In [1]:
```
#library and imports
import sys
import math
import struct
import wave
import random
import argparse
import numpy as np
import IPython.display as ipd
import matplotlib.pyplot as plt

from pylab import *
from itertools import *
from struct import pack
```

In [2]:
```
# sin wave generator
def generate_sin (freq, duration, srate = 44100.0, amp = 1.0, phase = 0.0):
    t = np.linspace(0, duration, int(srate * duration)) #jidaoji fenjifen
    data = amp*np.sin(2*np.pi*freq*t + phase)
    return data
```

In [3]:

```
#Q1.1 (10pts) (*) Write two functions peakAmplitude and peakRMS
#that take as input an array of audio samples and return the peak amplitude
#and the RMS amplitude of the input.

#peak amplitude function
def peak_amplitude(data):
    return np.max(data)

#root mean square amp function
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)

#dot product amplitude function
def dot_amplitude(data1, data2):
    dot_product = np.dot(data1, data2)
    return 2* (dot_product / len(data1))


#Answer:
F200 = generate_sin(200, 0.5, amp=0.5)
print(peak_amplitude(F200))
print(rms_amplitude(F200))
F440 = generate_sin(440, 0.5, amp=0.5)
print(peak_amplitude(F440))
print(rms_amplitude(F440))
F500 = generate_sin(500, 0.5, amp=0.5)
print(peak_amplitude(F500))
print(rms_amplitude(F500))
F200 = generate_sin(200, 0.5, amp=1.0)
print(peak_amplitude(F200))
print(rms_amplitude(F200))
F440 = generate_sin(440, 0.5, amp=1.0)
print(peak_amplitude(F440))
print(rms_amplitude(F440))
F500 = generate_sin(500, 0.5, amp=1.0)
print(peak_amplitude(F500))
print(rms_amplitude(F500))
F200 = generate_sin(200, 0.5, amp=2.0)
print(peak_amplitude(F200))
print(rms_amplitude(F200))
F440 = generate_sin(440, 0.5, amp=2.0)
print(peak_amplitude(F440))
print(rms_amplitude(F440))
F500 = generate_sin(500, 0.5, amp=2.0)
print(peak_amplitude(F500))
print(rms_amplitude(F500))
F200 = generate_sin(200, 0.5, amp=3.0)
print(peak_amplitude(F200))
print(rms_amplitude(F200))
F440 = generate_sin(440, 0.5, amp=3.0)
print(peak_amplitude(F440))
print(rms_amplitude(F440))
F500 = generate_sin(500, 0.5, amp=3.0)
```

```
print(peak_amplitude(F500))
print(rms_amplitude(F500))
```

```
0.499999998731
0.499988662003
0.499999998731
0.499988662003
0.499999998731
0.499988662003
0.999999997462
0.999977324006
0.999999997462
0.999977324006
0.999999997462
0.999977324006
1.99999999492
1.99995464801
1.99999999492
1.99995464801
1.99999999492
1.99995464801
2.99999999239
2.99993197202
2.99999999239
2.99993197202
2.99999999239
2.99993197202
```

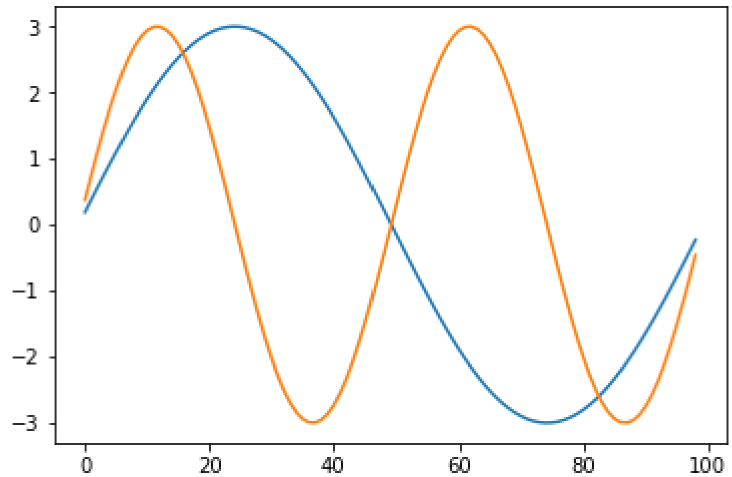
In [4]:

```
#Just play with frequencies
srate = 44100
freq1 = 440
freq2 = 4951
freq3 = 4186
freq4 = 587
freq5 = 659
freq6 = 698
freq7 = 784
freq8 = 880

data1 = generate_sin(freq1, 0.5, amp=3.0)
data2 = generate_sin(freq2, 0.5, amp=3.0)
data3 = generate_sin(freq3, 0.5, amp=3.0)
data4 = generate_sin(freq4, 0.5, amp=3.0)
data5 = generate_sin(freq5, 0.5, amp=3.0)
data6 = generate_sin(freq6, 0.5, amp=3.0)
data7 = generate_sin(freq7, 0.5, amp=3.0)
data8 = generate_sin(freq8, 0.5, amp=3.0)

#Stack data horizontally
data = np.hstack([data1,data2,data3,data4,data5,data6,data7,data8])
ipd.Audio(data, rate = srate)
```

