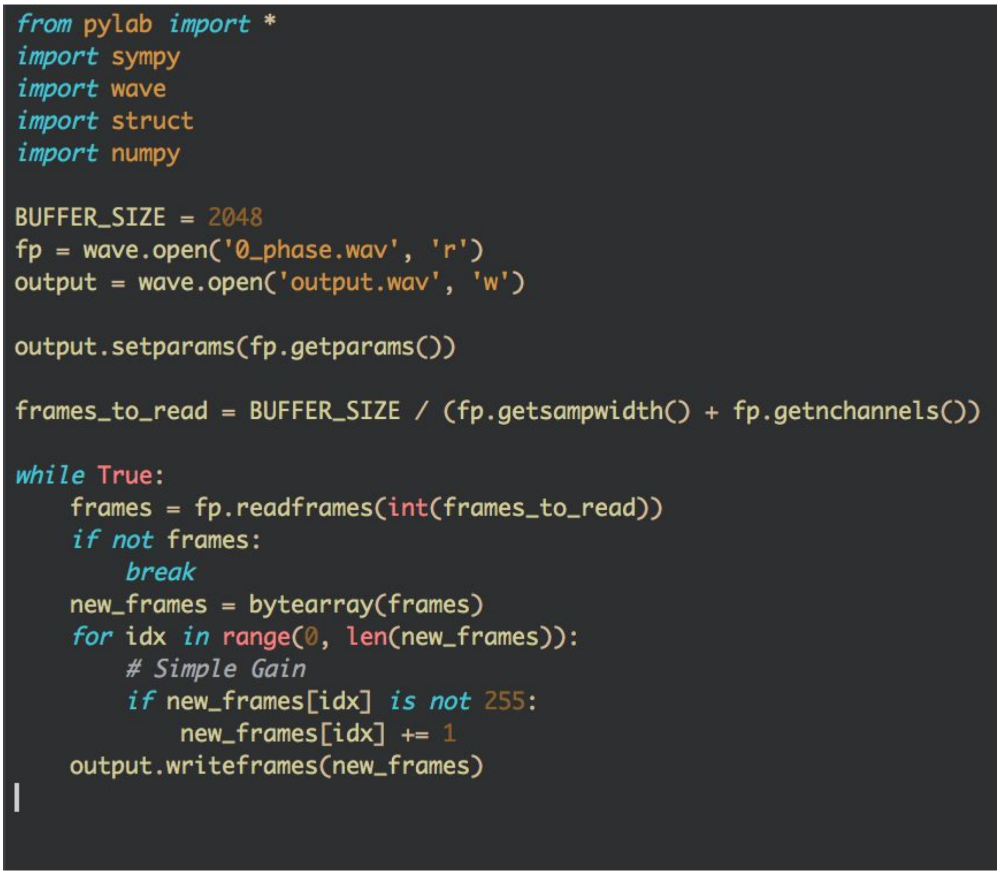
```
inner product of mixture with amplitude = 2.0 sinusoid of phase = 1 : -0.208068700055
```



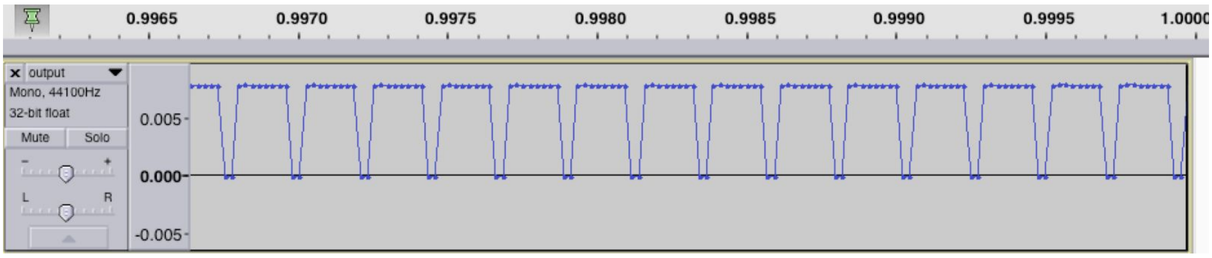
```
inner product of mixture with amplitude = 2.0 sinusoid of phase = 1 : -0.416137400111
```



