COMSYS LAB 3

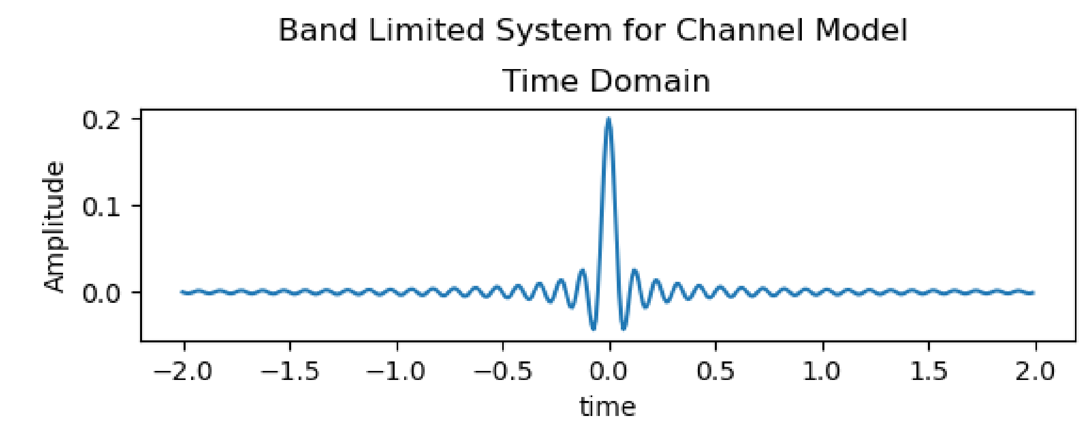
Name: Sidharth S. Nair | ID: 2019A3PS0178P

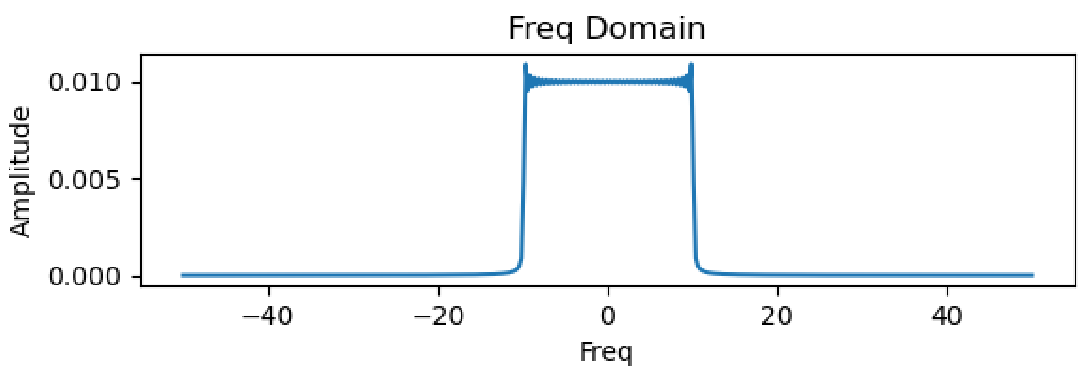
Task-1 : band limited system

code

```
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)
h_t = 2*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=np.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_title('Time Domain')
axs[0].set_xlabel('time')
axs[0].set_ylabel('Amplitude')
axs[1].plot(freq_axis,hf_abs_sorted)
axs[1].set_title('Freq Domain')
axs[1].set_xlabel('Freq')
axs[1].set_ylabel('Amplitude')
plt.show()
```

