ECE 351 - 52

Signals and Systems 1

Lab 12

Submitted By :
Owen Blair

Contents

1 Important Notes 2
2 Project Description 3
2.1 Part 1 3
3 Identifying Interfering Noise Frequencies 3
3.1 Unfiltered Sensor Signal 4
3.2 Unfiltered Sensor Signal Magnitude Vs. Frequency 5
4 Analog Filter Design 5
4.1 Code to Calculate Component Values 5
4.2 Filter Circuit 6
5 Filter Validation 6
5.1 All Frequencies Bode Plot 7
5.2 Low Frequencies Bode Plot 8
5.3 Data Frequencies Bode Plot 9
5.4 High Frequencies Bode Plot 10
5.5 Filtered Signal FFT 11
6 Questions 11
6.1 1 11
7 Listings 11
7.1 Listing 1: Fast Fourier Transform Function 11
7.2 Listing 2: Make Stem Workaround 12
8 Appendix A 13

1 Important Notes

It is important to note that plt.show() is needed at the end of the python code to properly show the plots of each function. It was not included in every section of code that plots a function(s) because it is assumed that its included at the end the code. It is also assumed that the following is included at the beginning of the program:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sig
import scipy.fftpack as fft
import math as m
import pandas as pd

# The following package is not included with the Anaconda
    distribution
# and needed to be installed separately. The control package also
    has issues
# working on macs and a PC or a linux distribution is needed
import control as con
```

A note of caution for Mac users is that the control package has issues with the mac operating systems and needs to be run on a windows or Linux machine.

The web address to the GitHub where LaTeXcode is stored is here:
https://github.com/Blairis123/ECE351_Reports

The web address to the GitHub where the Python code is here:
https://github.com/Blairis123/ECE351_Code

The Code written for this lab is also attached in *Appendix A: Lab 12 Code*

2 Project Description

2.1 Part 1

The goal of this project is to filter a signal so that the position readings from a position sensor are not degraded by noise. This is done using the coding language Python to simulate a filter and provide mathematical analysis of signals. Specificaly, this is using Spyder 5.1.5 from the Anaconda Launcher.
The data sheet shows that the data coming from the position sensor is an AC waveform voltage in the range of $1.8kHz \geq \omega \leq 2.0kHz$ and noise needs to be seperated from this range of frequencies. This is done by first identifying what frequencies the noise is on. This is described in the section *Identifying Interfering Noise Frequencies*. Then a filter was designed so that:

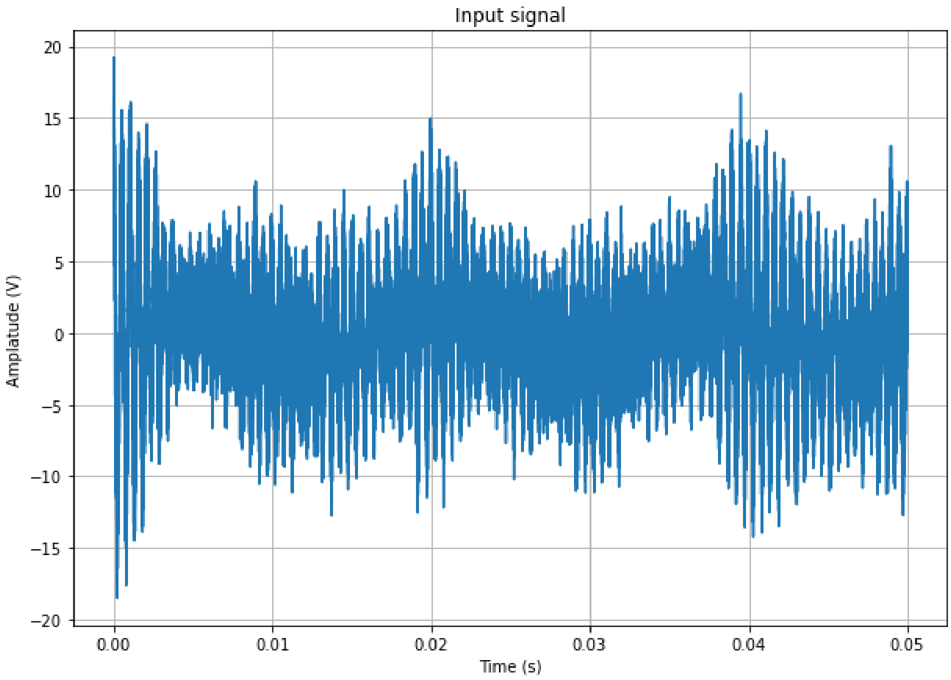
The sensor's information is attenuated by less than -0.3dB
The low frequency noise is attenuated by at least -30dB
The switching amplifier noise is attenuated by at least -21dB

The filter design can be seen in the section *Analog Filter Design* and the analyzing of the signal after the filter can be seen in the section *Filter Validation*.

3 Identifying Interfering Noise Frequencies

To identify the interference with the signal the signal was put through a Fast Fourier Transform function. The output of this function was then plotted with respect to frequency and was used to visually identify at what frequencies the noise in the signal was coming from. The function used to do so can be seen in *Listing 1: Fast Fourier Transform Function*. The unfiltered signal can be seen below.

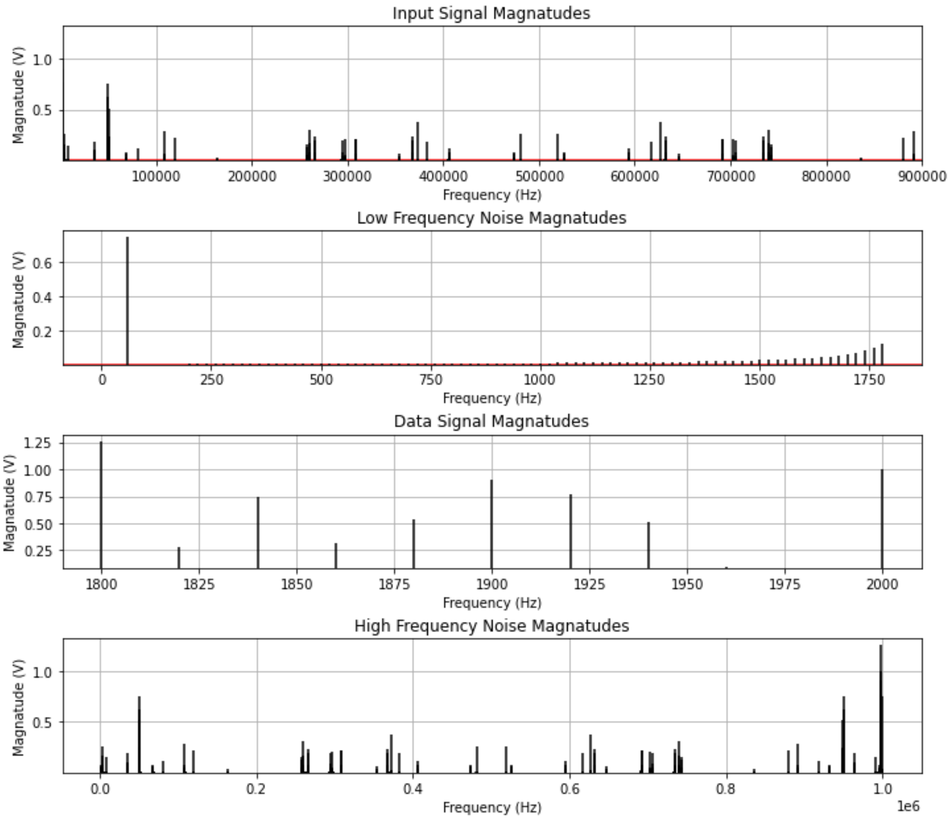
3.1 Unfiltered Sensor Signal



The following figure is the Fast Fourier Transform of the unfiltered signal. The figure is broken down into four parts. The first subplot is a plot of the magnitudes at their corresponding frequencies for the entirety of the input signal. The second subplot from the top is showing frequencies from 0kHz to 1.8kHz. These are frequencies below range where the position sensor is sending data. Third from the top is from 1.8kHz to 2kHz. These are the frequencies that contain the position sensor signals and should not be attenuated by the filter. At the bottom is a subplot that shows frequencies higher than 2kHz.

Frequencies that are lower than 1.8kHz are the low frequency HVAC noise signals. The frequencies that are higher than 2kHz are caused by the switching amplifier and other extraneous circumstances. The code for this project also relies on a code workaround that was provided and a modified version that was used for this project can be seen in *Listing 2: Make Stem Workaround*

3.2 Unfiltered Sensor Signal Magnitude Vs. Frequency



4 Analog Filter Design

The first iteration of a filter design used an online tool available at https://rf-tools.com/lc-filter/. The output from this online tool, although it worked was complicated to analyze. That is why the following RLC band-pass circuit was chosen. As seen below it is easy to analyze and the equations needed to estimate the values of the components was readily available though a textbook used in a previous class, ECE-212. The following was code also used to calculate the component values and print the result to the IPython console in Spyder.

4.1 Code to Calculate Component Values

