COMMUNICATION SYSTEMS LAB 3 DATE-7/9/2021

NAME – JAGRIT LODHA

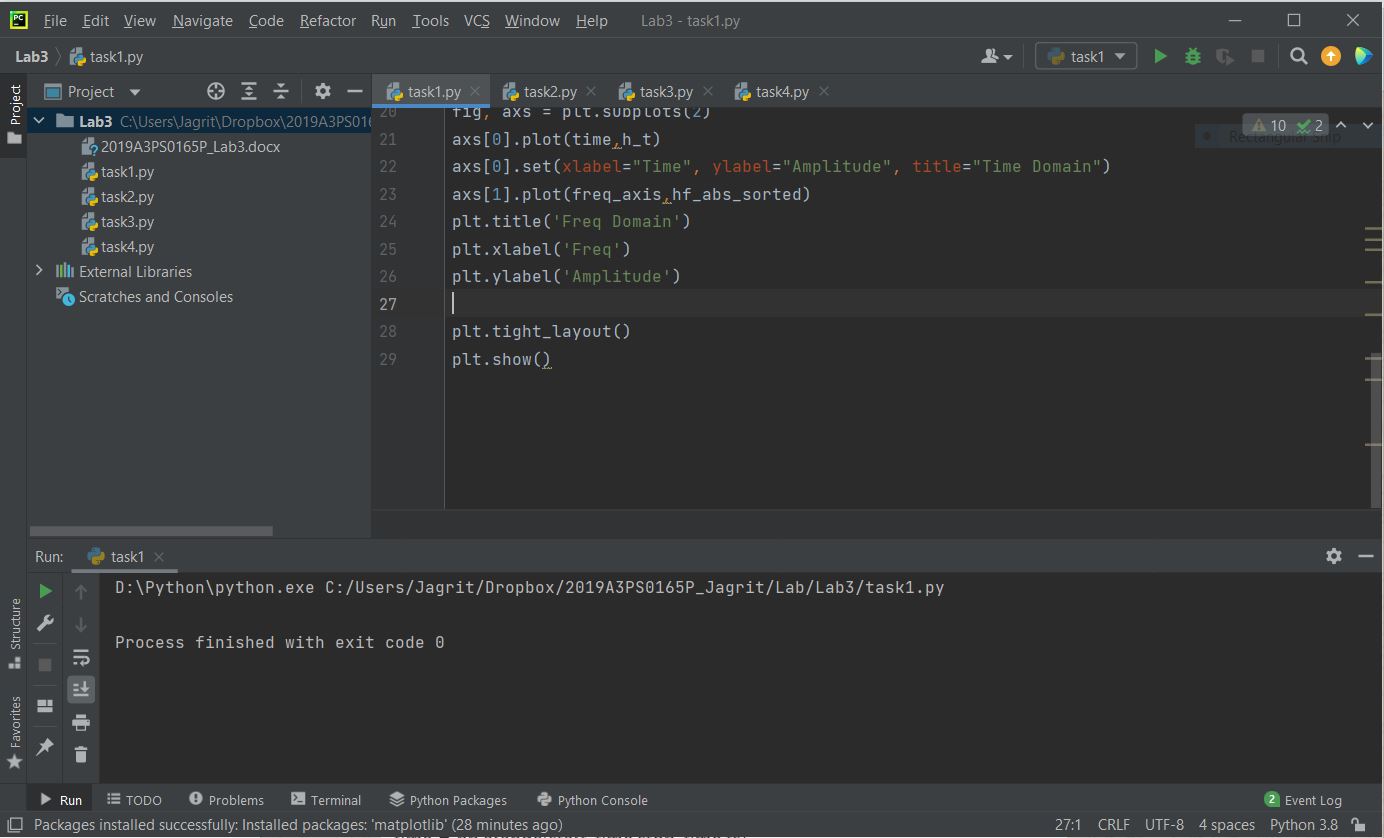
ID – 2019A3PS0165P

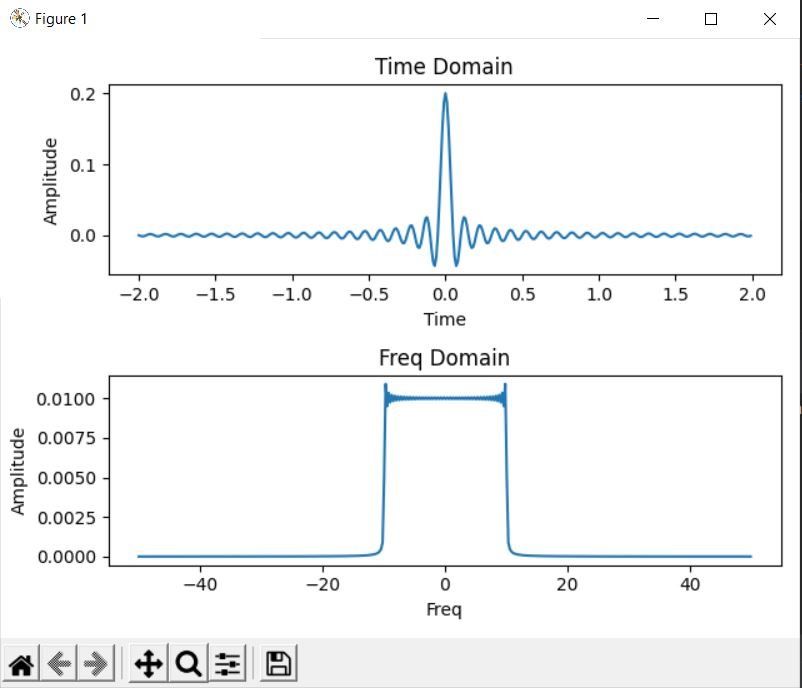
SECTION – P4

PYTHON

TASK 1 –

import numpy as np  
import matplotlib.pyplot as plt  
from numpy.fft import fft  
start\_time = -2  
stop\_time = 2  
fm = 10 # Maximum frequency component in Hertz for the given rectangle wave  
fs = 10\*fm  
ts = 1/fs  
time = np.arange(start\_time,stop\_time,ts)  
B = 0.01\*(2\*fm) ### Magnitude required = B/(2\*fm) in case of sinc function  
h\_t = B\*np.sinc(2\*fm\*time) ### np.sin(2\*fm\*np.pi\*time) for sinewave  
hf = fft(h\_t)/fs  
N = len(hf)  
freq\_axis = np.linspace(-fs/2, fs/2, N) ### to sample frequency axis as well  
  
hf\_abs = abs(hf)  
hf\_abs\_sorted = np.fft.fftshift(hf\_abs) ##### for increasing freq samples  
  
  
fig, axs = plt.subplots(2)  
axs[0].plot(time,h\_t)  
axs[0].set(xlabel="Time", ylabel="Amplitude", title="Time Domain")  
axs[1].plot(freq\_axis,hf\_abs\_sorted)  
plt.title('Freq Domain')  
plt.xlabel('Freq')  
plt.ylabel('Amplitude')  
  
plt.tight\_layout()  
plt.show()