output





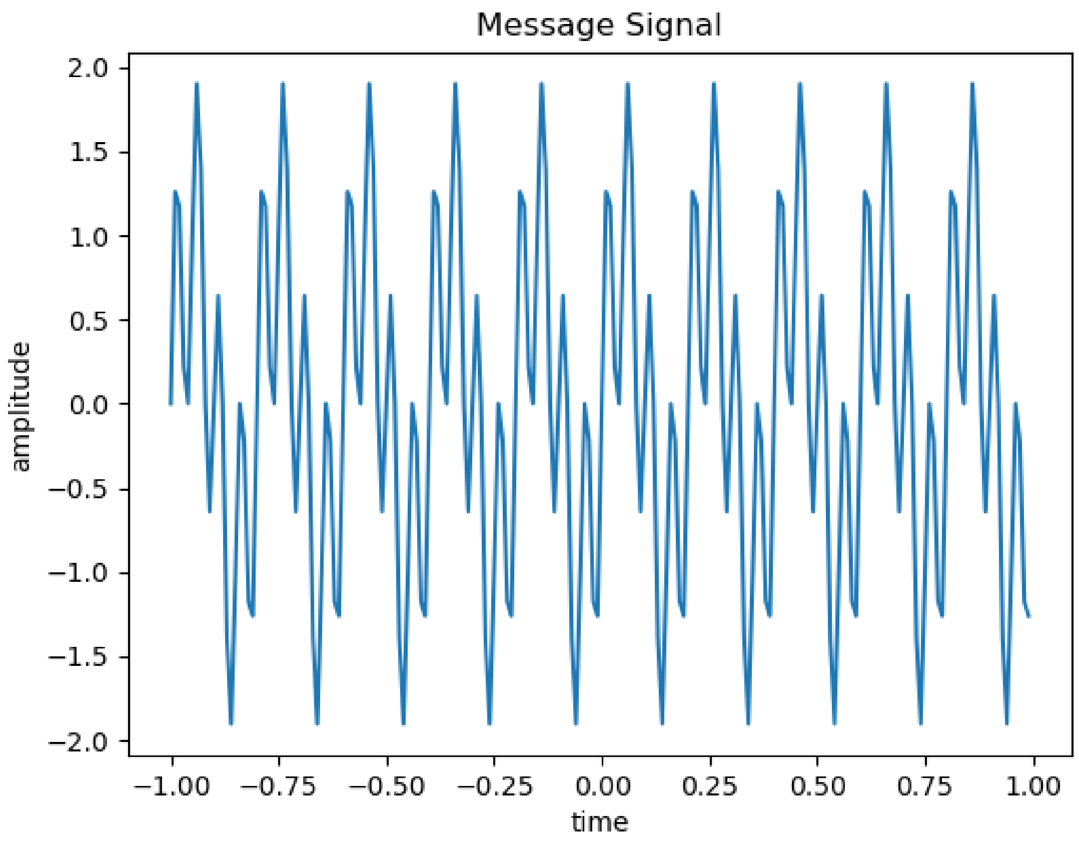
Question-2: Sine wave pulse

code

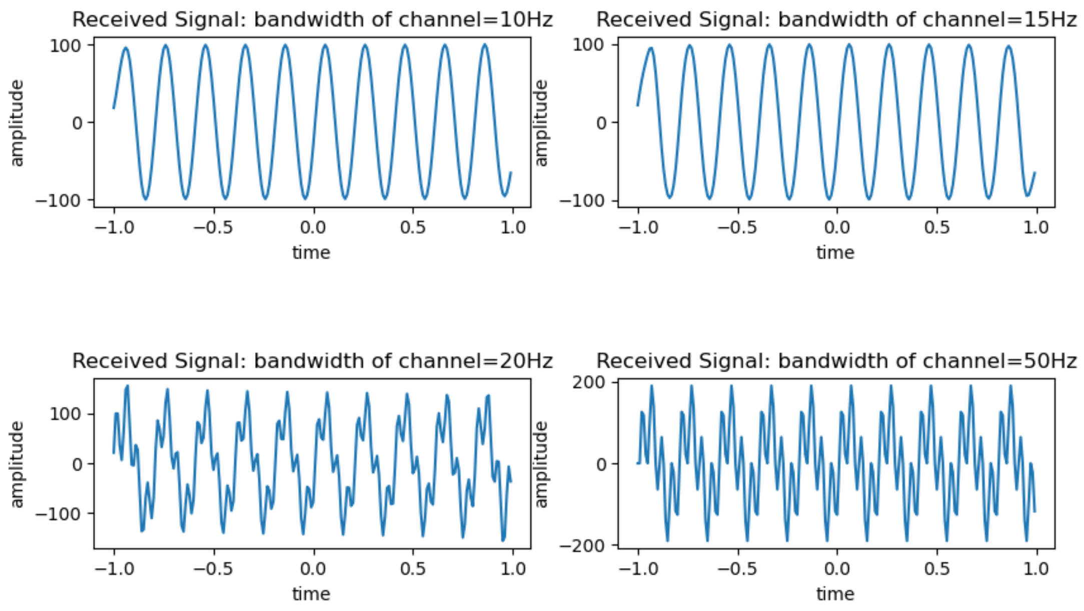
```
import numpy as np
import matplotlib.pyplot as plt
from  numpy.fft import fft
def getfft(bandwidth=10,start=-2,stop=2,ts=(1/100)):

    t=np.arange(start,stop,ts)
    channel=2*bandwidth*np.sinc(2*bandwidth*t)
    rec_signal=np.convolve(channel,m,mode='same')
    hf = fft(rec_signal)/fs
    N=len(hf)
    freq_axis = np.linspace(-fs/2, fs/2, N)  ###extremely important to sample freq
    hf_abs=np.abs(hf)
    hf_abs_sorted=np.fft.fftshift(hf_abs)
    return freq_axis,rec_signal,hf_abs_sorted
f1=5
f2=20
fs=100
ts=1/fs
start=-1
stop=1
```

```
t=np.arange(start,stop,ts)
m=np.sin(2*np.pi*f1*t)+np.sin(2*np.pi*f2*t)
bw=[10,15,20,50]
freq_axis=[[] for x in bw]
hf_abs=[[] for x in bw]
received=[[] for x in bw]
y_t=[[] for x in bw]
for i,j in enumerate(bw):
    freq_axis[i],y_t[i],hf_abs[i]=getfft(bandwidth=j,start=start,stop=stop,ts=ts)

plt.plot(t,m)
plt.xlabel('time')
plt.ylabel('amplitude')
plt.title('Message Signal')
N=2*(stop-start)*fs
fig,axs=plt.subplots(2,2)
fig2,ax=plt.subplots(2,2)
t2=np.linspace(2*start,2*stop,N)
for i in range(len(bw)):
    plt.subplots_adjust(hspace = 1)
    axs[i//2,i%2].plot(freq_axis[i],hf_abs[i])
    axs[i//2,i%2].set_xlabel('freq')
    axs[i//2,i%2].set_ylabel('amplitude')
    axs[i//2,i%2].set_title(f'bandwidth of channel={bw[i]}Hz')
    ax[i//2,i%2].plot(t,y_t[i])
    ax[i//2,i%2].set_xlabel('time')
    ax[i//2,i%2].set_ylabel('amplitude')
    ax[i//2,i%2].set_title(f'Received Signal: bandwidth of channel={bw[i]}Hz')
plt.show()
```