Out[4]:

0:00 / 0:04

In [5]:
```
# plot sin waves with frequency 440 and 880
plt.plot(data1[1:int(44100/440)])
plt.plot(data8[1:int(44100/440)])
show()
```



In [6]:
```
#Q1.2 (5pts) (*) Write a function that makes a mixture of three harmonically
#related sinusoids with frequencies f, 2f, 3f with user provided
#amplitudes and phases

#return sum of sine wave with 1f, 2f, 3f frequency and different amplitude
def sum_harmonics(amp1, amp2, amp3, freq, phase1, phase2, phase3):

    y1 = generate_sin (1*freq, 1, amp = amp1, phase = phase1 )
    y2 = generate_sin (2*freq, 1, amp = amp2, phase = phase2 )
    y3 = generate_sin (3*freq, 1, amp = amp3, phase = phase3 )
    sum_y = [ y1[x]+y2[x]+y3[x] for x in range(len(y1))]

    return sum_y
```

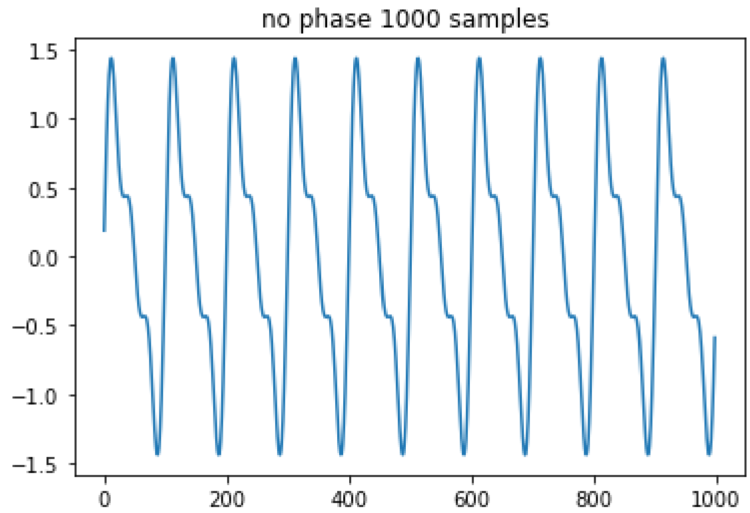
In [7]:
```
# no phase signal
wv_noPhase = sum_harmonics(1.0, 0.5, 0.33, 440, 0, 0, 0)

# generate random phase
freq_test = 440
rand1 = int(np.random.random()* int(srate/freq_test/1))
rand2 = int(np.random.random()* int(srate/freq_test/2))
rand3 = int(np.random.random()* int(srate/freq_test/3))

#random phase signal
wv_randomPhase = sum_harmonics(1.0, 0.5, 0.33, freq_test, rand1, rand2, rand3)

# stack phase and none phase signal horizontally
data = np.hstack([wv_noPhase,wv_randomPhase])
ipd.Audio(data, rate = srate)
```

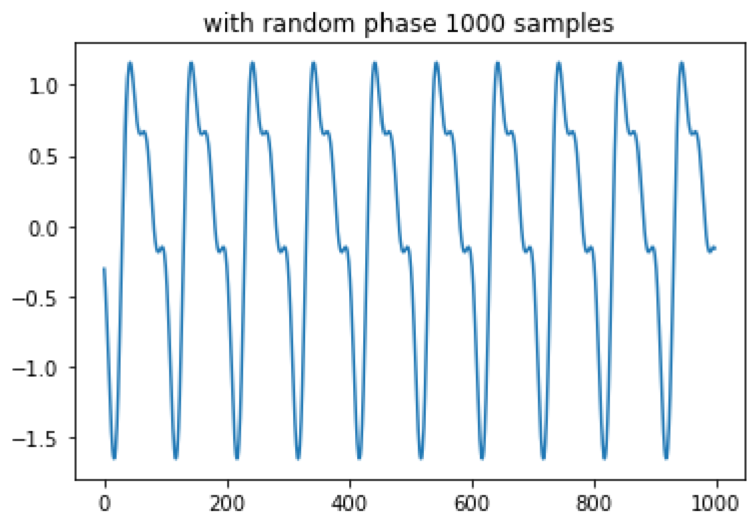
Out[7]:

0:00 / 0:02

In [8]:

```
# plot phase and none phase signal
plt.plot(wv_noPhase[1:int(1000)])
title("no phase 1000 samples ")
figure()

plt.plot(wv_randomPhase[1:int(1000)])
title("with random phase 1000 samples")
figure()

show()
```






```
<matplotlib.figure.Figure at 0x74c0f70>
```

In [ ]:

```
#Q1.3 (5pts) (**) Using the function from Q1b generate one second
#of audio in .wav format for a mixture with f = 440Hz.
wv_randomPhase = sum_harmonics(1.0, 1.0, 0.33, 550, rand1, rand2, rand3)

plt.plot(wv_randomPhase[1:int(1000)])
title(" 1000 samples ")
figure()

p = rms_amplitude(wv_randomPhase)
new = [x/p for x in wv_randomPhase]


def save_wav(file_name):

    wav_file=wave.open(file_name,"w")

    nchannels = 1
    sampwidth = 2

    nframes = len(new)
    comptype = "NONE"
    compname = "not compressed"
    wav_file.setparams((nchannels, sampwidth, srate, nframes, comptype, compname))

    for sample in new:
        wav_file.writeframes(struct.pack('h', int(sample)*(2**15-1)))


    wav_file.close()

    return

save_wav("out.wav")
```

