COMMUNICATION SYSTEMS LAB 5 DATE-21/9/2021

NAME – JAGRIT LODHA

ID – 2019A3PS0165P

SECTION – P4

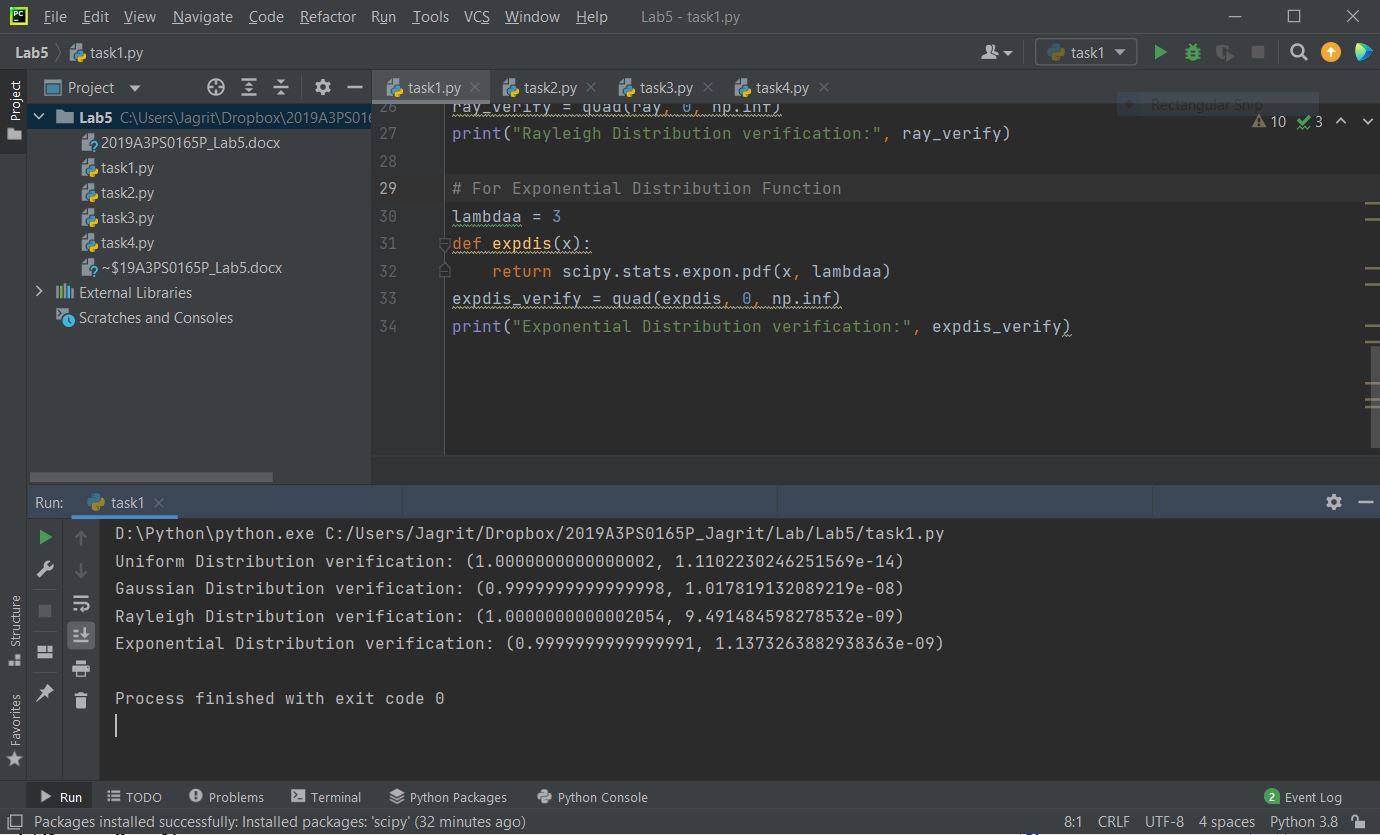
PYTHON

TASK 1 –

CODE:

import scipy  
from scipy import stats  
from scipy.integrate import quad  
import numpy as np  
  
