ASSIGNMENT-2 QUESTIONS

QUES1-

* Write a python program to create a *cosine wave* of frequency 2MHz with 256 samples per cycle.
* Plot it with proper annonation and axis labelling.
* Compute the FFT of the above signal and plot it.

QUES2-

* Create another a signal of frequency 3MHz, add it to above signal and do FFT for the resultant signal.
* Change the code such that the modulation frequency for 1 is 4MHz and for 0 it is 3MHz.
* Change the above code to simulate ASK modulation.

Ques3-

* Add demodulation to the above code and plot the time-domain waveform, as well as the FFT of the demodulated signal.
* Add a moving average filter to remove the high-frequency component from the demodulated signal.

QUES4-

* Add demodulation to the above code and plot the time-domain waveform, as well as the FFT of the demodulated signal.
* Add a moving average filter to remove the high-frequency component from the demodulated signal.

QUES5-

* Draw the spread-spectrum of the signal and also draw its FFT.

Q1.

#!/usr/bin/env python3

import numpy as np

import matplotlib.pyplot as plt

from scipy.fft import fft, fftfreq

## For saving plots to a file. Just couldn't get it to work from commandline

import matplotlib

matplotlib.use('Agg')

# Parameters

fc = 2e6 # Carrier Frequency

#fc2 = 2e6 # Carrier Frequency

fs = 256\*4e6 # Sampling frequency

ncycl = 32 # No of cycles of fc

Tsim = ncycl/fc # Total Simulation time

t = np.arange(0, Tsim, 1/fs) # Time vector

# Carrier signal

carrier = np.cos(2 \* np.pi \* fc \* t)

#carrier2 = np.cos(2 \* np.pi \* fc2 \* t)

#carrier = carrier + carrier2

#FFT

N = len(carrier)

yf = fft(carrier)

xf = fftfreq(N, 1/fs)

#Plotting

fig, axs = plt.subplots(2, 1)

axs[0].plot(t, carrier)

axs[0].set\_title('4MHz Signal')

axs[0].set\_xlim([0, Tsim])

#axs[0].set\_ylim([-1.2, 1.2])

#

axs[1].plot(xf,np.abs(yf))

axs[1].set\_title('FFT')

axs[1].set\_xlim([0, 2\*fc])

#axs[1].set\_ylim([-1.2, 1.2])

#plt.figure(figsize=(8,11))

plt.show();

plt.savefig("fft.png")

# Plot

#fig, axs = plt.subplots(2, 1)

#plt.figure(1)

#plt.clf()

#plt.plot(t,carrier)

#plt.grid()

#plt.show()

#plt.savefig("carrier.png")

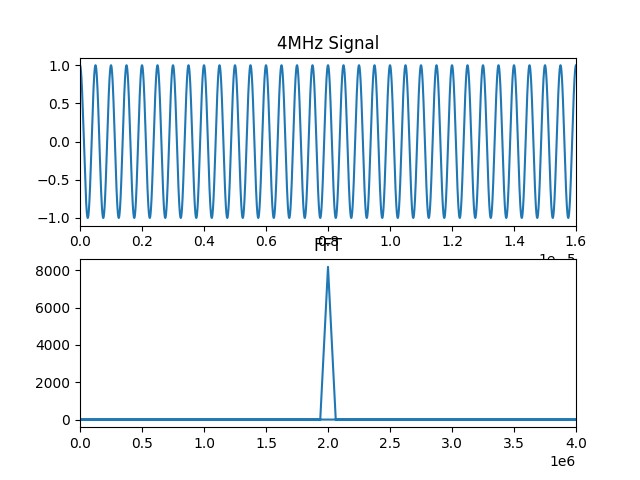
#

#plt.clf()

#plt.plot(xf,np.abs(yf))

#plt.xlim([0, 2\*fc])

#plt.savefig("fft.png")



Q2.