In [10]:

```
# SNR_db = 10 logbase10 (SNR)
# SNR_db = 10*logbase10[ (signal_amp/noise_amp)**2 ]
# SNR_db = 20*logbase10[ (signal_amp/noise_amp) ]
# samples = numpy.random.normal(mean, std, size=num_samples)

#Q1.4 (5pts) (*) Read about the concept of signal to noise ratio (SNR).
#Write a function that takes as input a SNR in decibels (dB) and a
#frequency in Hz in order to generate a mixture of white noise and a
#440 sine wave


def generate_noise_signal(snr_ratio, freq):
    #snr_ratio /= 20
    pure_signal = generate_sin(freq, 2)

    # SNR_db = 20*logbase10[ (signal_amp/noise_amp) ]
    signal_amp = 1
    noise_amp = 1/((snr_ratio/20)**2)

    #print(noise_amp)
    noise = np.random.normal(0,0.5,len(pure_signal))*noise_amp
    noise_signal = pure_signal + noise

    return noise_signal

pure_signal = generate_sin(440, 2)
noi1 = generate_noise_signal(120, 440)
noi2 = generate_noise_signal(90, 440)
noi3 = generate_noise_signal(70, 440)
noi4 = generate_noise_signal(50, 440)
noi5 = generate_noise_signal(30, 440)
noi6 = generate_noise_signal(10, 440)

plt.plot(pure_signal[1:int(100)])
title(" pure signal ")
figure()
plt.plot(noi1[1:int(100)])
title(" 440hz 120db ")
figure()
plt.plot(noi2[1:int(100)])
title(" 440hz 90db")
figure()
plt.plot(noi3[1:int(100)])
title(" 440hz 70db ")
figure()
plt.plot(noi4[1:int(100)])
title(" 440hz 50db ")
figure()
plt.plot(noi5[1:int(100)])
title(" 440hz 30db")
figure()
plt.plot(noi6[1:int(100)])
title(" 440hz 10db ")
figure()
zzz = np.hstack([pure_signal,noi1,noi2,noi3,noi4,noi5,noi6])
ipd.Audio(zzz, rate = srate)
```

Out[10]:

0:00 / 0:14

In [11]:

```
#Q1.5 (5pts) (**) Consider another way of estimating the amplitude
#of a sinusoid signal when you know the frequency.

freq1 = 440
duration1 = 0.1
srate1 = 4096
amp1 = 4.0
phase1 = 0.0
amp2 = 4.0


#unknown amplitude signal
wave_data1 = generate_sin(freq1,duration1,srate1,amp1,phase1)
#unit amplitude signal with same frequency
wave_data2 = generate_sin(freq1,duration1,srate1,1.0,phase1)
wave_data3 = generate_sin(freq1,duration1,srate1,amp2,phase1)
# dot of two signals finds amplitude
doted_amp1 = dot_amplitude(wave_data1,wave_data3)
doted_amp2 = dot_amplitude(wave_data1,wave_data2)

print(doted_amp1)
print(doted_amp2)
#smaller the amplitude, more accuracy of estimation

#Answer: I observed when a wave takes inner product with another wave with
#same frequency and phase it will produce a magnified version of the wave
#and amplitude is increase by a power of 2
#when we take dot product with same f, phi, and unit amplitude, we can get
#the amplitude of the wave
```

```
15.9608801956
3.9902200489
```

In [12]:
```
#Q1.6 (5pts) (***) Determine by listening 5 resonable values of SNR
#for a 440Hz sinusoid mixed with noise (from noise barely perceptible
#to sinusoid hard to hear in the noise) and create the corresponding
#signals
pure_signal = generate_sin(440, 2)
noi1 = generate_noise_signal(120, 440)
noi2 = generate_noise_signal(90, 440)
noi3 = generate_noise_signal(70, 440)
noi4 = generate_noise_signal(50, 440)
noi5 = generate_noise_signal(30, 440)
noi6 = generate_noise_signal(10, 440)

x1 = peak_amplitude(noi1)
y1 = rms_amplitude(noi1)
z1 = dot_amplitude(noi1,pure_signal)

x2 = peak_amplitude(noi2)
y2 = rms_amplitude(noi2)
z2 = dot_amplitude(noi2,pure_signal)

x3 = peak_amplitude(noi3)
y3 = rms_amplitude(noi3)
z3 = dot_amplitude(noi3,pure_signal)

x4 = peak_amplitude(noi4)
y4 = rms_amplitude(noi4)
z4 = dot_amplitude(noi4,pure_signal)

x5 = peak_amplitude(noi5)
y5 = rms_amplitude(noi5)
z5 = dot_amplitude(noi5,pure_signal)

x6 = peak_amplitude(noi6)
y6 = rms_amplitude(noi6)
z6 = dot_amplitude(noi6,pure_signal)

print([x1,y1,z1])
print([x2,y2,z2])
print([x3,y3,z3])
print([x4,y4,z4])
print([x5,y5,z5])
print([x6,y6,z6])

# by observation: the pure signal has amplitude of 1, dot product amplitude is more rob
ust than the other two methods
# dot better than > rms better than > peak amp
```

```
[1.0416431422270003, 1.0001789378418946, 0.99997977252082615]
[1.0778153142668798, 1.0006911281190065, 1.0000751198958033]
[1.1450149764855215, 1.0013542417954702, 0.99969089162919889]
[1.3128436097808662, 1.0061267449804669, 0.99972602000447064]
[1.8792242040200222, 1.0480070967934885, 0.99991624976462867]
[9.1353972366043585, 3.0050250238774034, 0.99436143189876114]
```