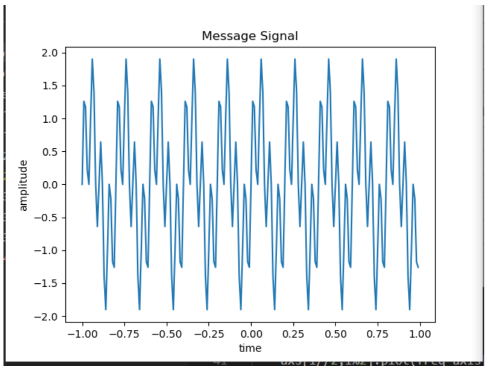
Plots

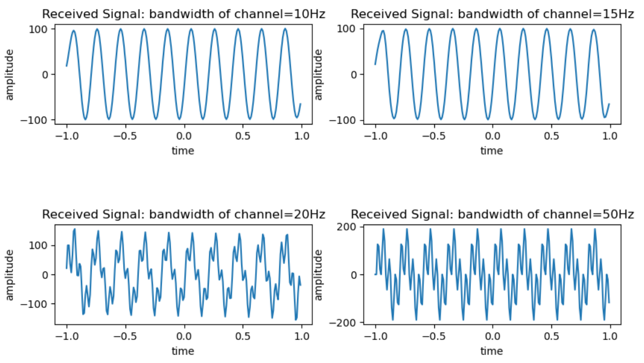


message signal



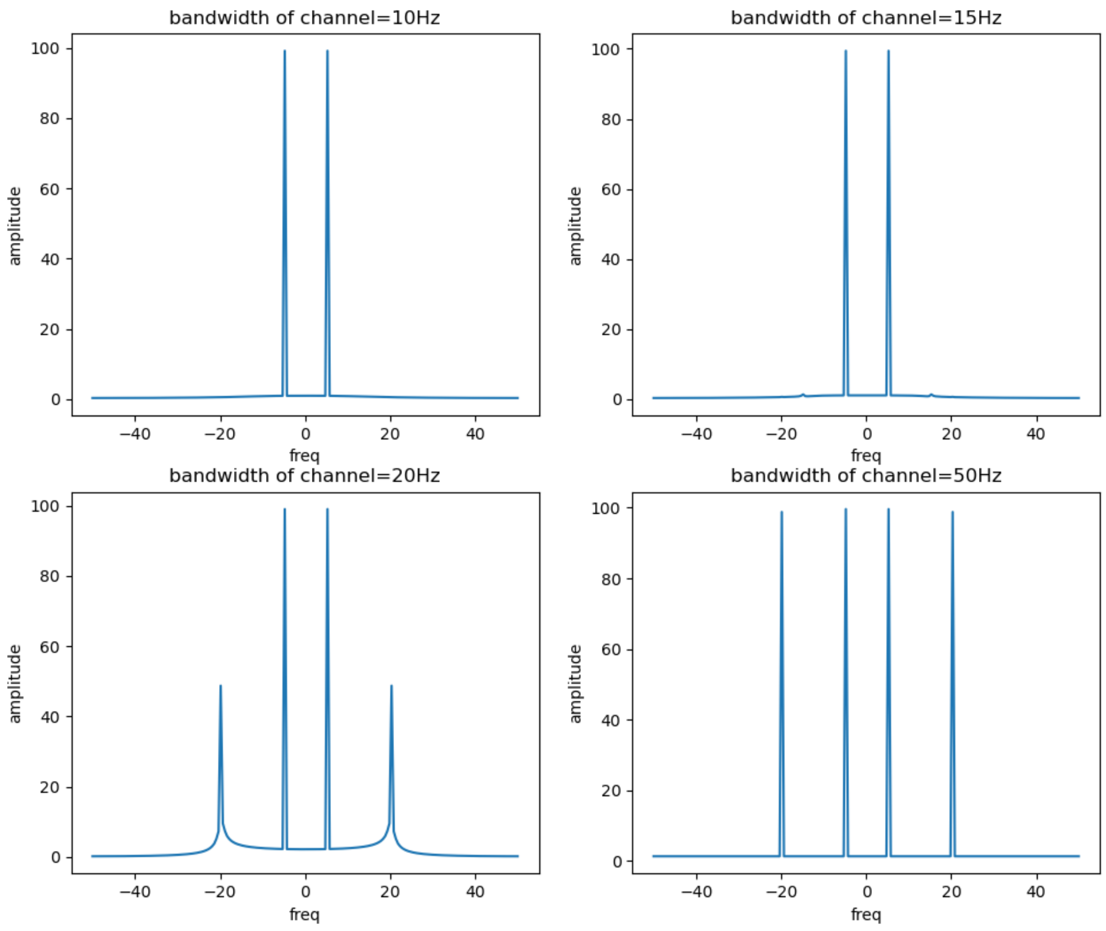
received signals plotted by bandwidth





comparison between the two

Spectrum Plot



plot of the received signals frequency spectrum at different BW