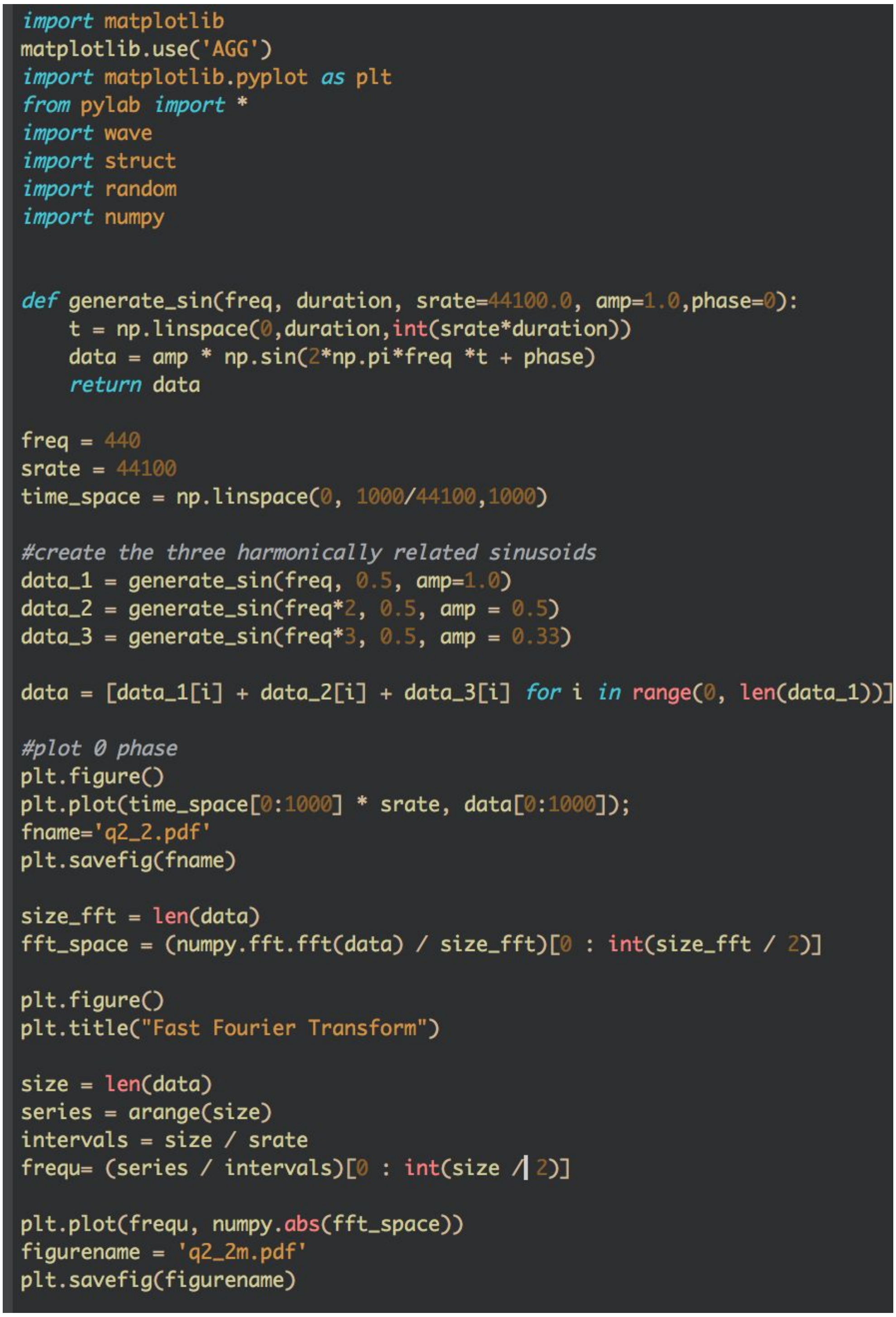
Q2.1



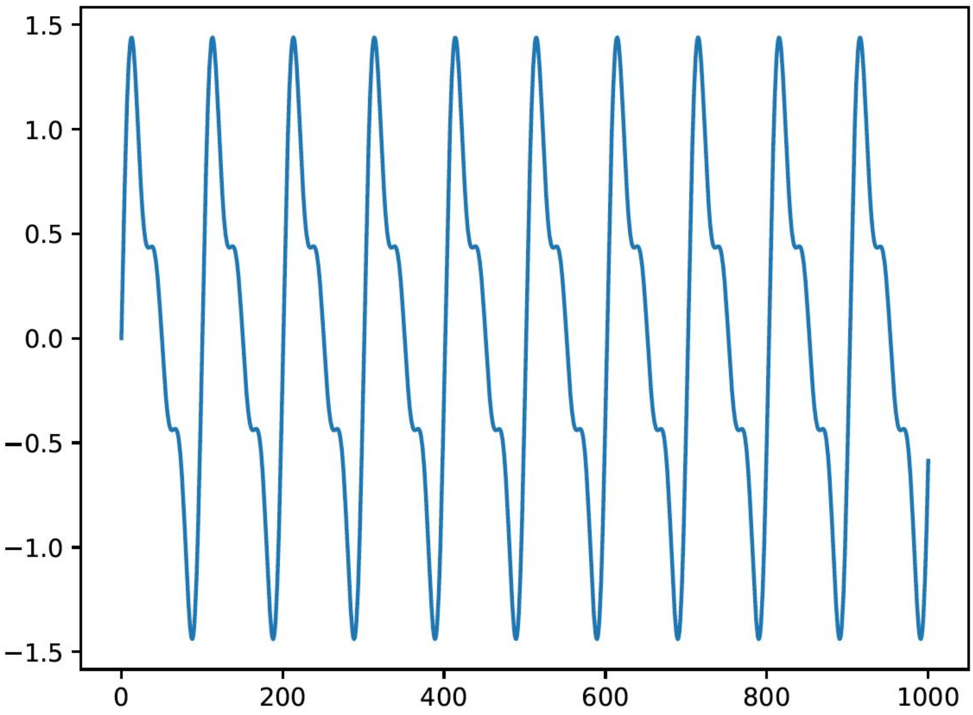
```
from pylab import *
import sympy
import wave
import struct
import numpy

BUFFER_SIZE = 2048
fp = wave.open('0_phase.wav', 'r')
output = wave.open('output.wav', 'w')

output.setparams(fp.getparams())

frames_to_read = BUFFER_SIZE / (fp.getsampwidth() + fp.getnchannels())

while True:
    frames = fp.readframes(int(frames_to_read))
    if not frames:
        break
    new_frames = bytearray(frames)
    for idx in range(0, len(new_frames)):
        # Simple Gain
        if new_frames[idx] is not 255:
            new_frames[idx] += 1
    output.writeframes(new_frames)
```