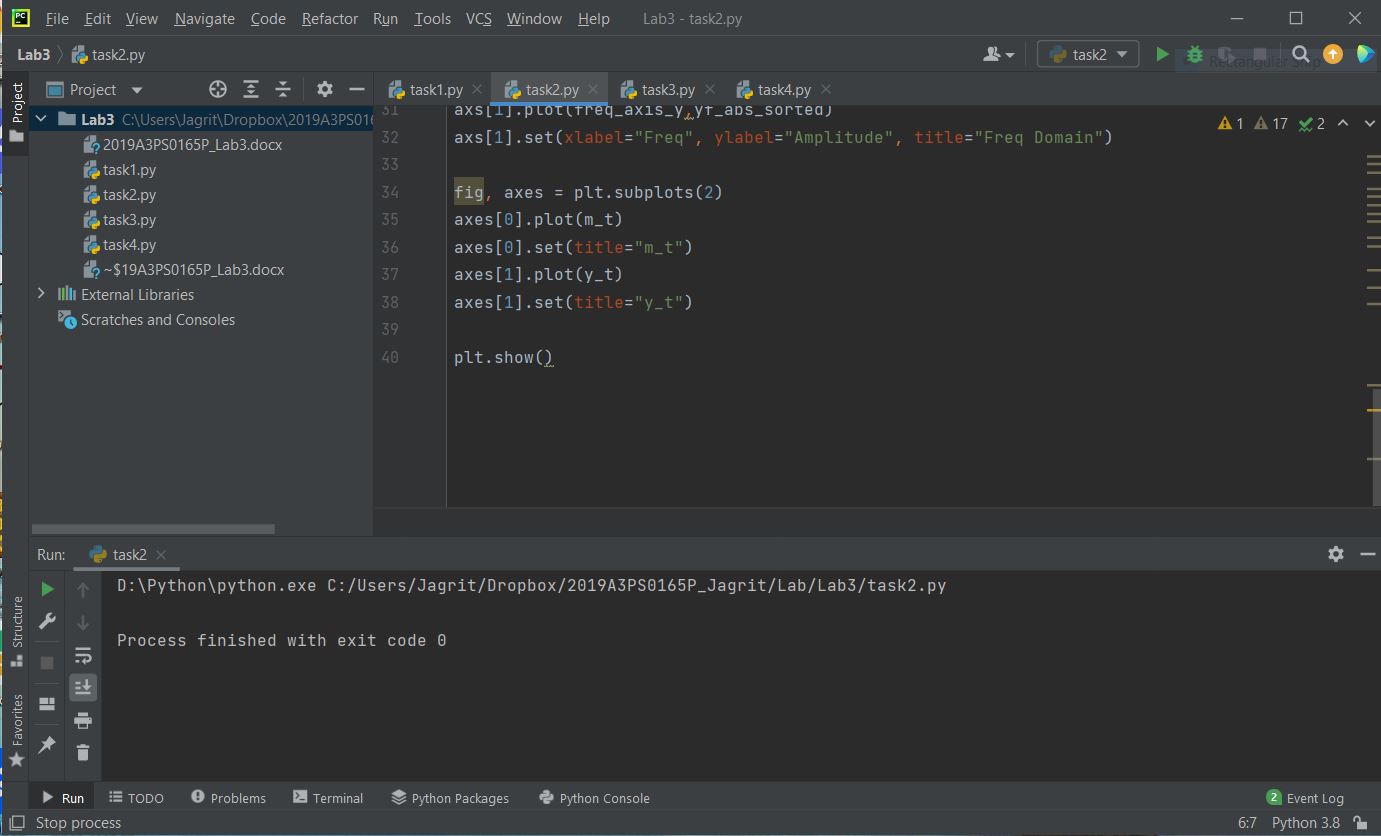




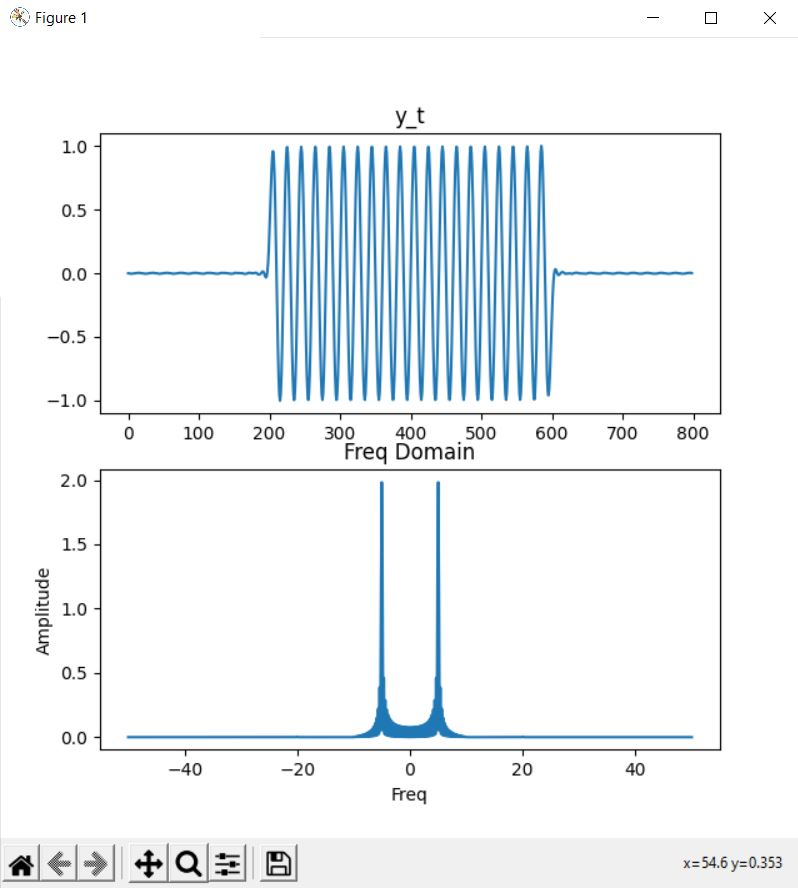
So by using the appropriate value for maximum value (0.2) and frequency (20) of sinc wave, we were able to generate the required frequency domain spectrum.