In [13]:

```
#Q1.7 (10pts) (**) Consider a mixture of 3 harmonically related sinusoids
#as the ones you created. What you would like to devise is a
#process to estimate the amplitudes of each sine wave assuming that you
#know the frequencies that the mixture is composed of.

mixed_wave_signal1 = sum_harmonics(5.0, 0.5, 0.33, 440, 111, 222, 333)

def estimate_amplitude(test_freq, mix_wv):
    max1 = 0.0
    max2 = 0.0
    max3 = 0.0


    for x in range(int(44100/test_freq)):
        freq1_signal = generate_sin(test_freq,1,44100,1.0,x)
        doted_amp1 = dot_amplitude(mix_wv, freq1_signal)
        if doted_amp1 > max1:
            max1 = doted_amp1

    for x in range(int(44100/2/test_freq)):
        freq2_signal = generate_sin(2*test_freq,1,44100,1.0,x)
        doted_amp2 = dot_amplitude(mix_wv, freq2_signal)
        if doted_amp2 > max2:
            max2 = doted_amp2

    for x in range(int(44100/3/test_freq)):
        freq3_signal = generate_sin(3*test_freq,1,44100,1.0,x)
        doted_amp3 = dot_amplitude(mix_wv, freq3_signal)
        if doted_amp3 > max3:
            max3 = doted_amp3

    return [max1,max2,max3]

#frequency included in mixture
amps1 = estimate_amplitude(440, mixed_wave_signal1)
#frequency smaller than mixture
amps2 = estimate_amplitude(300, mixed_wave_signal1)
#frequency greater than mixture
amps3 = estimate_amplitude(250, mixed_wave_signal1)

print(amps1)
print(amps2)
print(amps3)

#plt.plot(mixed_wave_signal1[0:int(44100)+1])
#plt.axis([0,1000,3,-3])
#show()

#Answer: when I take inner of frequency that is not in the mixture, the
# amplitude is very close to zero
# if we take the dot frequency that is in the mixture
# we can get number that is not close to zero
# so if we want to find the amplitude of sine waves,
# we can take multiple shifts to record the frequency when the amplitudes are not close
 zeroes
# we will implement dot product inside the function to do cross corrolation
# since dot product amplitude estimation is very unlikely to be wrong when there are no
```

```
ises
# so we can almost trust the results
```

```
[4.999254157066952, 0.49888251859082311, 0.32997967790785854]
[0.00017646960093374036, 0.00017646960093172103, 0.00017646960093467104]
[0.00017646960093253023, 0.00017646960091062034, 0.00017646960093285798]
```

In [14]:

```
#Q1.8 (5pts) (***) Now let’ s make the scenario a little bit more challenging.
#We still know the frequencies of the mixture but we want to
#estimate not only the amplitudes but also the phases.

def estimate_amplitude(test_freq, mix_wv):
    maxAmp1 = 0.0
    maxAmp2 = 0.0
    maxAmp3 = 0.0
    maxAngle1 = 0.0
    maxAngle2 = 0.0
    maxAngle3 = 0.0

    angle = 0.0
    for x in range(int(44100/test_freq)):
        freq1_signal = generate_sin(test_freq,1,44100,1.0,x)
        doted_amp1 = dot_amplitude(mix_wv, freq1_signal)
        if doted_amp1 >= maxAmp1:
            maxAmp1 = doted_amp1
            maxAngle1 = angle
        angle += 1

    angle = 0.0
    for x in range(int(44100/2/test_freq)):
        freq2_signal = generate_sin(2*test_freq,1,44100,1.0,x)
        doted_amp2 = dot_amplitude(mix_wv, freq2_signal)

        if doted_amp2 >= maxAmp2:
            maxAmp2 = doted_amp2
            maxAngle2 = angle
        angle += 1

    angle = 0.0
    for x in range(int(44100/3/test_freq)):
        freq3_signal = generate_sin(3*test_freq,1,44100,1.0,x)
        doted_amp3 = dot_amplitude(mix_wv, freq3_signal)
        if doted_amp3 >= maxAmp3:
            maxAmp3 = doted_amp3
            maxAngle3 = angle
        angle += 1

    return [maxAmp1,maxAmp2,maxAmp3,maxAngle1,maxAngle2,maxAngle3]

mixed_wave_signal2 = sum_harmonics(5.1, 2.5, 8.33, 312, 120, 43, 32)
ampsAndPhase1 = estimate_amplitude(312, mixed_wave_signal2)
mixed_wave_signal2 = sum_harmonics(2.3, 5.5, 0.3, 440, 90, 33, 10)
ampsAndPhase2 = estimate_amplitude(440, mixed_wave_signal2)
mixed_wave_signal2 = sum_harmonics(3.1, 3.5, 1.23, 550, 42, 22, 1)
ampsAndPhase3 = estimate_amplitude(550, mixed_wave_signal2)
print(ampsAndPhase1)
print(ampsAndPhase2)
print(ampsAndPhase3)
#Answer:
```