```
    # Fist number is in Hz but is multiplied by 2pi for rads/sec
bandwidth = 800 * (2*np.pi)
centerFrq = 1.9e3 * (2 * np.pi)

```

```
R = 10
L = R / bandwidth
C = bandwidth / (centerFrq**2 * R)
print(R, L, C)
```

This resulted in the output below and was used to create the filter circuit.

```
10 0.0019894367886486917 3.5269793482968494e-06
```

4.2 Filter Circuit

5 Filter Validation

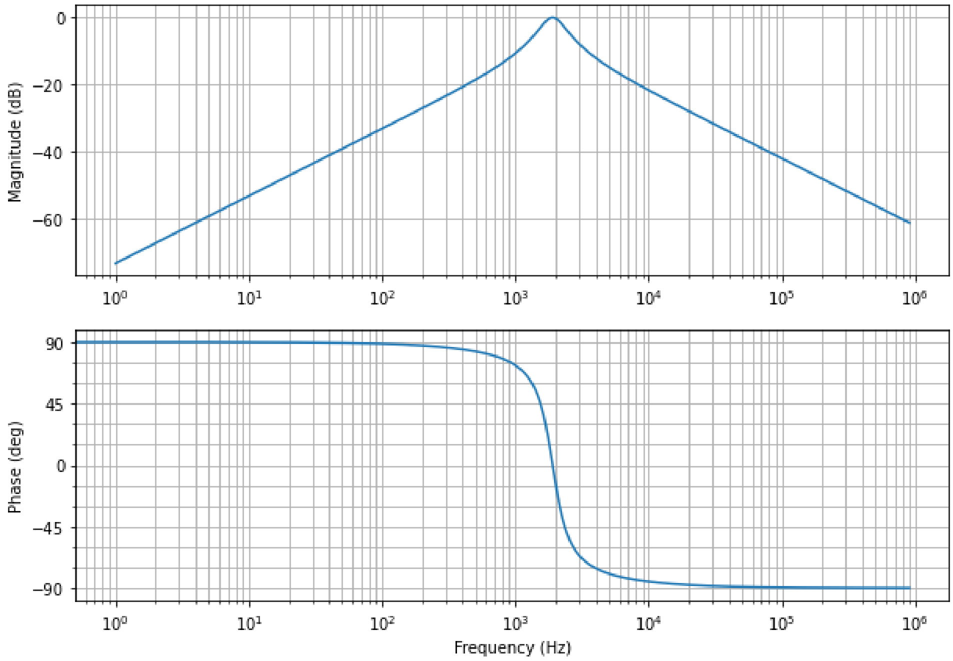
To ensure that the filter designed met the specifications the Bode plot of the filter was found along with the Fast Fourier Transform of the filtered signal. These were broken down into four plots. The first part is the entirety of the plot, the second is frequencies less than 1.8kHz, the third part are frequencies between 1.8 kHz and 2kHz that hold the position sensor data, and the last part are the frequencies above 2kHz.

The following code was used to calculate the transfer function based upon the component values. The variable filterNUM is a list of the exponents in the numerator of the s-domain transfer function and filterDEN is the denominator of the transfer function.