observations

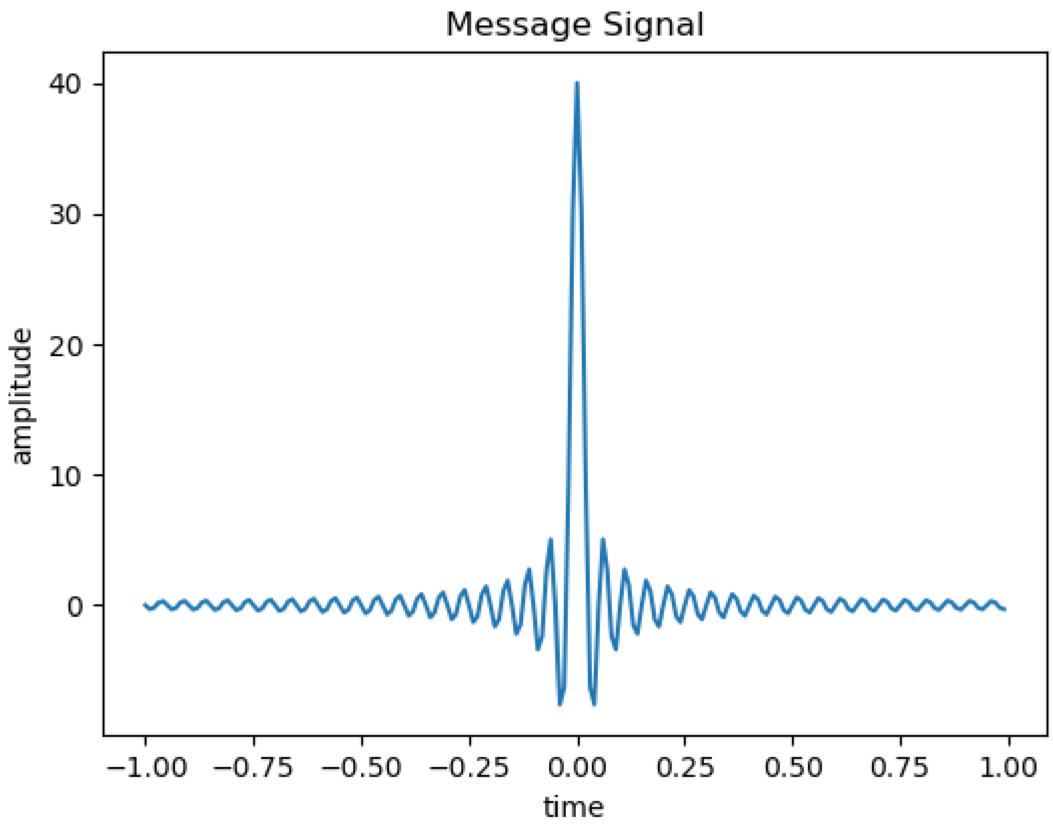
- We can see that as the bandwidth of channel increased all of the message signals frequency spectrum passed through the channel and the received signal now is actually similar to message signal