# For uniform distribution function  
a = 0  
b = 5  
def uniform(x):  
 return 1 / (b - a)  
uniform\_verify = quad(uniform, a, b)  
print("Uniform Distribution verification:", uniform\_verify)  
  
# For Gaussian distribution function  
mu = 0  
sigma = 1  
def gauss(x):  
 return scipy.stats.norm.pdf(x, mu, sigma)  
gaussian\_verify = quad(gauss, -np.inf, np.inf)  
print("Gaussian Distribution verification:", gaussian\_verify)  
  
# For Rayleigh Distribution Function  
alpha = 2  
def ray(x):  
 return scipy.stats.rayleigh.pdf(x, alpha)  
ray\_verify = quad(ray, 0, np.inf)  
print("Rayleigh Distribution verification:", ray\_verify)  
  
# For Exponential Distribution Function  
lambdaa = 3  
def expdis(x):  
 return scipy.stats.expon.pdf(x, lambdaa)  
expdis\_verify = quad(expdis, 0, np.inf)  
print("Exponential Distribution verification:", expdis\_verify)

* We use the quad function to integrate within the given limits and verify the PDFs of the random variables.
* We use the scipy.stats package to generate the Gaussian, Rayleigh and Exponential Distributions as writing their expressions in Python would have been difficult.



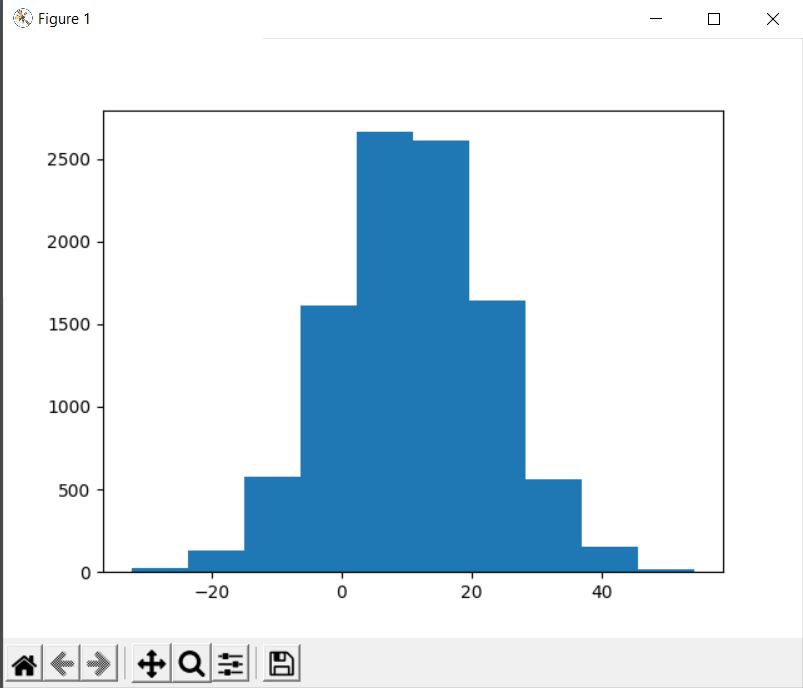
NOTE –

* We can verify from the command window that the real value of integration for all 4 distributions is 1, which is as required.
* The imaginary part is almost negligible, and is due to calculation errors duo to infinity.
* We integrate from a to b for Uniform, from -inf to inf for Gaussian and from 0 to inf for Rayleigh and Gaussian.