#!/usr/bin/env python3

import numpy as np

import matplotlib.pyplot as plt

from scipy.fft import fft, fftfreq

## For saving plots to a file. Just couldn't get it to work from commandline

import matplotlib

matplotlib.use('Agg')

# Parameters

fc = 3e6 # Carrier Frequency

#fc2 = 2e6 # Carrier Frequency

fs = 256\*3e6 # Sampling frequency

ncycl = 32 # No of cycles of fc

Tsim = ncycl/fc # Total Simulation time

t = np.arange(0, Tsim, 1/fs) # Time vector

# Carrier signal

carrier = np.cos(2 \* np.pi \* fc \* t)

#carrier2 = np.cos(2 \* np.pi \* fc2 \* t)

#carrier = carrier + carrier2

#FFT

N = len(carrier)

yf = fft(carrier)

xf = fftfreq(N, 1/fs)

#Plotting

fig, axs = plt.subplots(2, 1)

axs[0].plot(t, carrier)

axs[0].set\_title('4MHz Signal')

axs[0].set\_xlim([0, Tsim])

#axs[0].set\_ylim([-1.2, 1.2])

#

axs[1].plot(xf,np.abs(yf))

axs[1].set\_title('FFT')

axs[1].set\_xlim([0, 2\*fc])

#axs[1].set\_ylim([-1.2, 1.2])

#plt.figure(figsize=(8,11))

plt.show();

plt.savefig("fft1.png")

# Plot

#fig, axs = plt.subplots(2, 1)

#plt.figure(1)

#plt.clf()

#plt.plot(t,carrier)

#plt.grid()

#plt.show()

#plt.savefig("carrier.png")

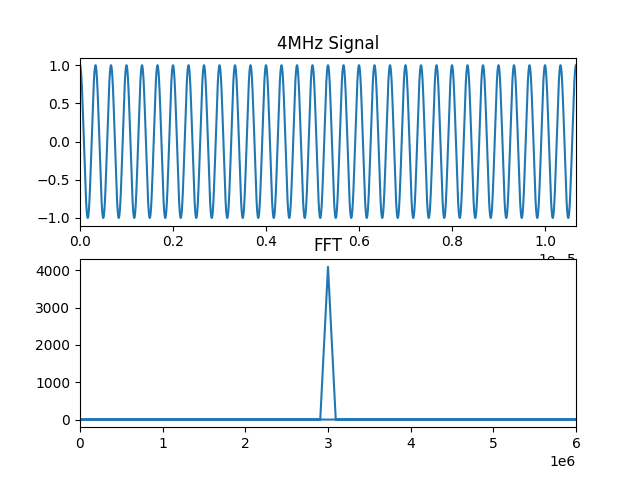
#

#plt.clf()

#plt.plot(xf,np.abs(yf))

#plt.xlim([0, 2\*fc])

#plt.savefig("fft.png")



Q3.

#!/usr/bin/env python3

import numpy as np

import matplotlib.pyplot as plt

from scipy.fft import fft, fftfreq

## For saving plots to a file. Just couldn't get it to work from commandline

import matplotlib

matplotlib.use('Agg')

# Parameters

fc0 = 4e6 # 1 Carrier Frequency

fc1 = 3e6 # 0 Carrier Frequency

fs = 256\*4e6 # Sampling frequency

ncycl = 512 # No of cycles of fc

nfc0 = 8 # number of fc0 cycles for one symbol

Tsim = ncycl/fc0 # Total Simulation time

t = np.arange(0, Tsim, 1/fs) # Time vector

# Message signal (binary data)

data = np.random.randint(0, 2, int(ncycl/nfc0)) # Random binary data

nupData = int(t.size/data.size)

data = np.repeat(data, nupData) # Upsample binary data

print(data.size, t.size)

# FSK Modulation

modulated\_signal = np.zeros\_like(t)

for i in range(len(t)):

if data[i] == 0:

modulated\_signal[i] = np.cos(2 \* np.pi \* fc0 \* t[i])

else:

modulated\_signal[i] = np.cos(2 \* np.pi \* fc1 \* t[i])