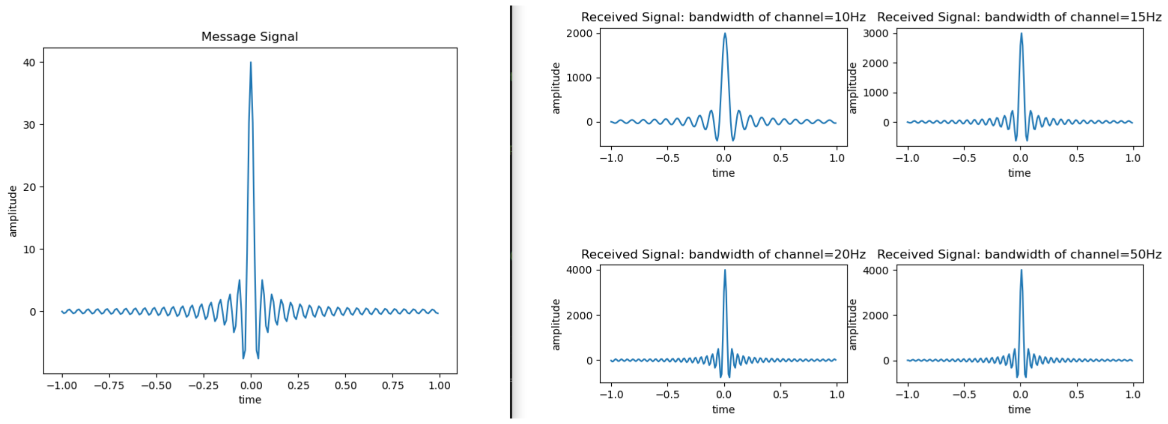
Task -3 : Sinc pulse message signal

code

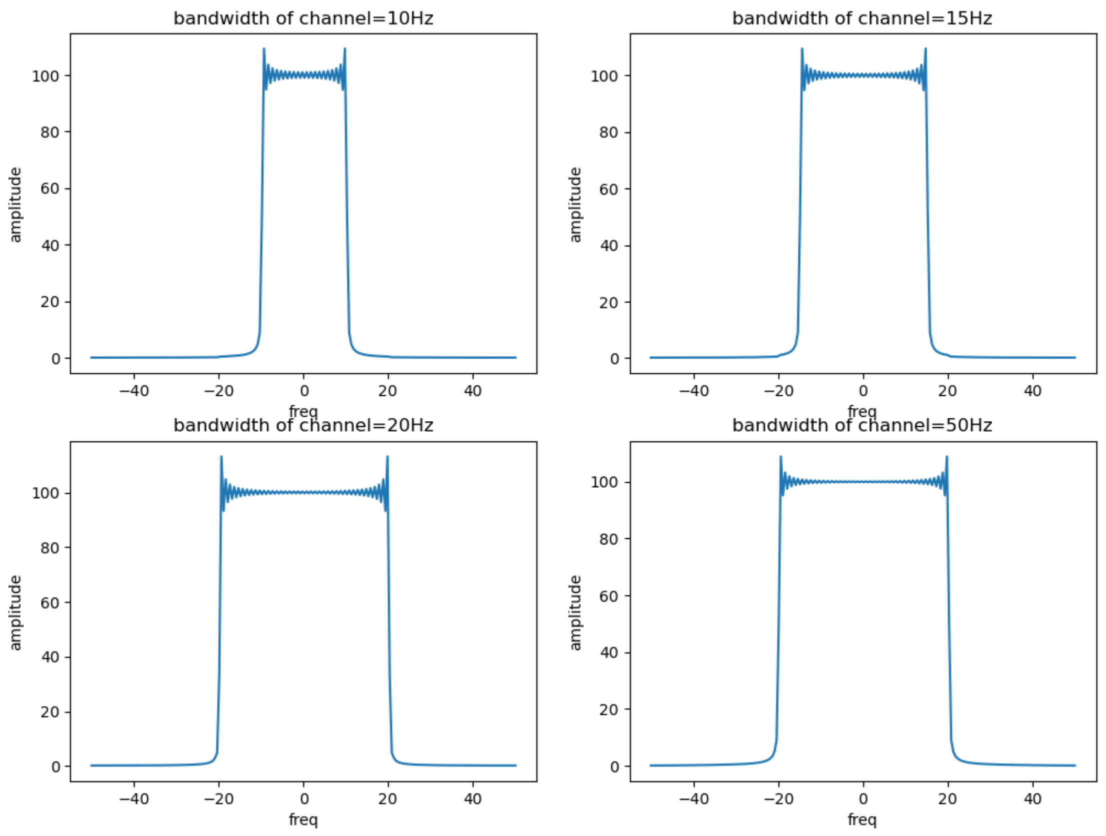
```
import numpy as np
import matplotlib.pyplot as plt
from  numpy.fft import fft
def getfft(bandwidth=10,start=-2,stop=2,ts=(1/100)):

    t=np.arange(start,stop,ts)
    channel=2*bandwidth*np.sinc(2*bandwidth*t)
    rec_signal=np.convolve(channel,m,mode='same')
    hf = fft(rec_signal)/fs
    N=len(hf)
    freq_axis = np.linspace(-fs/2, fs/2, N)  ###extremely important to sample freq
    hf_abs=np.abs(hf)
    hf_abs_sorted=np.fft.fftshift(hf_abs)
    return freq_axis,rec_signal,hf_abs_sorted
fm=20 # 20Hz is the Bandwidth chosen for the sinc pulse (in freq domain)
fs=100
ts=1/fs
start=-1
stop=1
t=np.arange(start,stop,ts)
m=2*fm*np.sinc(2*fm*t)
bw=[10,15,20,50]
freq_axis=[[] for x in bw]
hf_abs=[[] for x in bw]
received=[[] for x in bw]
y_t=[[] for x in bw]
for i,j in enumerate(bw):
    freq_axis[i],y_t[i],hf_abs[i]=getfft(bandwidth=j,start=start,stop=stop,ts=ts)

plt.plot(t,m)
plt.xlabel('time')
plt.ylabel('amplitude')
plt.title('Message Signal')
fig,axs=plt.subplots(2,2)
fig2,ax=plt.subplots(2,2)
for i in range(len(bw)):
    plt.subplots_adjust(hspace = 1)
    axs[i//2,i%2].plot(freq_axis[i],hf_abs[i])
    axs[i//2,i%2].set_xlabel('freq')
    axs[i//2,i%2].set_ylabel('amplitude')
    axs[i//2,i%2].set_title(f'bandwidth of channel={bw[i]}Hz')
    ax[i//2,i%2].plot(t,y_t[i])
    ax[i//2,i%2].set_xlabel('time')
    ax[i//2,i%2].set_ylabel('amplitude')
    ax[i//2,i%2].set_title(f'Received Signal: bandwidth of channel={bw[i]}Hz')
plt.show()
```