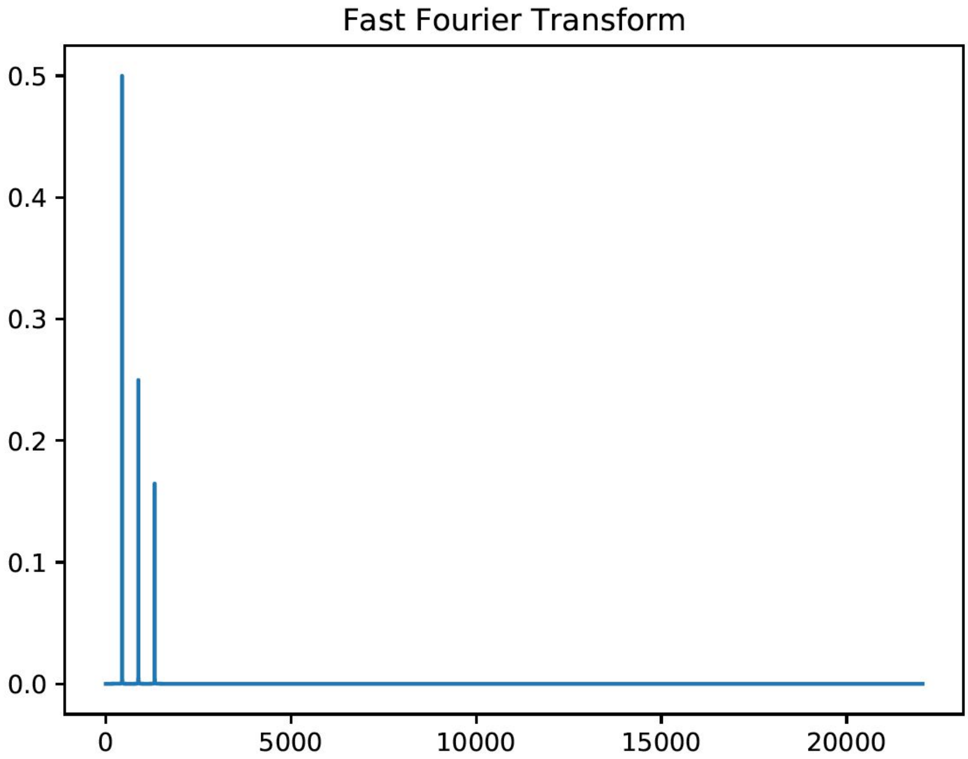


Q2.2

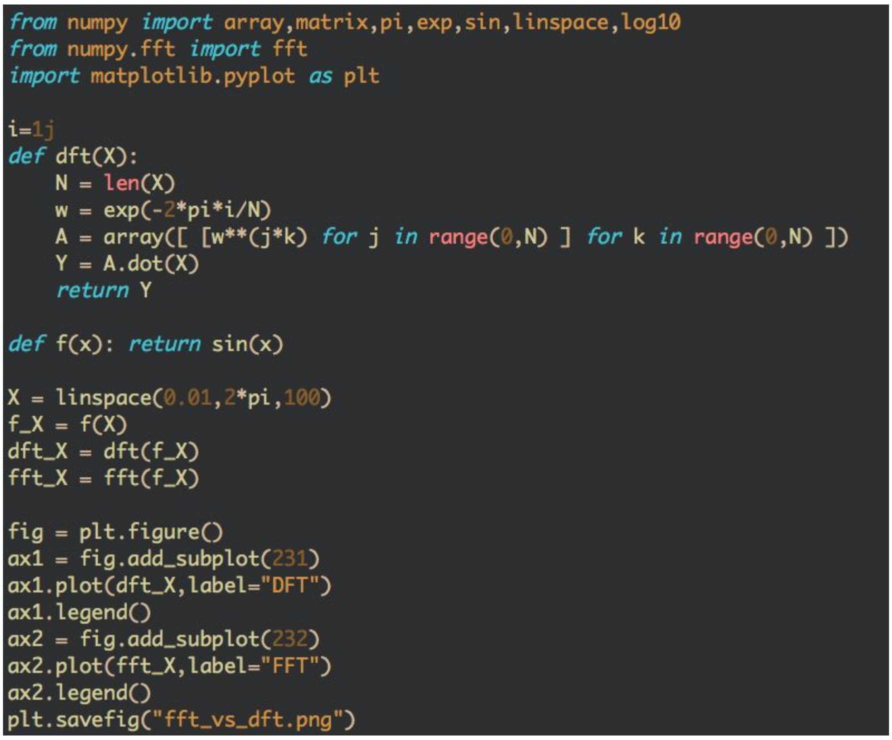


```
import matplotlib
matplotlib.use('AGG')
import matplotlib.pyplot as plt
from pylab import *
import wave
import struct
import random
import numpy


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
time_space = np.linspace(0, 1000/44100,1000)

#create the three harmonically related sinusoids
data_1 = generate_sin(freq, 0.5, amp=1.0)
data_2 = generate_sin(freq*2, 0.5, amp = 0.5)
data_3 = generate_sin(freq*3, 0.5, amp = 0.33)

data = [data_1[i] + data_2[i] + data_3[i] for i in range(0, len(data_1))]

#plot 0 phase
plt.figure()
plt.plot(time_space[0:1000] * srate, data[0:1000]);
fname='q2_2.pdf'
plt.savefig(fname)

size_fft = len(data)
fft_space = (numpy.fft.fft(data) / size_fft)[0 : int(size_fft / 2)]

plt.figure()
plt.title("Fast Fourier Transform")

size = len(data)
series = arange(size)
intervals = size / srate
frequ= (series / intervals)[0 : int(size / 2)]

plt.plot(frequ, numpy.abs(fft_space))
figurename = 'q2_2m.pdf'
plt.savefig(figurename)
```