# FFT of the modulated signal

N = len(modulated\_signal)

yf = fft(modulated\_signal)

xf = fftfreq(N, 1 / fs)

# Plotting

fig, axs = plt.subplots(3, 1, figsize=(10, 12))

axs[0].plot(t, data)

axs[0].set\_title('Original Binary Data')

axs[0].set\_xlim([0, Tsim])

#axs[0].set\_ylim([-0.2, 1.2])

axs[1].plot(t, modulated\_signal)

axs[1].set\_title('FSK Modulated Signal')

axs[1].set\_xlim([0, Tsim])

axs[2].plot(xf, np.abs(yf))

axs[2].set\_title('FFT of Modulated Signal')

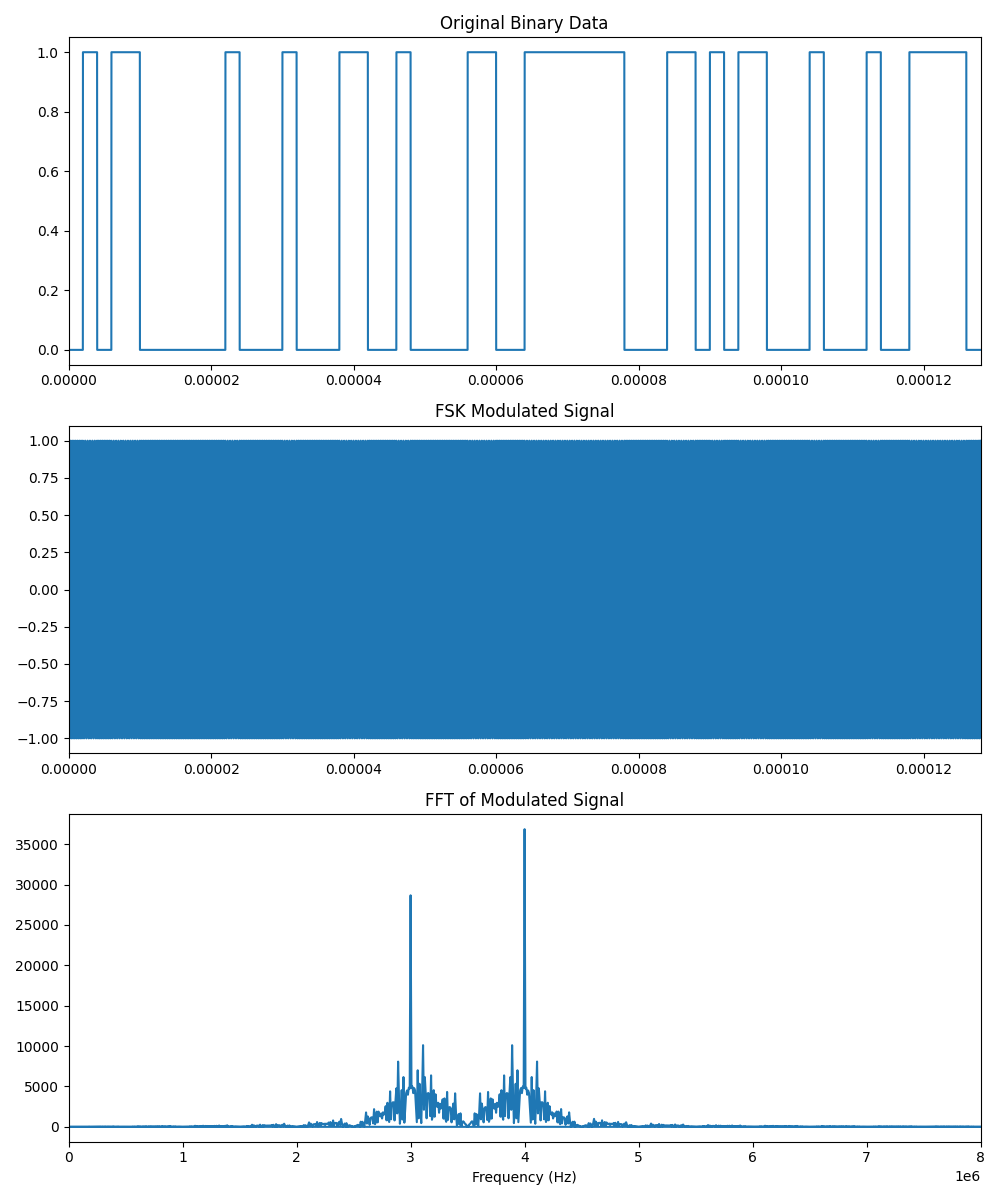
axs[2].set\_xlim([0, 2\*fc0])