Outputs





Comparison with message and received signal



Spectra comparison for different bandwidths

Task 4: Rectangular Pulse Message

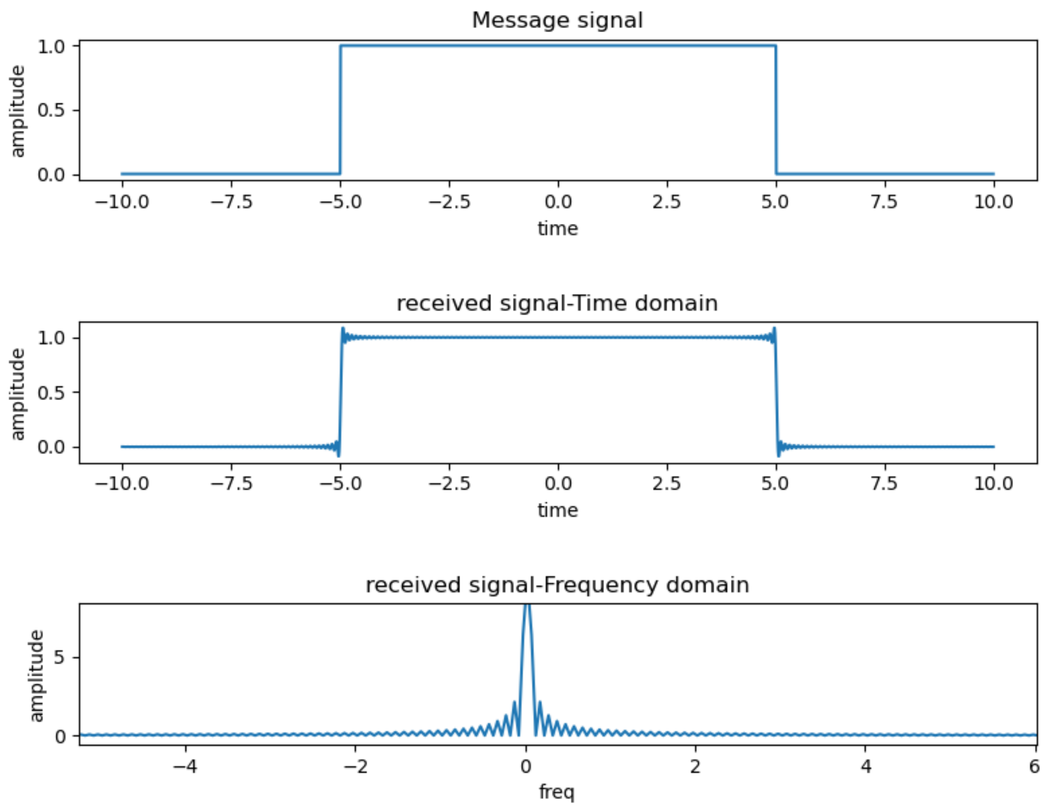
Code

```
import numpy as np
import pylab as plt
from  numpy.fft import fft
def rect(x,T):
    return np.where(np.logical_and(x<T,x>-T),1,0)

fs=200
ts=1/fs
start=-3
stop=3
t=np.arange(start,stop,ts)
duration=[0.1,1,2] # Width of the rect signal = 5 sec
bandwidth=10

#m=rect(t,duration)  # Rect signal message
def getfft(m):
    channel=2*bandwidth*np.sinc(2*bandwidth*t)
    y_t=np.convolve(channel,m,mode='same')
   # y_t=y_t/(duration*np.max(channel)) #normalising the received signal
    hf = fft(y_t)/fs
    N=len(hf)
    freq_axis = np.linspace(-fs/2,fs/2 , N)  ###extremely important to sample freq
    hf_abs=np.abs(hf)
    hf_abs_sorted=np.fft.fftshift(hf_abs)
    return freq_axis,y_t, hf_abs_sorted

hf_abs_sorted=[[] for x in duration]
freq_axis=[[] for x in duration]
y_t=[[] for x in duration]
m=[[] for x in duration]
for i,j in enumerate(duration):
    m[i]=rect(t,j)
    freq_axis[i],y_t[i],hf_abs_sorted[i]=getfft(m[i])
fig,axs=plt.subplots (3,1)
fig1,axs1=plt.subplots (3,1)
fig2,axs2=plt.subplots (3,1)
fig.subplots_adjust(hspace = 1)
fig1.subplots_adjust(hspace = 1)
fig2.subplots_adjust(hspace = 1)
axs[0].plot(t,m[0])
axs[0].set_title('Message signal')
axs[0].set_xlabel('time')
axs[0].set_ylabel('amplitude')

axs[1].plot(t,y_t[0])
axs[1].set_title('received signal-Time domain')
axs[1].set_xlabel('time')
axs[1].set_ylabel('amplitude')
axs[2].plot(freq_axis[0],hf_abs_sorted[0])
axs[2].set_title('received signal-Frequency domain')
axs[2].set_xlabel('freq')
axs[2].set_ylabel('amplitude')
fig.suptitle('Duration of message signal=0.1s')
axs1[0].plot(t,m[1])
axs1[0].set_title('Message signal')
axs1[0].set_xlabel('time')
axs1[0].set_ylabel('amplitude')

axs1[1].plot(t,y_t[1])
axs1[1].set_title('received signal-Time domain')
axs1[1].set_xlabel('time')
axs1[1].set_ylabel('amplitude')
axs1[2].plot(freq_axis[1],hf_abs_sorted[1])
axs1[2].set_title('received signal-Frequency domain')
axs1[2].set_xlabel('freq')
axs1[2].set_ylabel('amplitude')
fig1.suptitle('Duration of message signal=1s')
axs2[0].plot(t,m[2])
axs2[0].set_title('Message signal')
axs2[0].set_xlabel('time')
axs2[0].set_ylabel('amplitude')

axs2[1].plot(t,y_t[2])
axs2[1].set_title('received signal-Time domain')
axs2[1].set_xlabel('time')
axs2[1].set_ylabel('amplitude')
axs2[2].plot(freq_axis[2],hf_abs_sorted[2])
axs2[2].set_title('received signal-Frequency domain')
```

```
axs2[2].set_xlabel('freq')
axs2[2].set_ylabel('amplitude')
fig2.suptitle('Duration of message signal=2s')
plt.show()
```