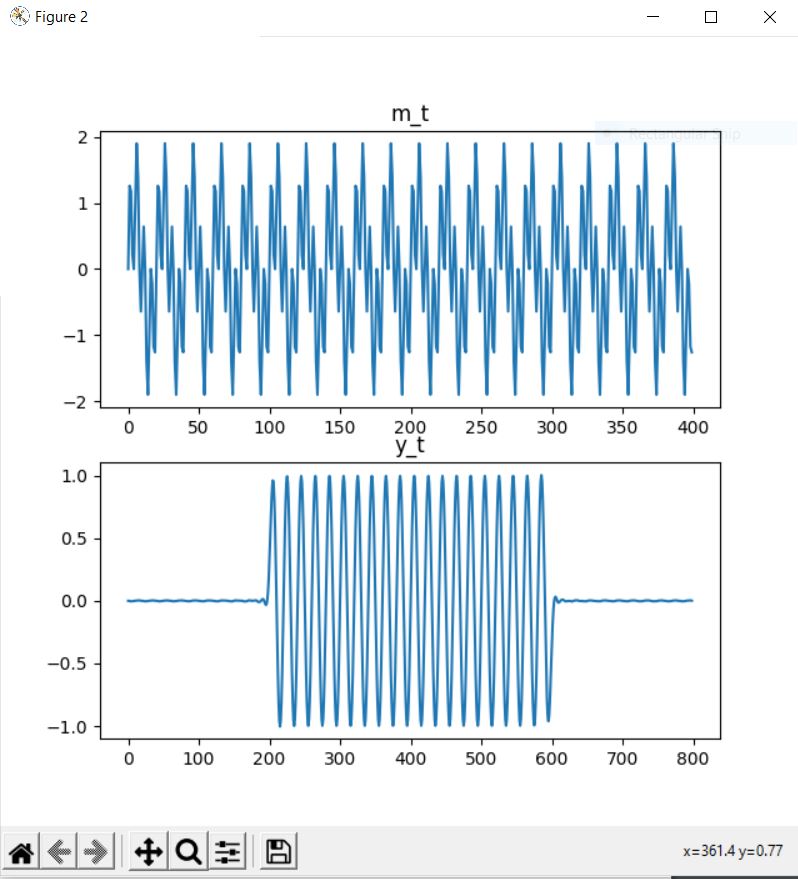
TASK 2 –

import numpy as np  
import matplotlib.pyplot as plt  
from numpy.fft import fft  
start\_time=-2  
stop\_time=2  
fm = 10 # Maximum frequency component in Hertz for the given spectrum  
fs=10\*fm  
ts=1/fs  
time = np.arange(start\_time,stop\_time,ts)  
B = 0.01\*(2\*fm) ### Magnitude required = B/(2\*fm) in case of sinc function  
h\_t = B\*np.sinc(2\*fm\*time) ### np.sin(2\*fm\*np.pi\*time) for sinewave  
hf = fft(h\_t)/fs  
N=len(hf)  
freq\_axis = np.linspace(-fs/2, fs/2, N) ###extremely important to sample freq  
  
hf\_abs = abs(hf)  
hf\_abs\_sorted = np.fft.fftshift(hf\_abs) ##### for increasing freq samples  
  
### From here we start the convolution part  
m\_t = np.sin(10\*np.pi\*time) + np.sin(40\*np.pi\*time)  
y\_t = np.convolve(m\_t,h\_t)  
y\_f = fft(y\_t)/fs  
Ny = len(y\_f)  
freq\_axis\_y = np.linspace(-fs/2, fs/2, Ny)  
yf\_abs = abs(y\_f)  
yf\_abs\_sorted = np.fft.fftshift(yf\_abs)  
  
fig, axs = plt.subplots(2)  
axs[0].plot(y\_t)  
axs[0].set(title="y\_t")  
axs[1].plot(freq\_axis\_y,yf\_abs\_sorted)  
axs[1].set(xlabel="Freq", ylabel="Amplitude", title="Freq Domain")  
  
fig, axes = plt.subplots(2)  
axes[0].plot(m\_t)  
axes[0].set(title="m\_t")  
axes[1].plot(y\_t)  
axes[1].set(title="y\_t")  
  
plt.show()



These graphs are for the case when **fm=10** (the original channel)