axs[2].set\_xlabel('Frequency (Hz)')

plt.tight\_layout()

plt.show()

plt.savefig("fsk-lab2.png")



Q4.

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import butter, filtfilt

# Generate a sample AM signal

fs = 1000 # Sampling frequency

t = np.arange(0, 1.0, 1.0/fs) # Time vector

carrier\_freq = 100 # Carrier frequency in Hz

mod\_freq = 5 # Modulating frequency in Hz

mod\_index = 0.5 # Modulation index

carrier = np.cos(2 \* np.pi \* carrier\_freq \* t)

modulating\_signal = 1 + mod\_index \* np.cos(2 \* np.pi \* mod\_freq \* t)

am\_signal = carrier \* modulating\_signal

# Demodulation (envelope detection)

demodulated\_signal = np.abs(am\_signal)

# Low-pass RC filter design using butterworth filter

def rc\_low\_pass\_filter(data, cutoff, fs, order=5):

nyquist = 0.5 \* fs

normal\_cutoff = cutoff / nyquist

b, a = butter(order, normal\_cutoff, btype='low', analog=False)

y = filtfilt(b, a, data)

return y

cutoff\_freq = 10 # Cutoff frequency of the RC filter in Hz

filtered\_signal = rc\_low\_pass\_filter(demodulated\_signal, cutoff\_freq, fs)

# Plot the results

plt.figure(figsize=(12, 8))

plt.subplot(4, 1, 1)

plt.plot(t, modulating\_signal)

plt.title('Modulating Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 1, 2)

plt.plot(t, carrier)

plt.title('Carrier Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 1, 3)

plt.plot(t, am\_signal)

plt.title('AM Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.subplot(4, 1, 4)

plt.plot(t, demodulated\_signal, label='Demodulated Signal')

plt.plot(t, filtered\_signal, label='Filtered Signal')

plt.title('Demodulated and Filtered Signal')

plt.xlabel('Time [s]')

plt.ylabel('Amplitude')

plt.legend()