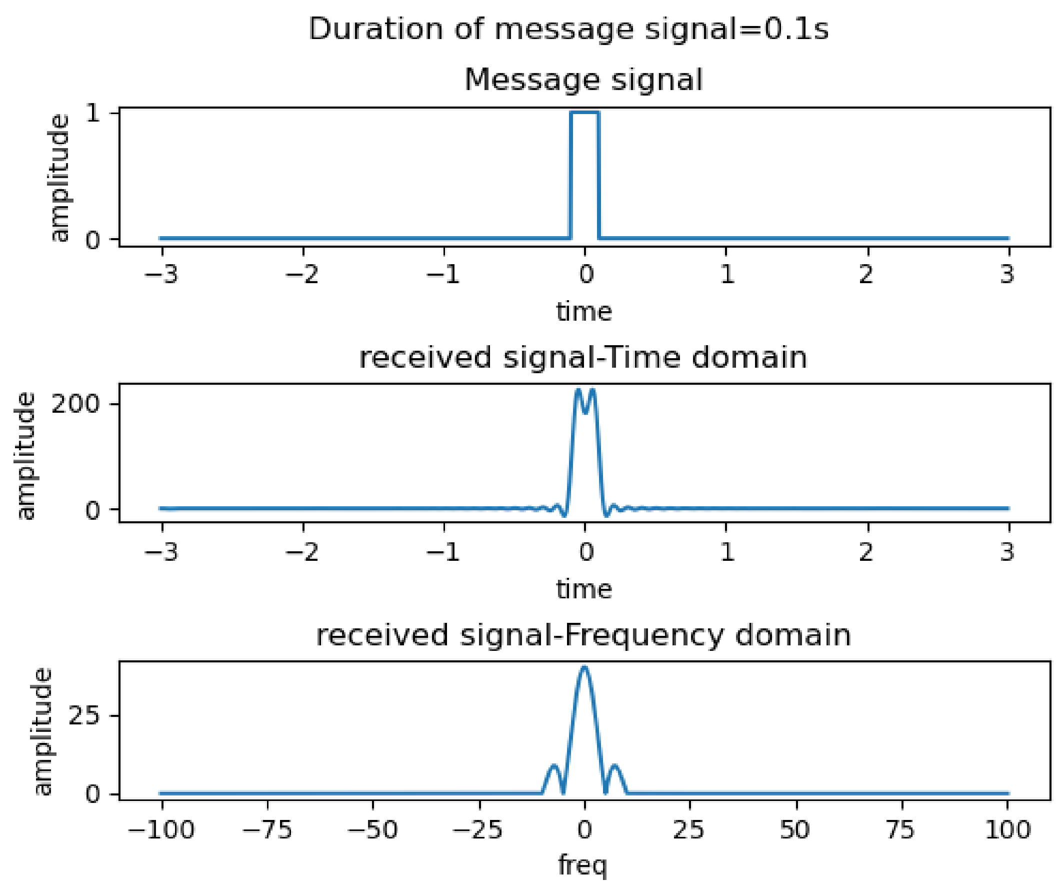
Outputs

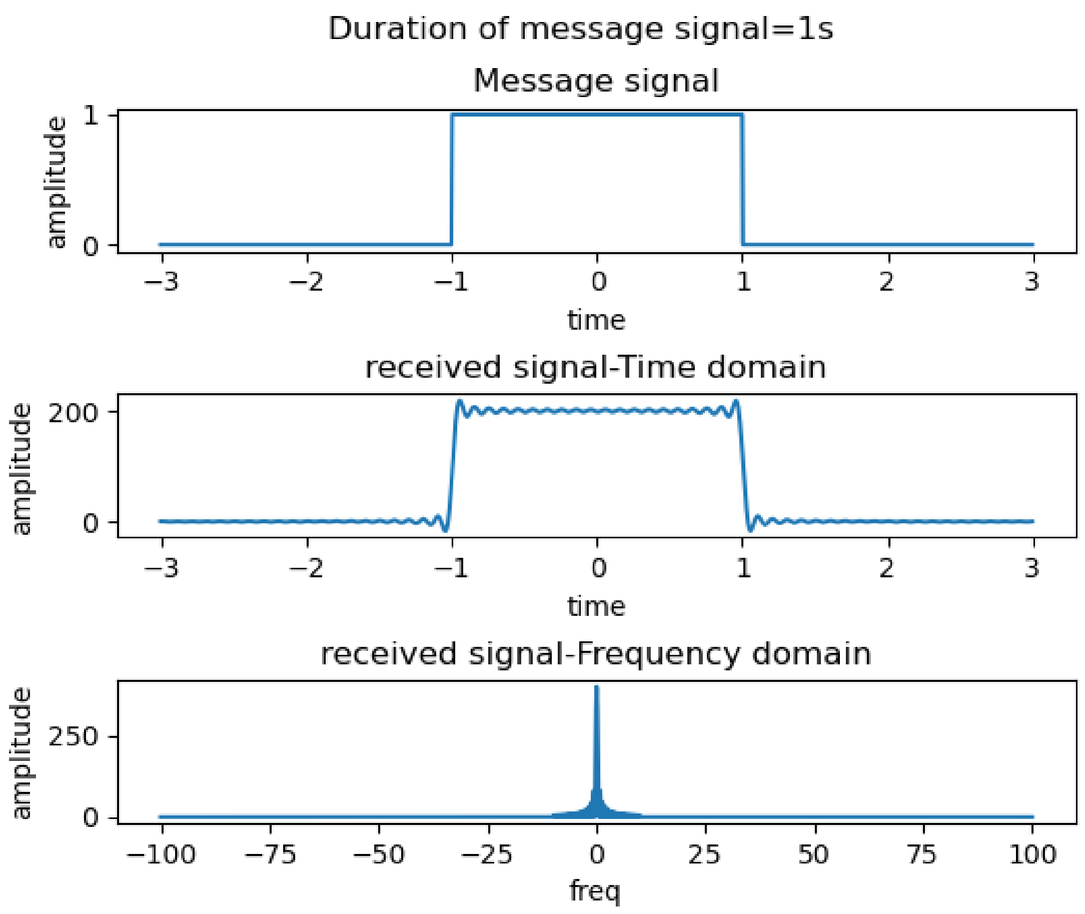
For sample output, message signal $Rect(t/\tau)$ where $\tau = 10s$ was used. Bandwidth of channel=10Hz. Received signal was normalized by dividing with the max value in `channel` variable and again dividing by duration (τ)