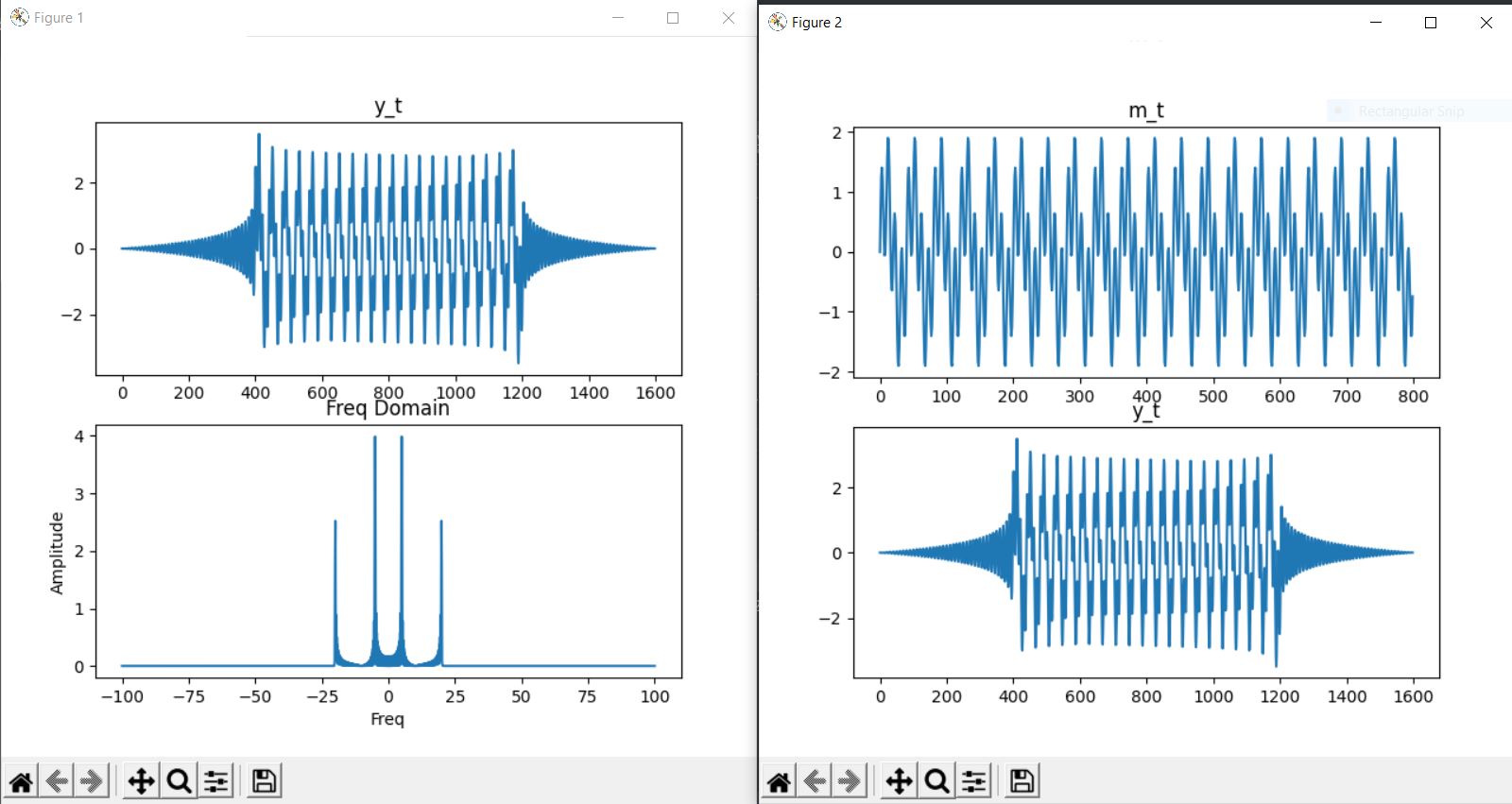


The two frequency peaks are at w=-5 and 5.

The corresponding y\_t for given m\_t are also shown.

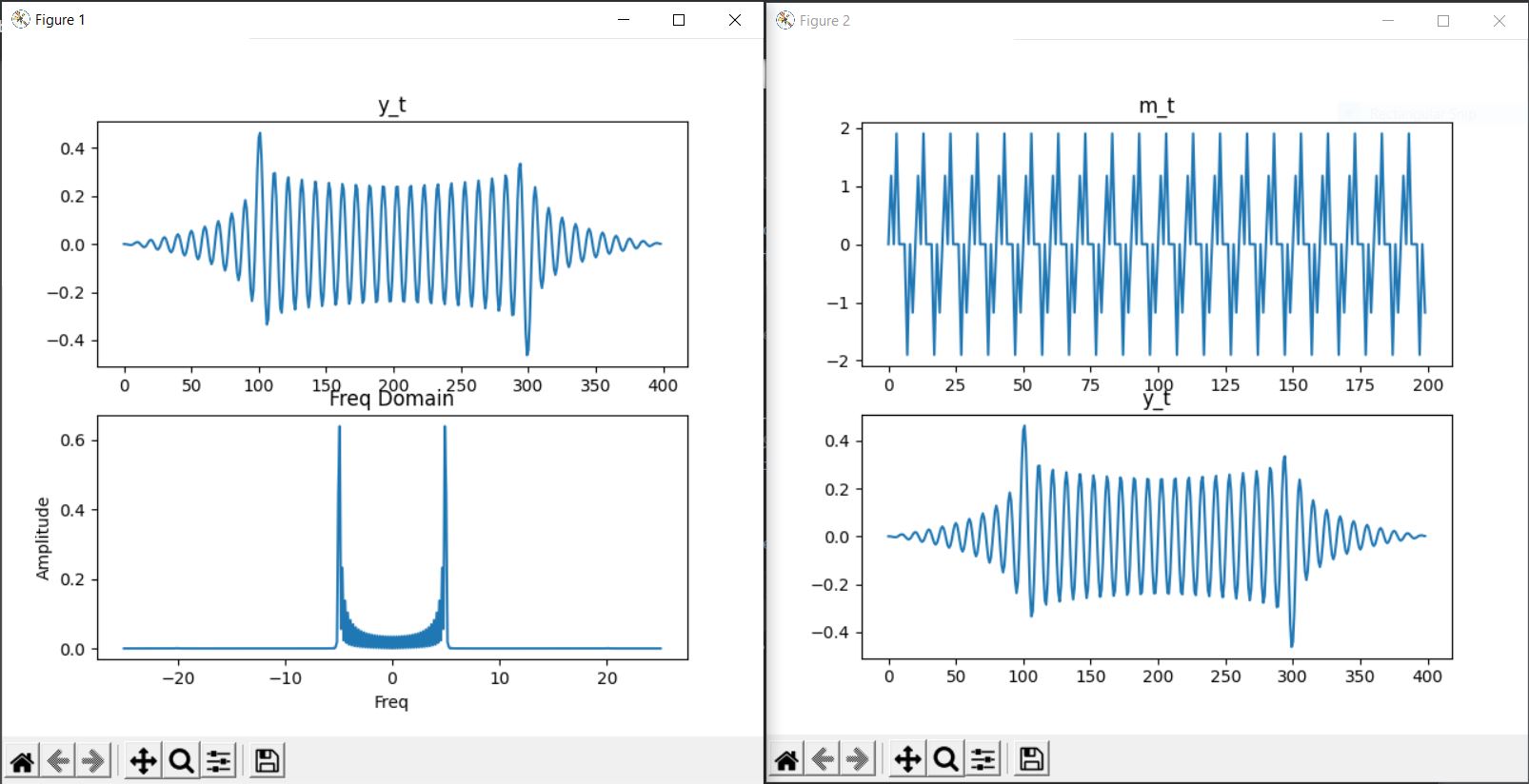
Now, to show the effect of increasing the channel bandwidth, i.e., changing fm –

We take **fm=20** and the results obtained are shown below:



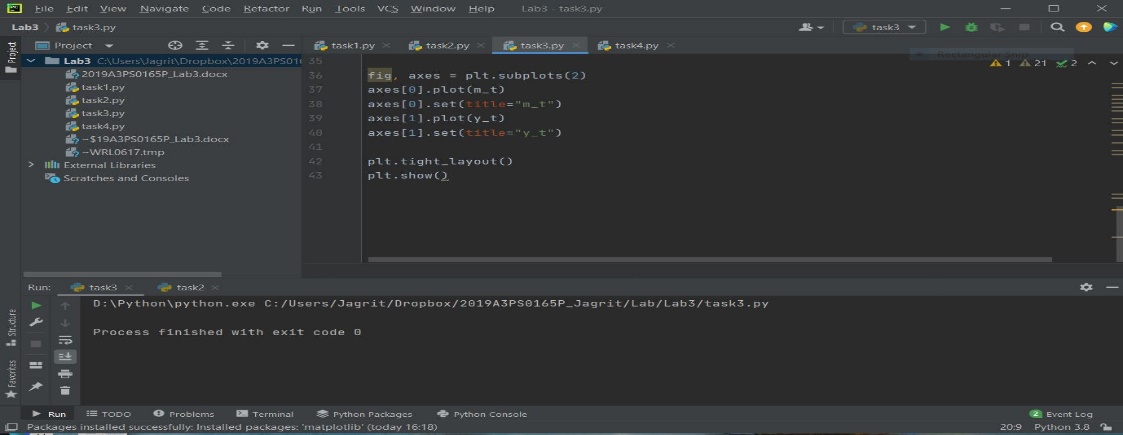
So we can see on increasing the channel bandwidth, more frequency peaks are observed along with a more complex y\_t.

Similarly, for **fm=5**:

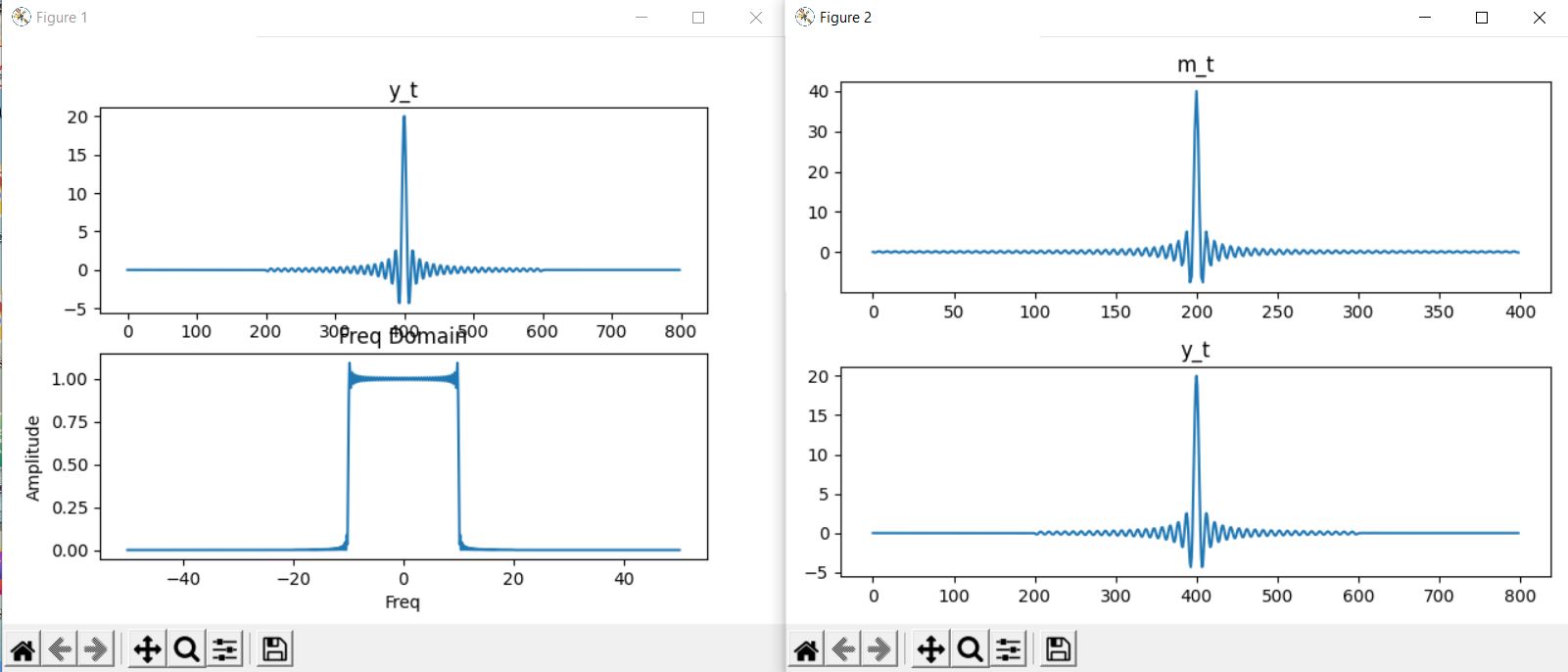


TASK 3 –

import numpy as np  
import matplotlib.pyplot as plt  
from numpy.fft import fft  
start\_time=-2  
stop\_time=2  
fm = 10 # Maximum frequency component in Hertz for the given spectrum  
fs=10\*fm  
ts=1/fs  
time = np.arange(start\_time,stop\_time,ts)  
B = 0.01\*(2\*fm) ### Magnitude required = B/(2\*fm) in case of sinc function  
h\_t = B\*np.sinc(2\*fm\*time) ### np.sin(2\*fm\*np.pi\*time) for sinewave  
hf = fft(h\_t)/fs  
N=len(hf)  
freq\_axis = np.linspace(-fs/2, fs/2, N) ###extremely important to sample freq  
  
hf\_abs= abs(hf)  
hf\_abs\_sorted=np.fft.fftshift(hf\_abs) ##### for increasing freq samples  
  
### From here we start the convolution part  
fms = 10  
m\_t = 2\*fms\*np.sinc(2\*fms\*time) ### This fms is different than that of the channel bandwidth fm  
y\_t = np.convolve(m\_t,h\_t)  
y\_f = fft(y\_t)/fs  
Ny = len(y\_f)  
freq\_axis\_y = np.linspace(-fs/2, fs/2, Ny)  
yf\_abs = abs(y\_f)  
yf\_abs\_sorted = np.fft.fftshift(yf\_abs)  
  
fig, axs = plt.subplots(2)  
axs[0].plot(y\_t)  
axs[0].set(title="y\_t")  
axs[1].plot(freq\_axis\_y,yf\_abs\_sorted)  
#axs[1].plot(freq\_axis,hf\_abs\_sorted)  
axs[1].set(xlabel="Freq", ylabel="Amplitude", title="Freq Domain")  
  
fig, axes = plt.subplots(2)  
axes[0].plot(m\_t)  
axes[0].set(title="m\_t")  
axes[1].plot(y\_t)  
axes[1].set(title="y\_t")  
  
plt.tight\_layout()  
plt.show()