```
filterNUM = [R/L, 0]
filterDEN = [1, R/L, 1/(L*C)]
```

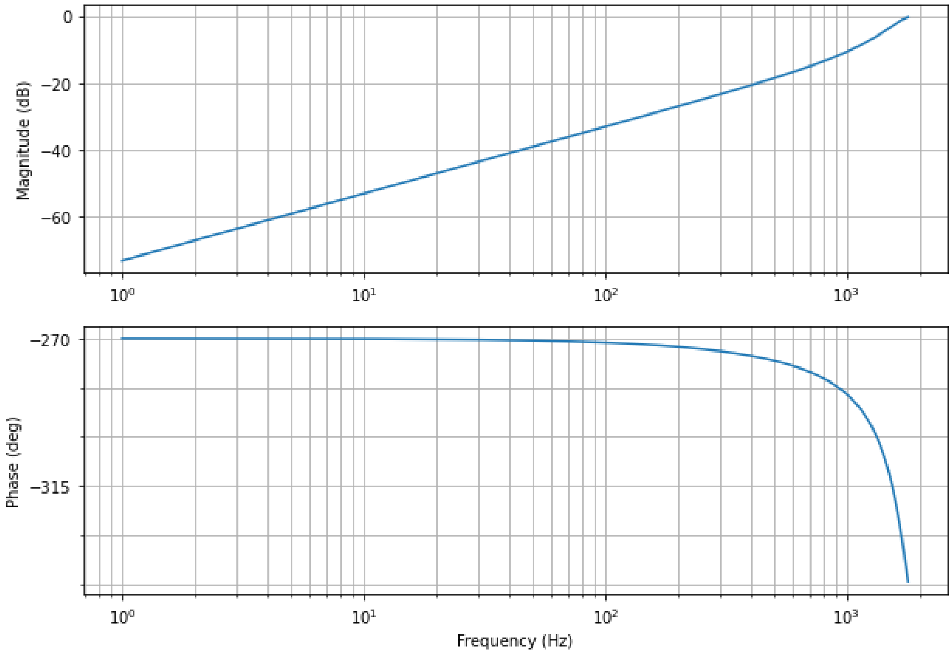
The resulting Bode plots can be seen below

5.1 All Frequencies Bode Plot



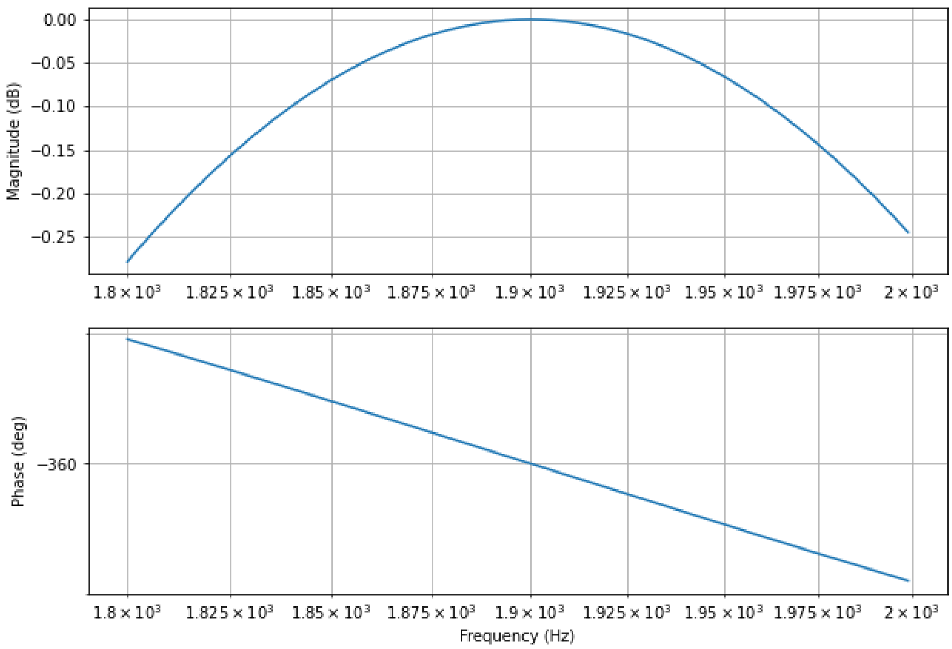
What is important about this plot is that by examination it can be found that at 60 Hz the signal is attenuated by more than -30db.

5.2 Low Frequencies Bode Plot

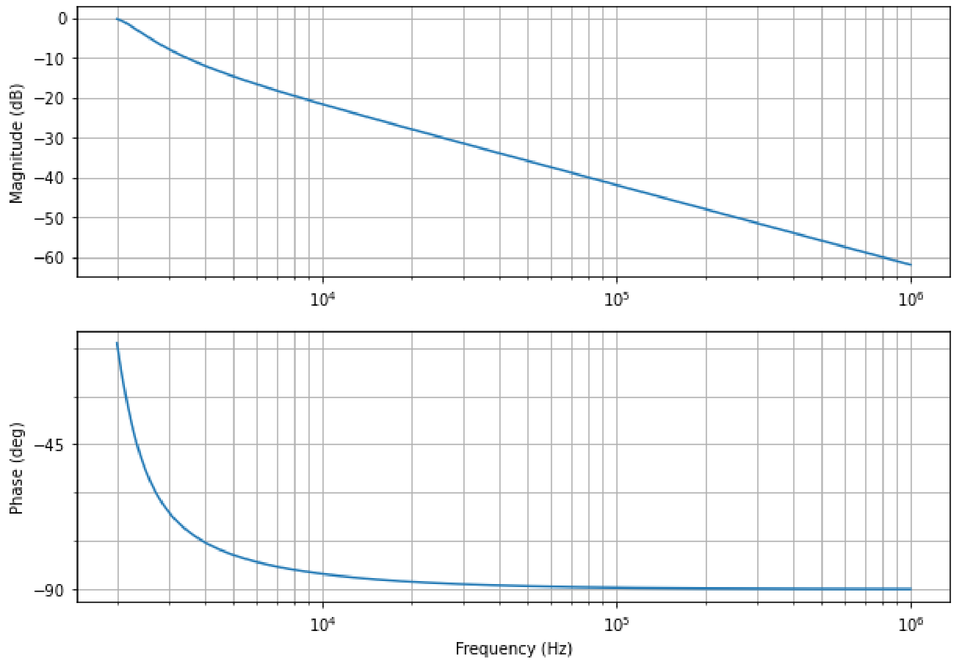