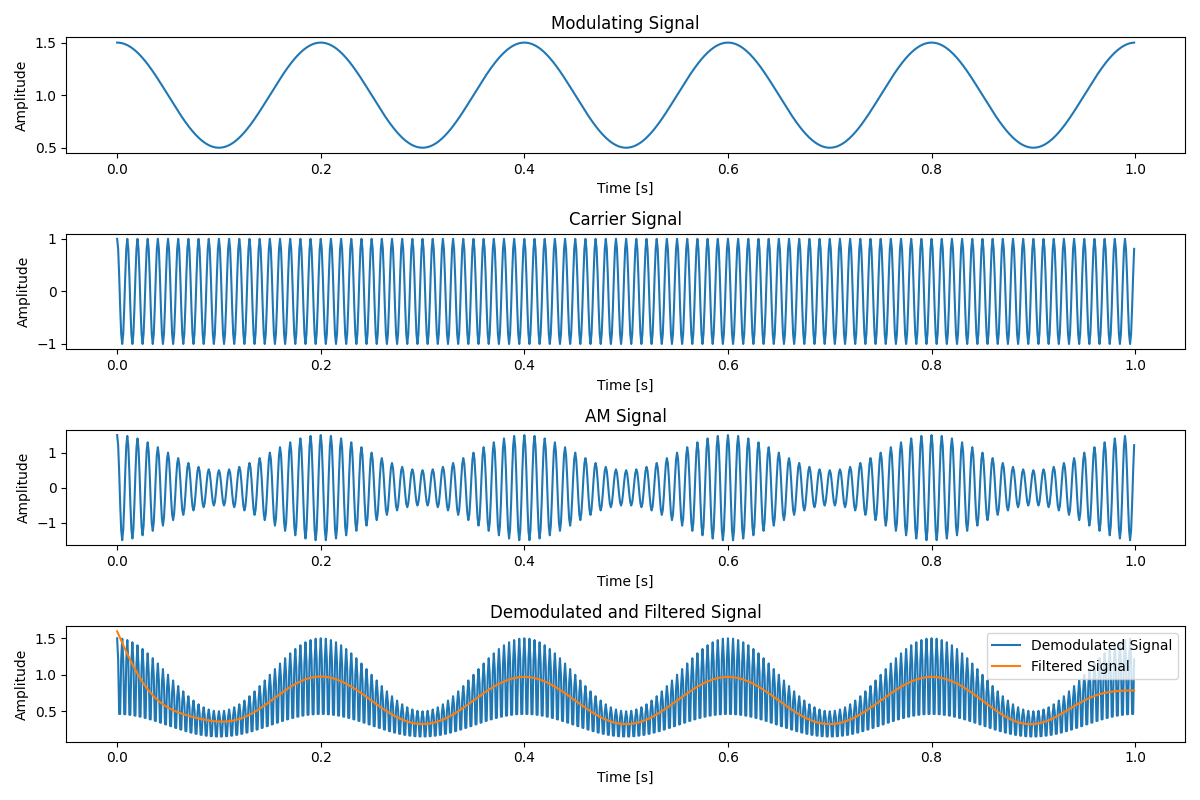
plt.tight\_layout()

# Save the plot as a PNG file

plt.savefig('demodulated\_and\_filtered\_signal.png')

# Display the plot (optional)

# plt.show()



Q5.

import numpy as np

import matplotlib.pyplot as plt

# Function to generate a pseudo-random noise (PN) sequence

def generate\_pn\_sequence(length, seed=0):

np.random.seed(seed)

return np.random.choice([1, -1], size=length)

# Function to modulate the message using DSSS

def dsss\_modulate(message, pn\_sequence):

return message \* pn\_sequence

# Function to demodulate the DSSS signal

def dsss\_demodulate(dsss\_signal, pn\_sequence):

return dsss\_signal \* pn\_sequence

# Generate a sample message signal

message\_length = 100

message = np.random.choice([1, -1], size=message\_length)

# Generate a PN sequence

pn\_sequence = generate\_pn\_sequence(message\_length, seed=42)

# Modulate the message using DSSS

dsss\_signal = dsss\_modulate(message, pn\_sequence)

# Demodulate the DSSS signal

demodulated\_signal = dsss\_demodulate(dsss\_signal, pn\_sequence)

# Plot the original message, PN sequence, DSSS signal, and demodulated signal

plt.figure(figsize=(12, 8))

plt.subplot(4, 1, 1)

plt.stem(message)

plt.title('Original Message')

plt.subplot(4, 1, 2)

plt.stem(pn\_sequence)

plt.title('PN Sequence')

plt.subplot(4, 1, 3)

plt.stem(dsss\_signal)

plt.title('DSSS Signal')

plt.subplot(4, 1, 4)

plt.stem(demodulated\_signal)

plt.title('Demodulated Signal')

plt.tight\_layout()

# Save the figure as an image file

plt.savefig('dsss\_signal.png')

# Display the plot

plt.show()

# Verify that the demodulated signal matches the original message

print("Original Message: ", message)

print("Demodulated Signal: ", demodulated\_signal)