TASK 2 –

CODE for Part a – To verify the generated samples using histogram :

import numpy as np  
import matplotlib.pyplot as plt  
sigma = 12 # Sum of last three digits = 1+6+5 = 12  
mu = 11 # Sum of last two digits = 6+5 = 11  
x = sigma\*np.random.randn(10000) + mu  
plt.hist(x)  
plt.show()

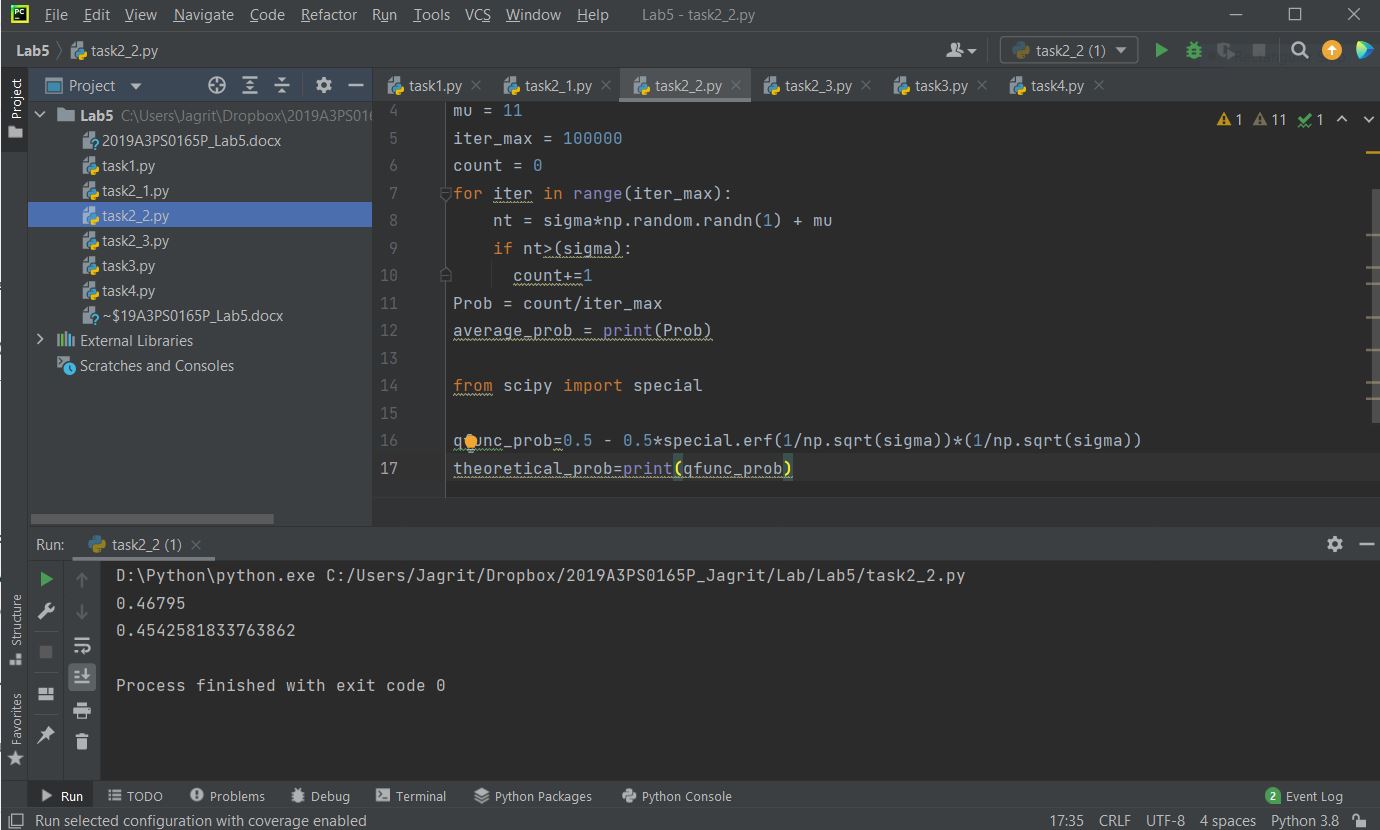


So we can clearly see how the peak of the Normal Distribution is at the mean (mu=11) and most of the values are concentrated within the first 1 or 2 confidence intervals (sigma square = 12)