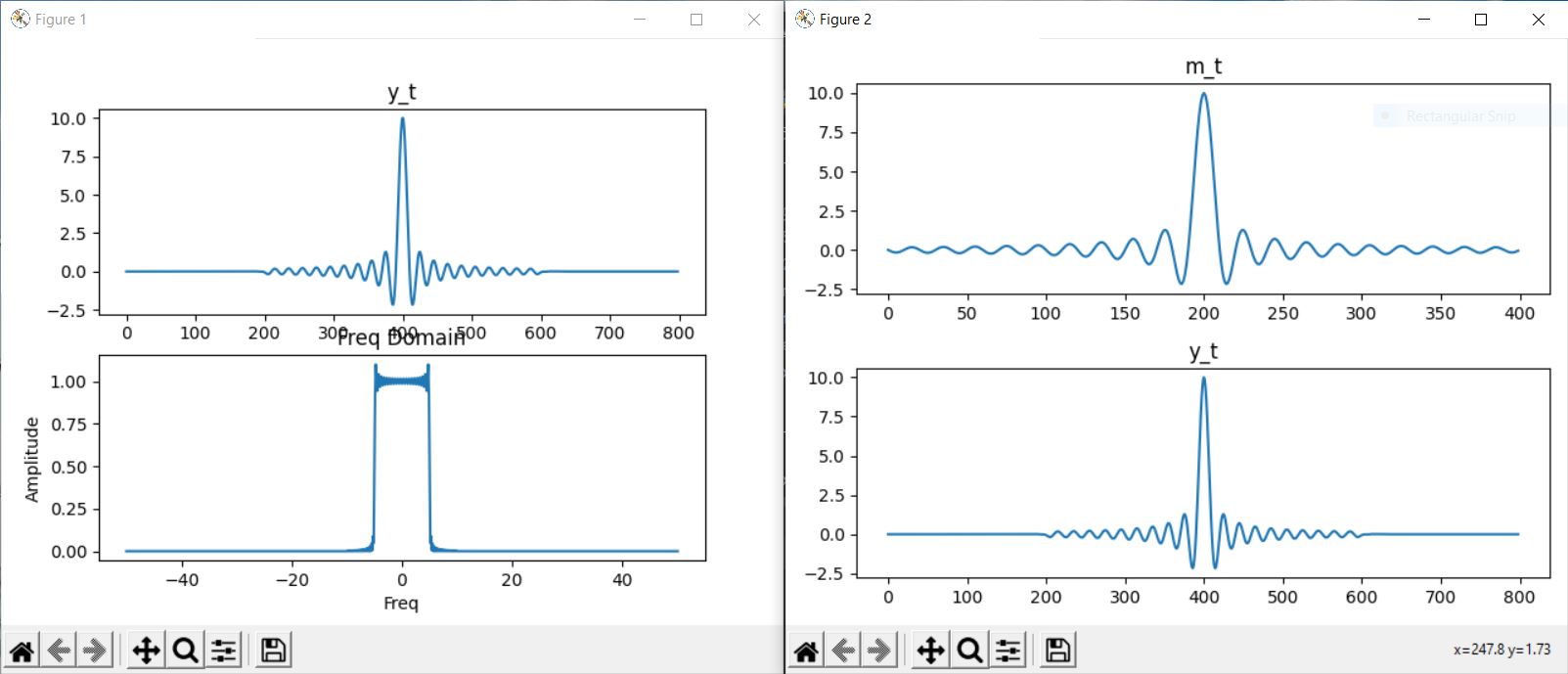


Initially, taken for **fm of signal as 20** –



The peak in frequency domain of received signal y\_t is at -10 and 10.

Then changing the bandwidth of the signal, we take **fm of signal as 5** –

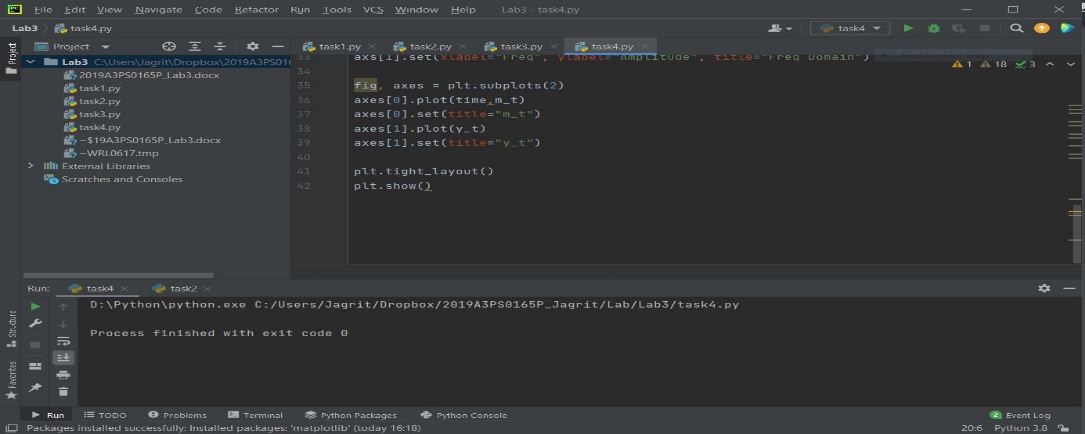


In this case, the frequency domain peaks of y\_t are at -5 and 5.