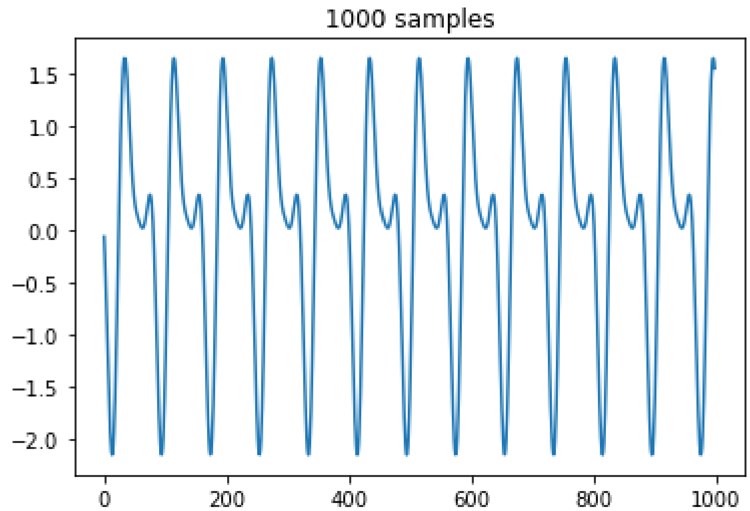
```
[5.1000285208003113, 2.4997367791241047, 8.329948031592405, 120.0, 43.0, 32.0]
[2.3002475678652852, 5.5002105147233245, 0.29981080831165974, 90.0, 33.0, 10.0]
[3.1000060690039426, 3.4999213724041289, 1.2299019982034951, 42.0, 22.0, 1.0]
```

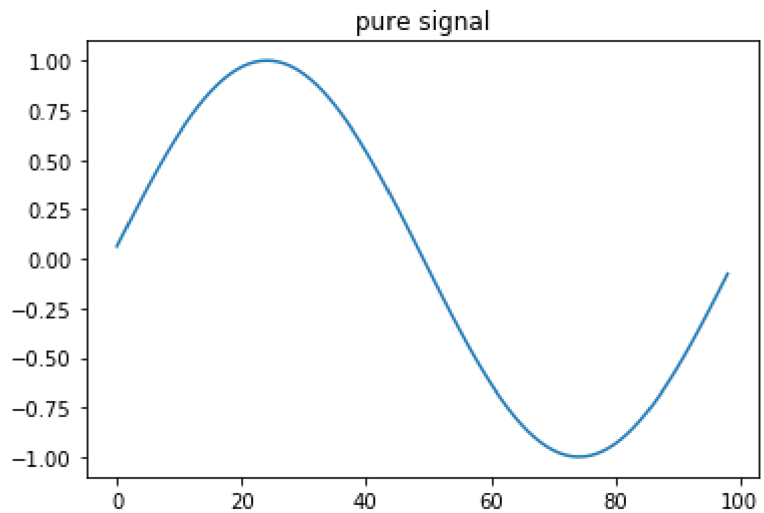
In [ ]:

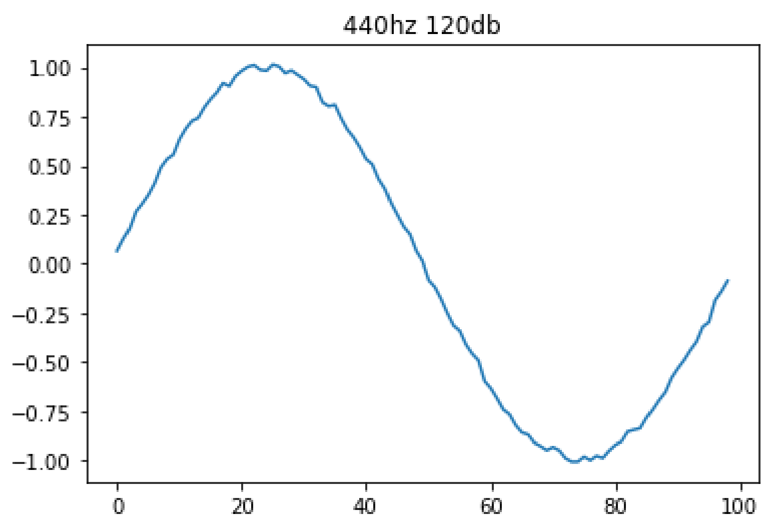
```
#Q2.1 (10pt) (*) Write code to read/write data from a .wav file (you
#can use a library) in buffers of 2048. Verify that your code can read,
#apply simple processing (like applying a simple gain) and write audio
#files correctly.

SIZE = 2048
IN = wave.open('www.wav', 'r')
INsw = IN.getsampwidth()
INch = IN.getnchannels()
OUT1 = wave.open('xxx.wav', 'w')
OUT2 = wave.open('yyy.wav', 'w')
OUT1.setparams(IN.getparams())
OUT2.setparams(IN.getparams())

framenum = SIZE / (INsw + INch)

while True:
    frame_1= IN.readframes(int(framenum))
    frame_2 = bytearray(frame_1)
    frame_3 = bytearray(frame_1)
    for i in range(0, len(frame_2)):
        if frame_2[i]-10 > 0:
            frame_2[i] -= 10

    for i in range(0, len(frame_3)):
        if frame_3[i]+10 < 256:
            frame_3[i] += 10

    OUT1.writeframes(frame_2)
    OUT2.writeframes(frame_3)
```