Code for Part b:

import numpy as np  
import matplotlib.pyplot as plt  
sigma = 12  
mu = 11  
iter\_max = 100000  
count = 0  
for iter in range(iter\_max):  
 nt = sigma\*np.random.randn(1) + mu  
 if nt>(sigma):  
 count+=1  
Prob = count/iter\_max  
average\_prob = print(Prob)  
  
from scipy import special  
  
qfunc\_prob=0.5 - 0.5\*special.erf(1/np.sqrt(sigma))\*(1/np.sqrt(sigma))  
theoretical\_prob=print(qfunc\_prob)

Here are the obtained probabilities –

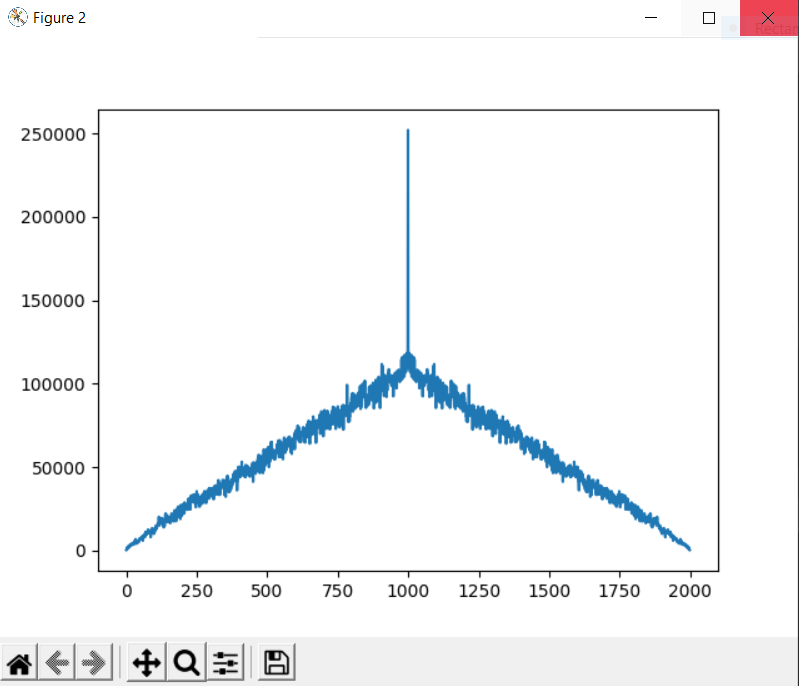


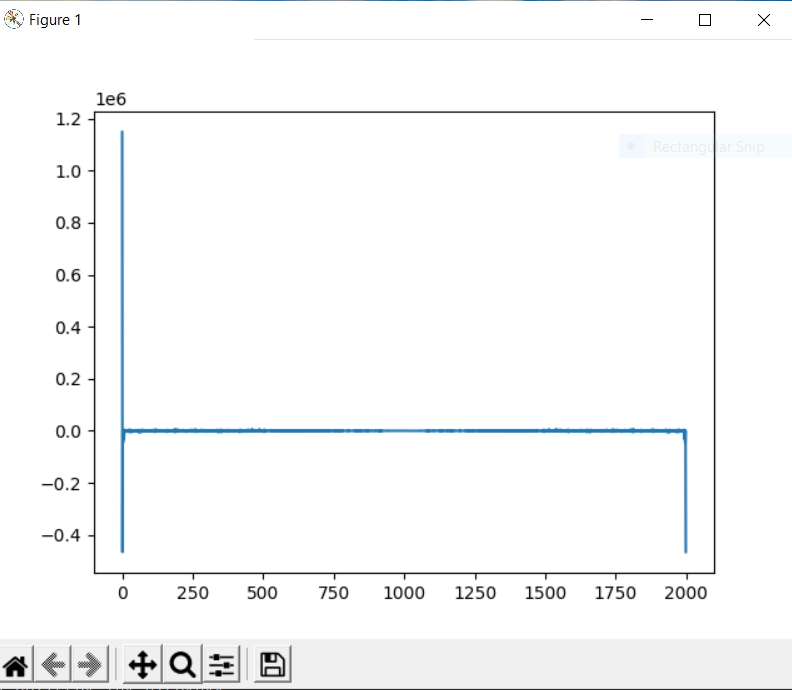
There is a slight error due to computational constraints.