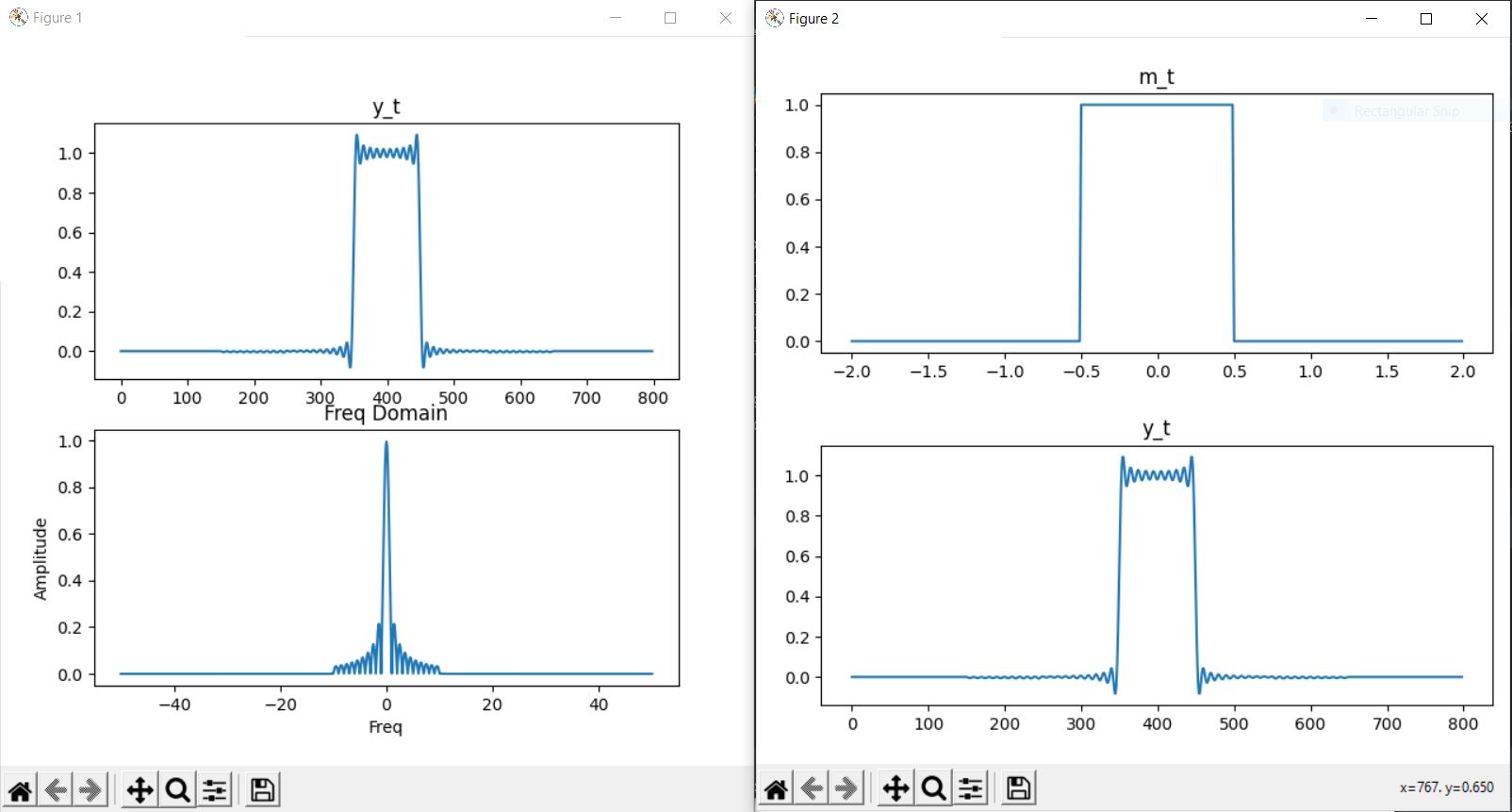
So we can easily see that by changing the bandwidth of our input sinc wave, we can alter the bandwidth of our received signal even for the same channel bandwidth.

TASK 4 –

import numpy as np  
import matplotlib.pyplot as plt  
from numpy.fft import fft  
start\_time = -2  
stop\_time = 2  
fm = 10 # Maximum frequency component in Hertz for the given channel spectrum  
fs = 10\*fm  
ts = 1/fs  
time = np.arange(start\_time,stop\_time,ts)  
B = 0.01\*(2\*fm) ### Magnitude required = B/(2\*fm) in case of sinc function  
h\_t = B\*np.sinc(2\*fm\*time) ### np.sin(2\*fm\*np.pi\*time) for sinewave  
hf = fft(h\_t)/fs  
N=len(hf)  
freq\_axis = np.linspace(-fs/2, fs/2, N) ###extremely important to sample freq  
  
hf\_abs= abs(hf)  
hf\_abs\_sorted=np.fft.fftshift(hf\_abs) ##### for increasing freq samples  
  
### From here we start the convolution part  
T = 1  
m\_t = [1 if abs(i)<=T/2 else 0 for i in time] ### Too gennerate the rectangular pulse  
y\_t = np.convolve(m\_t,h\_t)  
y\_f = fft(y\_t)/fs  
Ny = len(y\_f)  
freq\_axis\_y = np.linspace(-fs/2, fs/2, Ny)  
yf\_abs = abs(y\_f)  
yf\_abs\_sorted = np.fft.fftshift(yf\_abs)  
  
fig, axs = plt.subplots(2)  
axs[0].plot(y\_t)  
axs[0].set(title="y\_t")  
axs[1].plot(freq\_axis\_y,yf\_abs\_sorted)  
axs[1].set(xlabel="Freq", ylabel="Amplitude", title="Freq Domain")  
  
fig, axes = plt.subplots(2)  
axes[0].plot(time,m\_t)  
axes[0].set(title="m\_t")  
axes[1].plot(y\_t)  
axes[1].set(title="y\_t")  
  
plt.tight\_layout()  
plt.show()