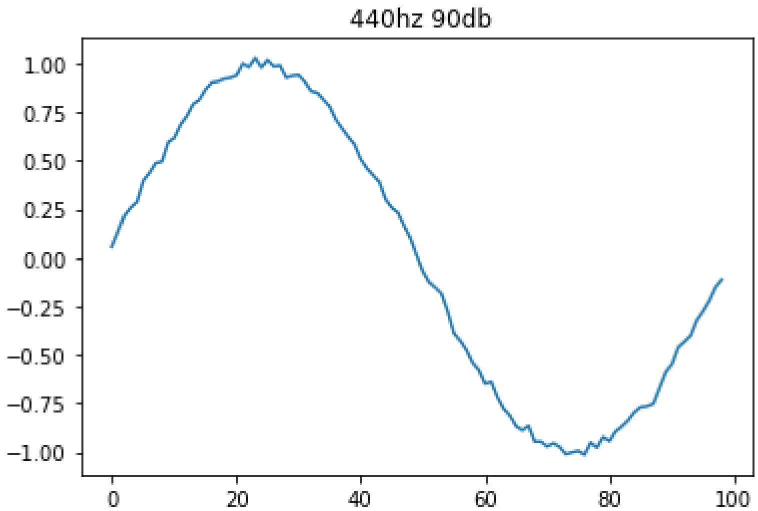
In [16]:

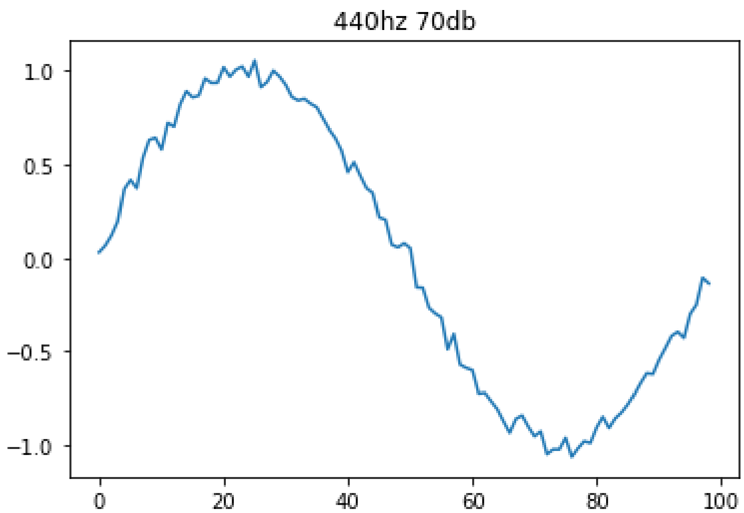
```
#Q2.2 (10pt) (*) Using any library or implementation of the Fast
#Fourier Transform for your programming language calculate the frequency
#domain complex spectrum of the 3 component mixture signal
#from question 1. Plot the magnitude spectrum.

fft_size = 4096
data = sum_harmonics(0.5, 0.7, 0.59, 440, 0, 0, 0)
#plt.plot(data[0:fft_size])
#plt.title('original data plot')
#plt.figure()

complex_spectrum = np.fft.fft(data[0:fft_size])
magnitude_spectrum = np.abs(complex_spectrum)
phase_spectrum = np.angle(complex_spectrum)
plt.plot(magnitude_spectrum)
plt.title('magitude spec plot 1')
plt.figure()

half_magnitude_spectrum = magnitude_spectrum[0: int(len(magnitude_spectrum)/2)]
plt.plot(half_magnitude_spectrum)
plt.title('half mag spec plot')
plt.figure()

plt.plot(phase_spectrum)
plt.title('phase spec plot')
plt.figure()

#magnitude_spectrum = np.zeros(len(magnitude_spectrum))
#fft_bin = 100
#fft_bin2 = 200
#magnitude_spectrum[fft_bin] = 1
#magnitude_spectrum[fft_bin2] = 1
#magnitude_spectrum[len(magnitude_spectrum)-fft_bin] = 1
#magnitude_spectrum[len(magnitude_spectrum)-fft_bin2] = 1
#plt.plot(magnitude_spectrum[90:110])
#plt.title('magitude spec plot 2')
#plt.figure()

#real_spectrum = magnitude_spectrum * np.cos(phase_spectrum)
#imag_spectrum = magnitude_spectrum * np.sin(phase_spectrum)
#back_to_time_domain = np.fft.ifft(real_spectrum + 1j * imag_spectrum)
#plt.plot(np.real(back_to_time_domain[1:100]))#bask to time domain
#plt.title('bask to time domain')
#plt.figure()

show()
```