TASK 2 –

Code for Part c:

import numpy as np  
import matplotlib.pyplot as plt  
sigma = 12  
mu = 11  
nt = sigma\*np.random.randn(1000) + mu  
ac = np.correlate(nt, nt, mode = 'full')  
print(ac)  
plt.figure(2)  
plt. plot(ac)  
plt.show()





This is the code added for finding PSD (which is Fourier Transform of the AutoCorrelation Function)

import numpy as np  
import matplotlib.pyplot as plt  
from numpy.fft import fft  
  
sigma = 12  
mu = 11  
nt = sigma\*np.random.randn(1000) + mu  
ac = np.correlate(nt, nt, mode = 'full')  
print(ac)  
af = fft(ac) / 100  
plt.figure(2)  
plt. plot(ac)  
plt.figure(1)  
plt. plot(af)  
plt.show()

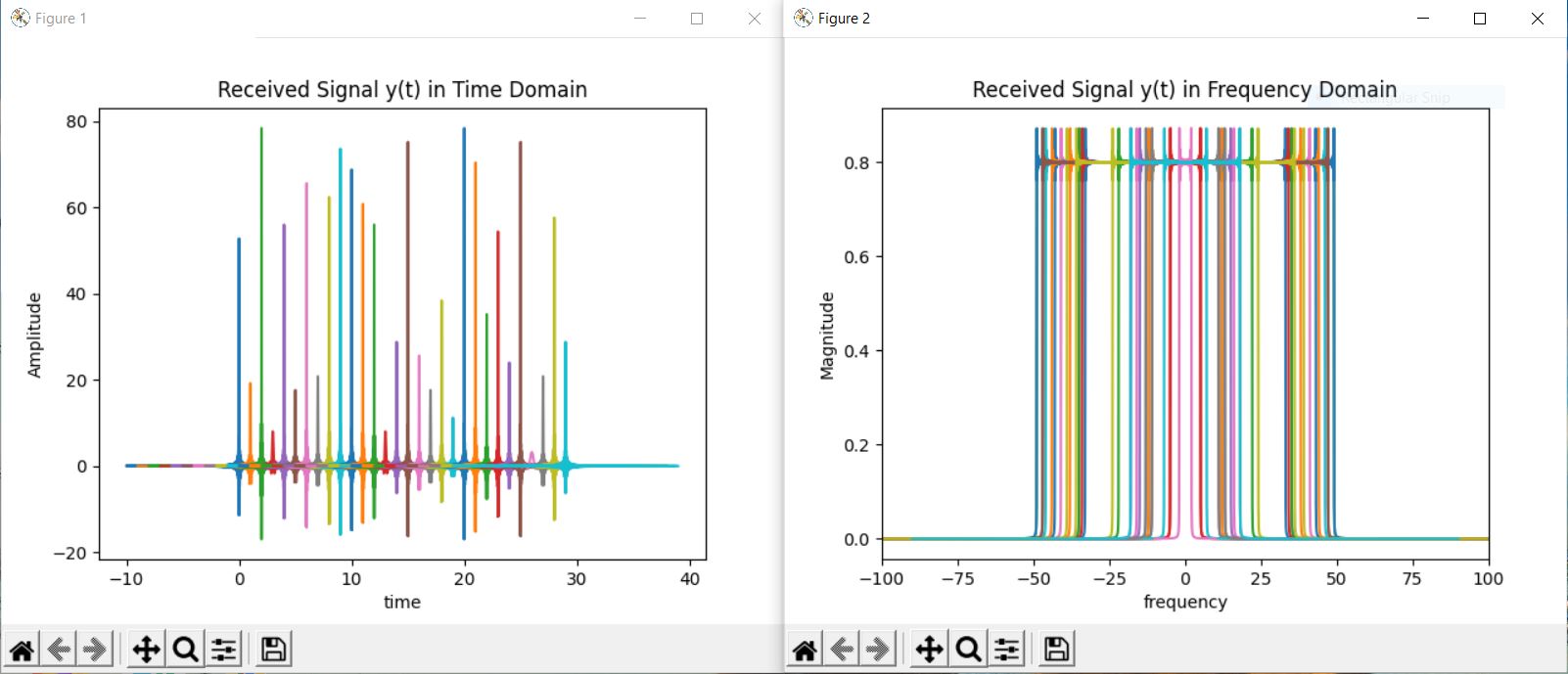
TASK 3 –

CODE:

import numpy as np  
from scipy import signal  
import matplotlib.pyplot as plt  
from numpy.fft import fft  
  
a = 0.8 # a<1 selected randomly for channel attenuation  
sigma = 1 # sigma for n\_t  
for T in range(30):  
 start\_time=-10  
 stop\_time=10  
 N = np.random.randint(1,50)  
 fm = N # Maximum frequency component in Hertz for the given spectrum  
 fs=10\*fm  
 ts=1/fs  
 time = np.arange(start\_time, stop\_time, ts)  
  
 # Message signal  
 m\_t = 2 \* N \* np.sinc(2 \* N \* time)  
  
 mf = fft(m\_t) / fs  
 N0 = len(mf)  
 freq\_axis = np.linspace(-fs / 2, fs / 2, N0) # sampling freq for freq domain plot  
 mf\_abs = abs(mf)  
 mf\_abs\_sorted = np.fft.fftshift(mf\_abs)  
  
 # Channel (although we are not using it directly)  
 h\_t = a \* signal.unit\_impulse(len(time), idx='mid')  
  
 # Noise signal  
 n\_t = sigma \* np.random.randn(len(time)) # for every iteration a new random Variable n(t) is created for that second  
  
 # Received Signal y(t):  
 y\_t = a\*m\_t +n\_t # Convolving with channel and adding noise  
 yf = fft(y\_t) / fs  
 N = len(yf)  
 freq\_axis\_y = np.linspace(-fs / 2, fs / 2, N) # sampling freq for freq domain plot  
 yf\_abs = abs(yf)  
 yf\_abs\_sorted = np.fft.fftshift(yf\_abs)  
  
 plt.figure(1)  
 plt.plot(time + T, y\_t)  
 plt.title('Received Signal y(t) in Time Domain')  
 plt.xlabel('time')  
 plt.ylabel('Amplitude')  
 plt.figure(2)  
 plt.plot(freq\_axis\_y, yf\_abs\_sorted)  
 plt.title('Received Signal y(t) in Frequency Domain')  
 plt.xlabel('frequency')  
 plt.ylabel('Magnitude')  
 plt.xlim(-100, 100)  
  
plt.pause(0.5)  
plt.show()