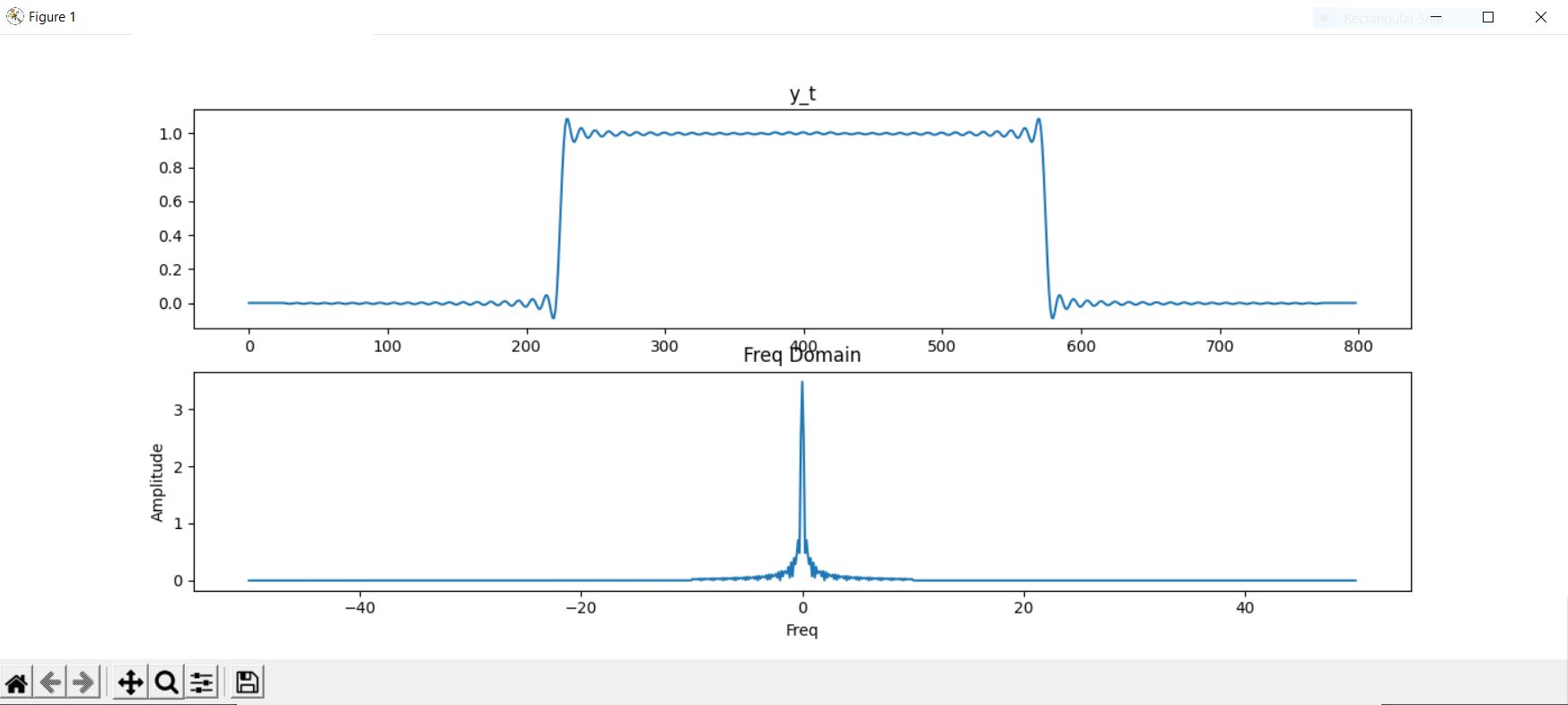


Initally, taking signal duration **(T=1)**

****

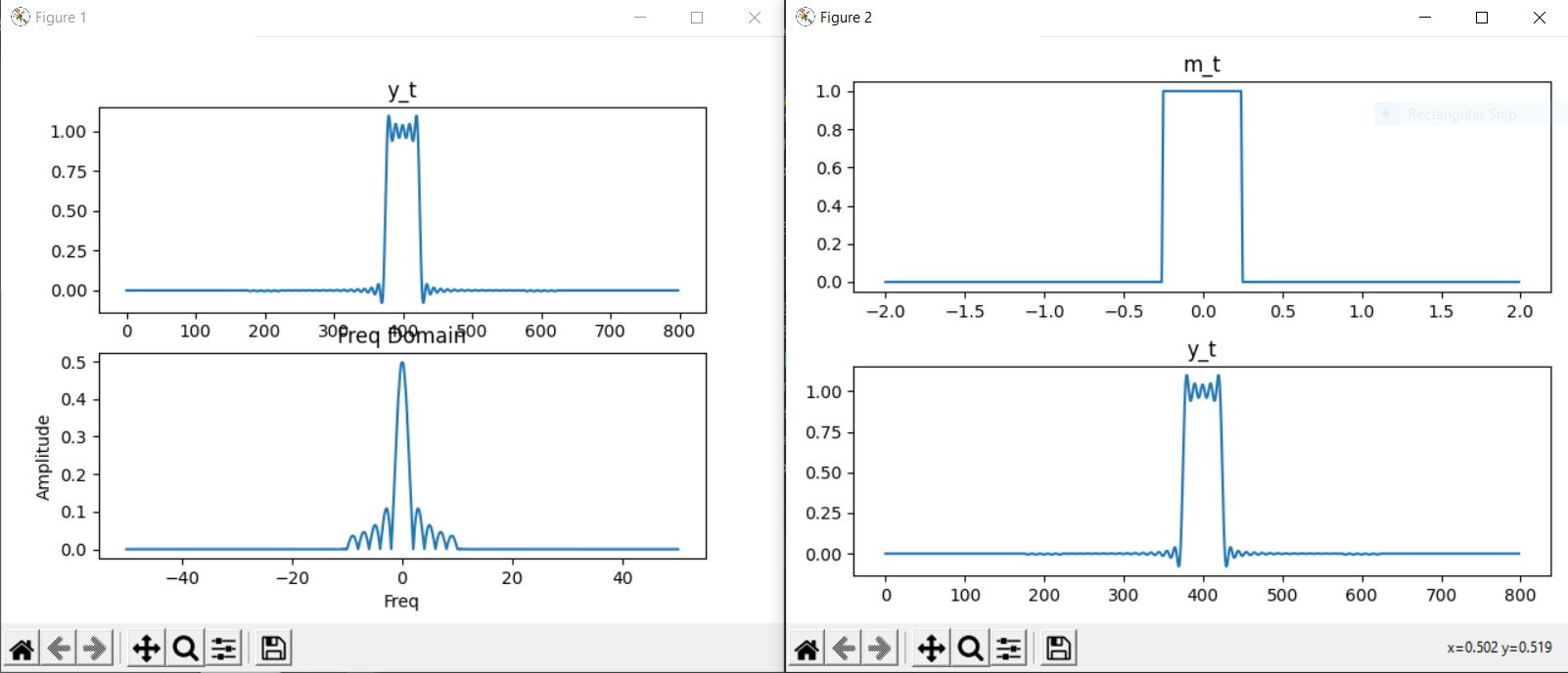
The frequency domain of y\_t goes up to 10 only (the channel bandwidth)

Now, taking **T=3.5 –**

****

Here we can see that the frequency response still goes till 10 only, but it is much more compressed.

Finally, we take the case for **T=0.5** –



Now it can easily be observed that the frequency domain is much more spread out (while still being within the same channel bandwidth of -10 to 10).