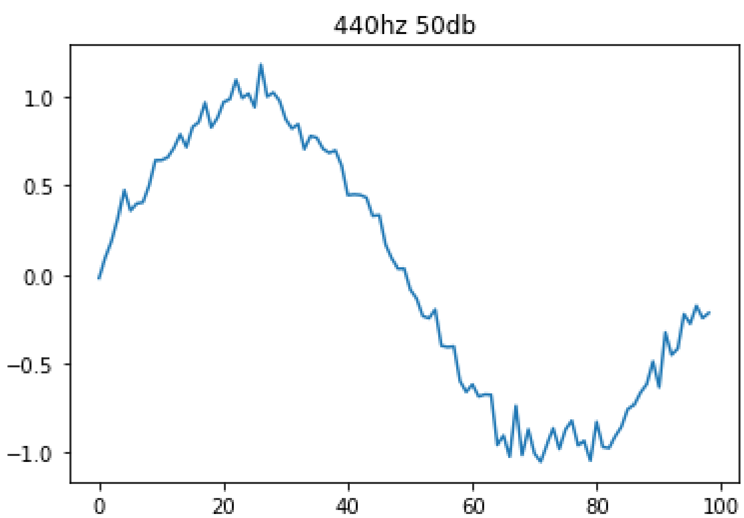

1000 samples
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
200
400
600
800
1000