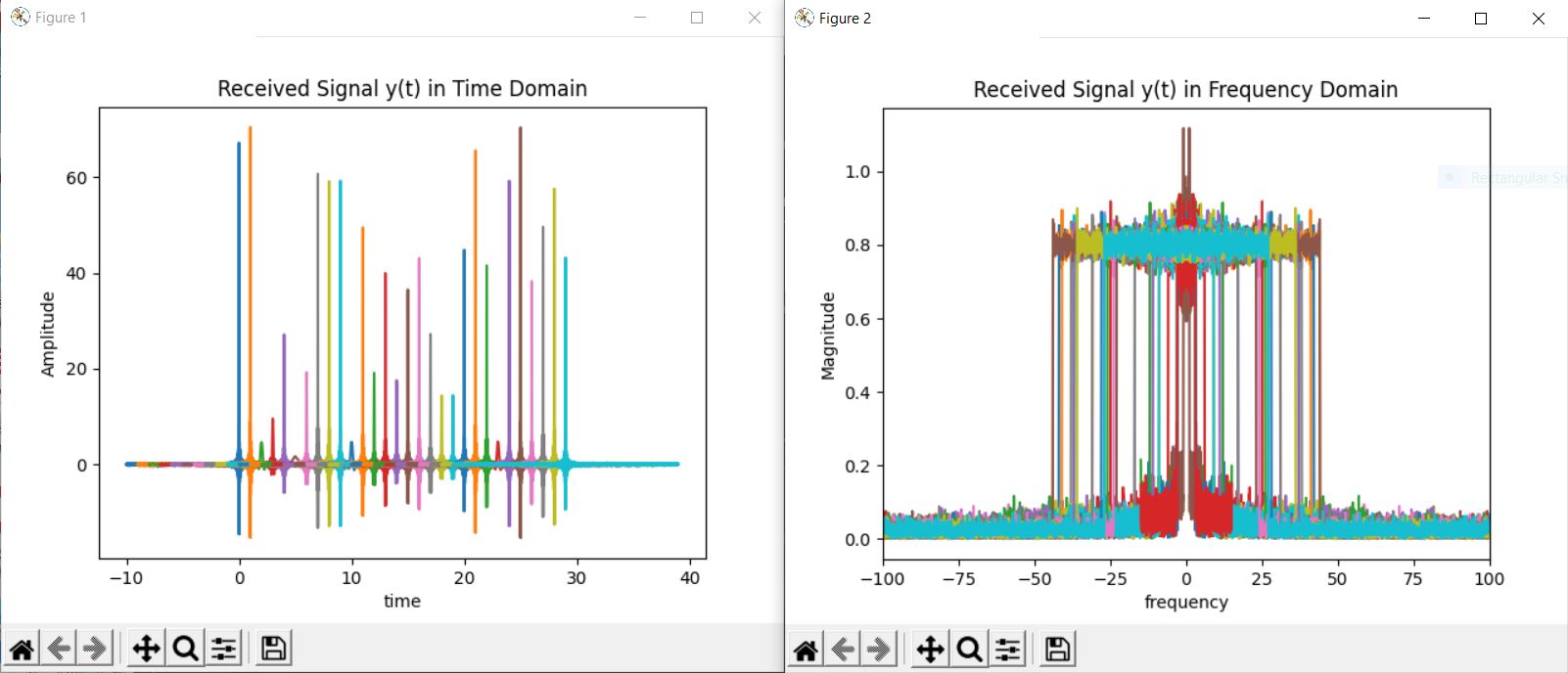
For all the cases, we take a=0.8 (channel attenuation)