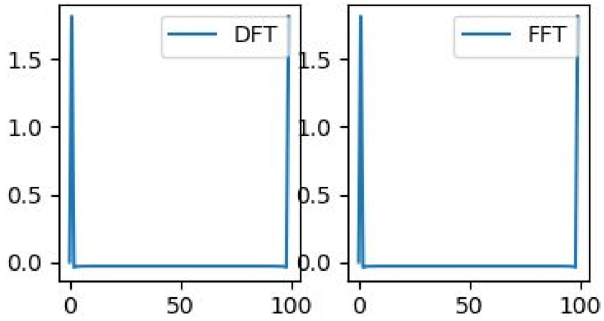

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
200
400
600
800
1000


Fast Fourier Transform
0.5
0.4
0.3
0.2
0.1
0.0
0
5000
10000
15000
20000

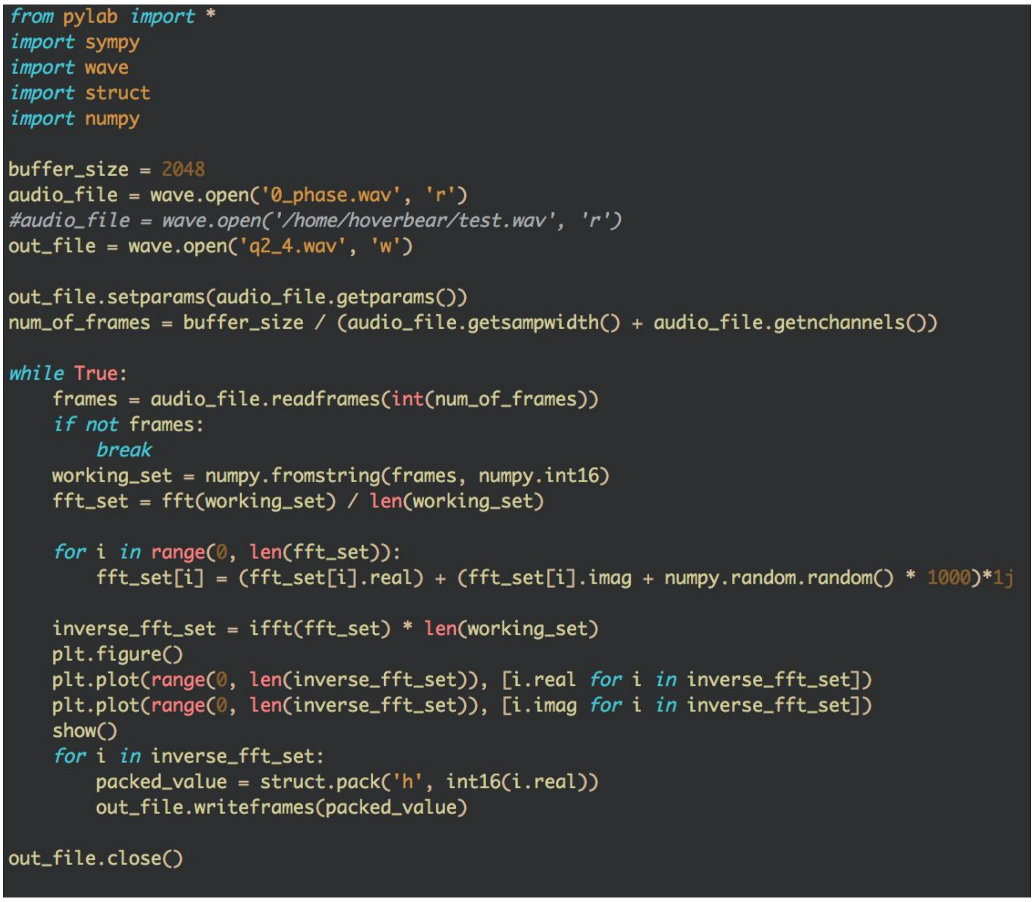
Q2.3



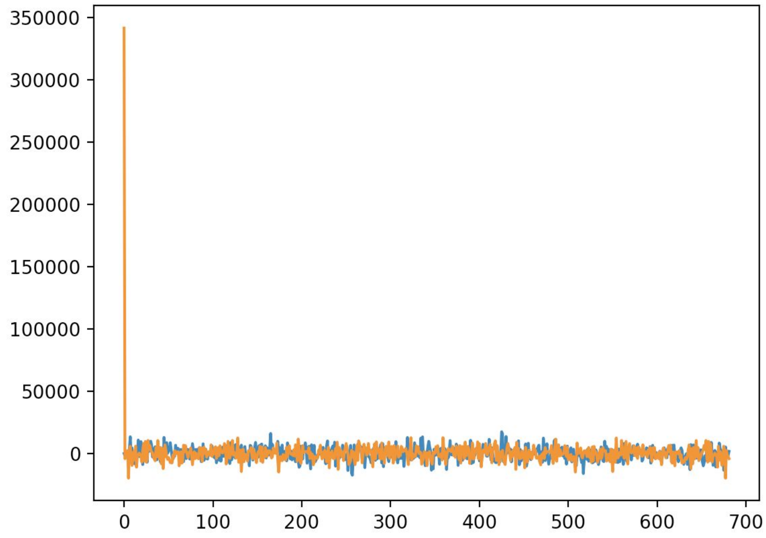
```python
from numpy import array,matrix,pi,exp,sin,linspace,log10
from numpy.fft import fft
import matplotlib.pyplot as plt

i=1j
def dft(X):
    N = len(X)
    w = exp(-2*pi*i/N)
    A = array([ [w**(j*k) for j in range(0,N) ] for k in range(0,N) ])
    Y = A.dot(X)
    return Y

def f(x): return sin(x)

X = linspace(0.01,2*pi,100)
f_X = f(X)
dft_X = dft(f_X)
fft_X = fft(f_X)

fig = plt.figure()
ax1 = fig.add_subplot(231)
ax1.plot(dft_X,label="DFT")
ax1.legend()
ax2 = fig.add_subplot(232)
ax2.plot(fft_X,label="FFT")
ax2.legend()
plt.savefig("fft_vs_dft.png")
```



Q2.4



```
from pylab import *
import sympy
import wave
import struct
import numpy

buffer_size = 2048
audio_file = wave.open('0_phase.wav', 'r')
#audio_file = wave.open('/home/hoverbear/test.wav', 'r')
out_file = wave.open('q2_4.wav', 'w')

out_file.setparams(audio_file.getparams())
num_of_frames = buffer_size / (audio_file.getsampwidth() + audio_file.getnchannels())

while True:
    frames = audio_file.readframes(int(num_of_frames))
    if not frames:
        break
    working_set = numpy.fromstring(frames, numpy.int16)
    fft_set = fft(working_set) / len(working_set)

    for i in range(0, len(fft_set)):
        fft_set[i] = (fft_set[i].real) + (fft_set[i].imag + numpy.random.random() * 1000)*1j

    inverse_fft_set = ifft(fft_set) * len(working_set)
    plt.figure()
    plt.plot(range(0, len(inverse_fft_set)), [i.real for i in inverse_fft_set])
    plt.plot(range(0, len(inverse_fft_set)), [i.imag for i in inverse_fft_set])
    show()
    for i in inverse_fft_set:
        packed_value = struct.pack('h', int16(i.real))
        out_file.writeframes(packed_value)

out_file.close()
```



The resulting audio has been complicated.

Q2.5
PASS

Q2.6
PASS