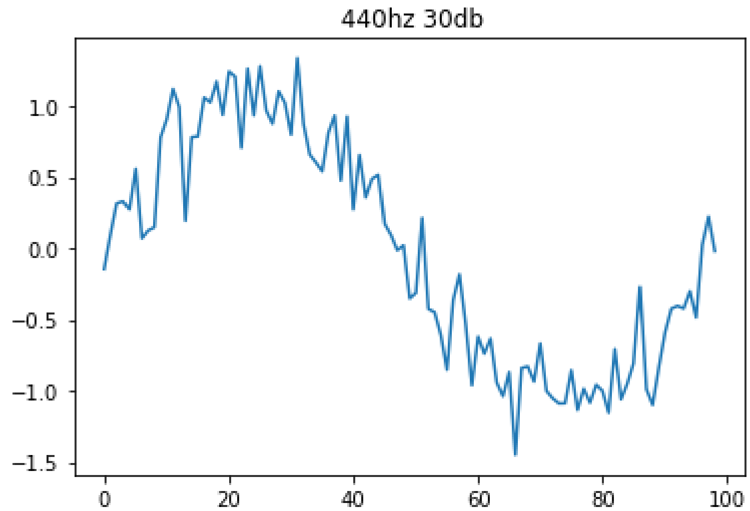

pure signal
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100

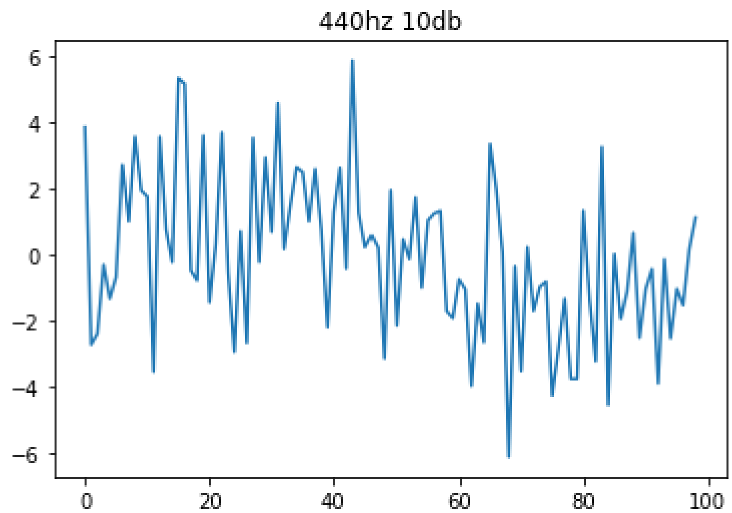

440hz 120db
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100

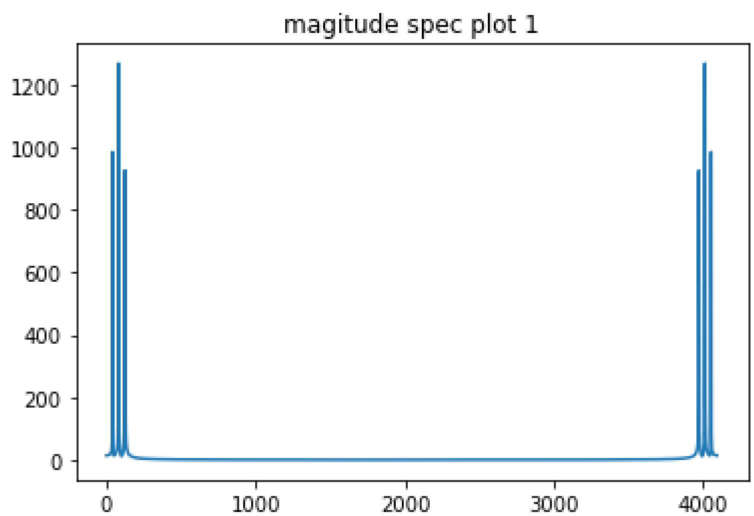

440hz 90db

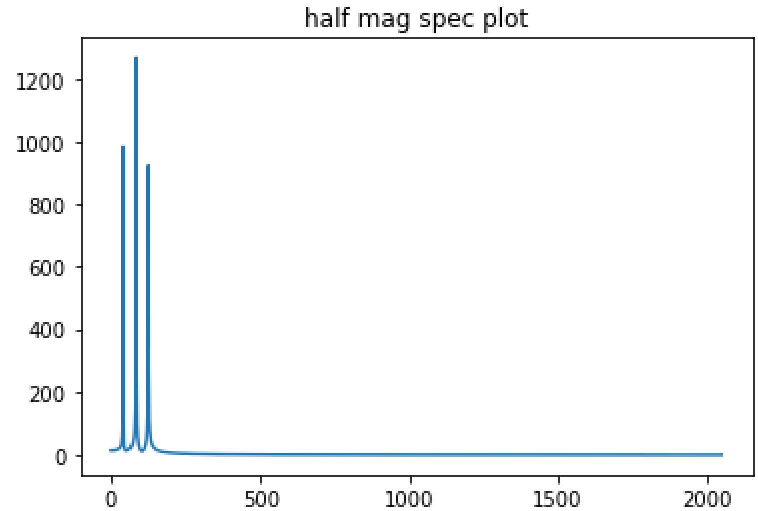

440hz 70db

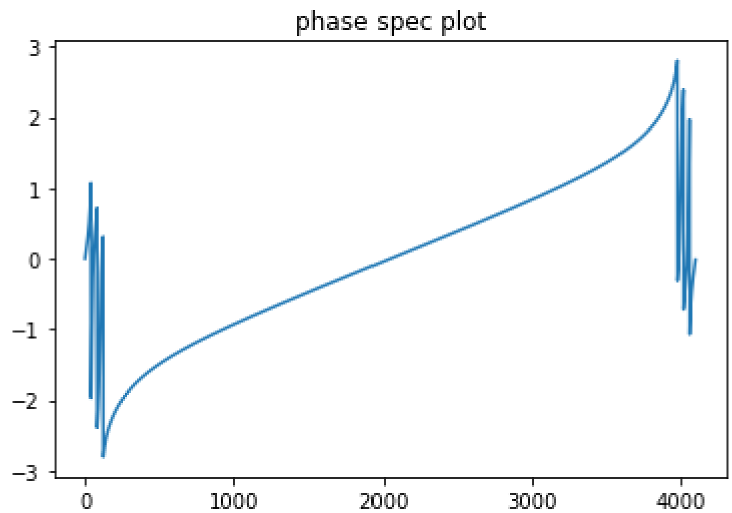

440hz 50db


440hz 30db
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
20
40
60
80
100


440hz 10db
6
4
2
0
−2
−4
−6
0
20
40
60
80
100


magitude spec plot 1
1200
1000
800
600
400
200
0
0
1000
2000
3000
4000






```
<matplotlib.figure.Figure at 0x9f8cbd0>
```

In [ ]:
```
#Q2.3 (10pt) (*) Using a programming language of your choice write
#code to directly compute the Discrete Fourier Transform (DFT) of an
#input array.

####!!!!!!see bottom or attachment!!!!!!!################################

#FINISHED dont forget import
```

In [ ]:
```
#Q2.4 (5pt) (**)
#Modify the code that reads/writes data in buffers so that each buffer is
#converted to a frequency domain complex spectrum. Compute magnitude
#and phase spectrum for each buffer and plot an example for each
#type.

SIZE = 2048
IN = wave.open('www.wav', 'r')
INsw = IN.getsampwidth()
INch = IN.getnchannels()
OUT = wave.open('xxx.wav', 'w')
OUT.setparams(IN.getparams())

framenum = SIZE / (INsw + INch)

while True:
    frame_1= IN.readframes(int(framenum))
    frame_2 = bytearray(frame_1)

    for i in range(0, len(frame_2)):
  ################????????###


    OUT.writeframes(frame_2)
```

In [ ]:
```
#Q2.5 (5pt) (**) Consider a sinusoid with frequency equal to one of
#the DFT bins (the array indices of the mangitude spectrum). What
#does the magnitude spectrum looks like for this input ? How does it
#change when you change the amplitude of the input sinusoid ?How does
#it change when you change the phase ?
????
```

In [ ]:
```
#Q2.6 (10pt) (***) Plot the time domain waveforms of the two basis
#signals (the cosine and sine corresponding to the frequency of the
#bin) as well as the time domain signal corresponding to the point-wise
#multiplication of the input to these two basis functions (the part of the
#inner-product before the summation).

????
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```