1. When sigma is taken to be 0 (i.e., NO NOISE) –

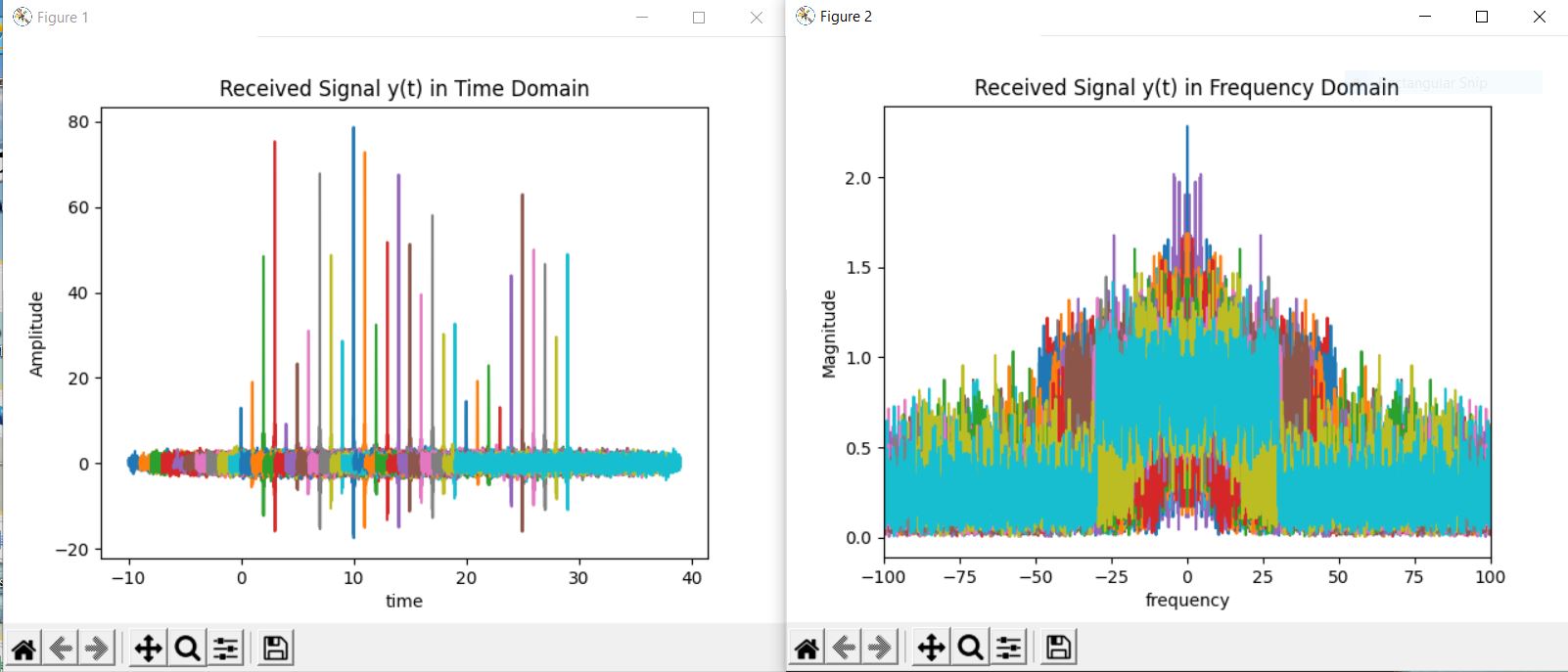


* The value of random N is taken to be from 1 to 50
* So, the bandwidth peaks are between -50 and 50
* The Magnitude is 0.8 because of the attenuation

2. When sigma = 0.1 –



3. When sigma = 1 –



* So it is clearly visible how the distortion increases with the increase in noise in the message signal.

TASK 4 –

import numpy as np  
from scipy import signal  
import matplotlib.pyplot as plt  
from numpy.fft import fft  
  
a = 0.8 # a<1 selected randomly for channel attenuation  
sigma = 1 # sigma for n\_t  
for T in range(30):  
 start\_time=-10  
 stop\_time=10  
 B = np.random.randint(1,50)  
 fm = NB # Maximum frequency component in Hertz for the given spectrum  
 fs=10\*fm  
 ts=1/fs  
 time = np.arange(start\_time, stop\_time, ts)  
  
 # Channel signal  
 h\_t = 2 \* B \* np.sinc(2 \* B \* time)  
  
 mf = fft(m\_t) / fs  
 N0 = len(mf)  
 freq\_axis = np.linspace(-fs / 2, fs / 2, N0) # sampling freq for freq domain plot  
 mf\_abs = abs(mf)  
 mf\_abs\_sorted = np.fft.fftshift(mf\_abs)  
  
 # Noise signal  
 n\_t = sigma \* np.random.randn(len(time)) # for every iteration a new random Variable n(t) is created for that second  
  
 # Received Signal y(t):  
 y\_t = a\*m\_t +n\_t # Convolving with channel and adding noise  
 yf = fft(y\_t) / fs  
 N = len(yf)  
 freq\_axis\_y = np.linspace(-fs / 2, fs / 2, N) # sampling freq for freq domain plot  
 yf\_abs = abs(yf)  
 yf\_abs\_sorted = np.fft.fftshift(yf\_abs)  
  
 plt.figure(1)  
 plt.plot(time + T, y\_t)  
 plt.title('Received Signal y(t) in Time Domain')  
 plt.xlabel('time')  
 plt.ylabel('Amplitude')  
 plt.figure(2)  
 plt.plot(freq\_axis\_y, yf\_abs\_sorted)  
 plt.title('Received Signal y(t) in Frequency Domain')  
 plt.xlabel('frequency')  
 plt.ylabel('Magnitude')  
 plt.xlim(-100, 100)  
  
plt.pause(0.5)  
plt.show()

* The channel bandwidth (B) is changed after every second.
* The audio file needs to be read and transmitted in intervals of one second.