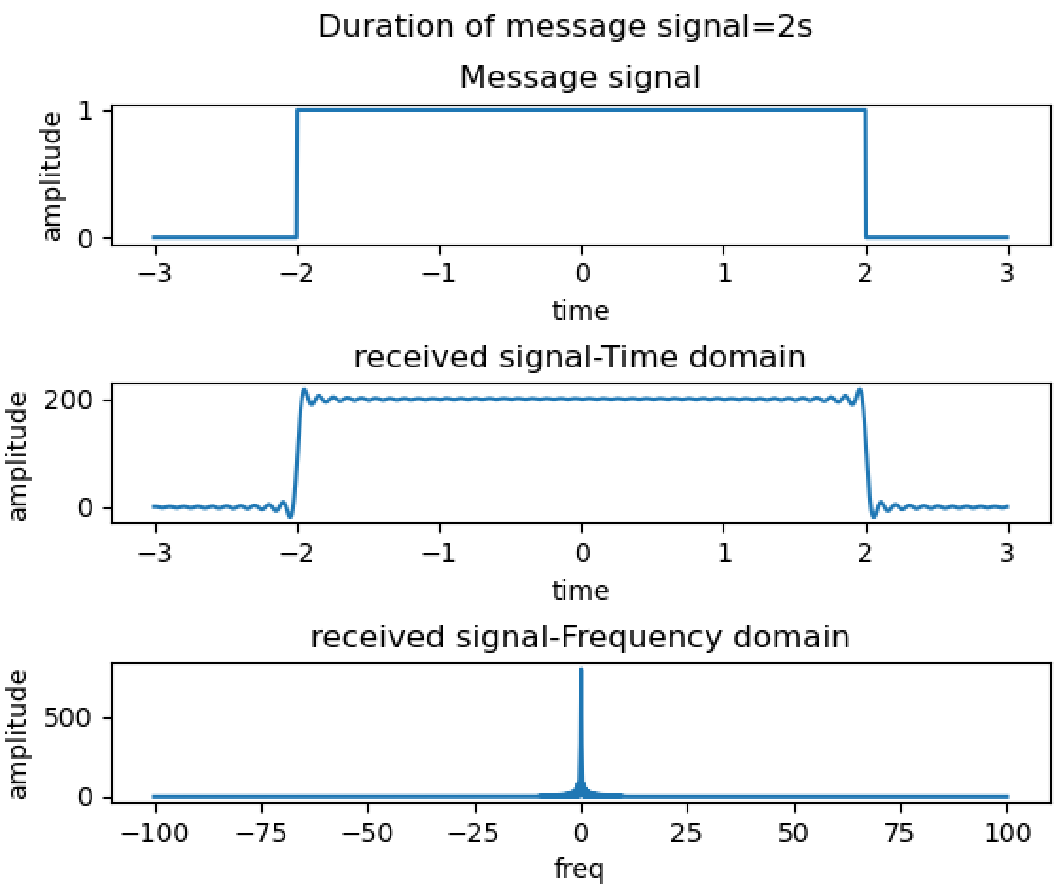


Plots with different durations

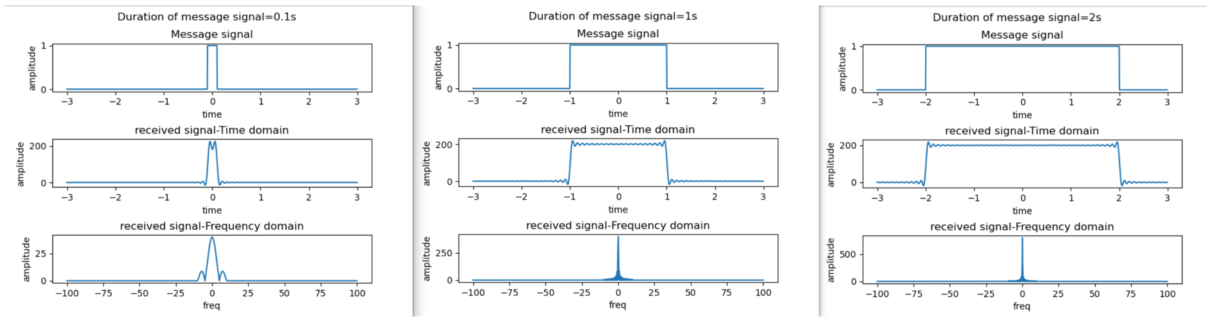
We have reduced the length of the square wave to bring out differences


Duration of message signal=0.1s
Message signal
amplitude
time
received signal-Time domain
amplitude
time
received signal-Frequency domain
amplitude
freq


Duration of message signal=1s
Message signal
amplitude
time
received signal-Time domain
amplitude
time
received signal-Frequency domain
amplitude
freq



Comparison



Comparison between different durations mentioned above

Observations

- The frequency spectrum resolution changes with the duration of the square signal