What is important about this plot is that the magnitude never goes below -0.3 dB. This shows that the requirement that the position sensor signal can not be attenuated by more than -0.3db is satisfied.

5.3 Data Frequencies Bode Plot

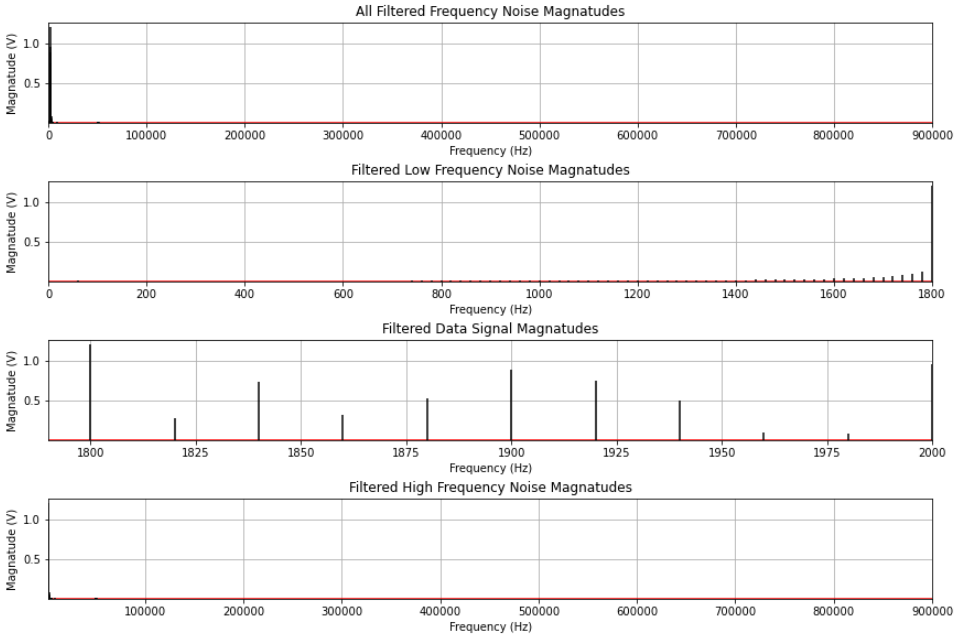


5.4 High Frequencies Bode Plot



After the input signal was put through the filter the results of the Fast Fourier Transform function were also recorded and plotted. This is to visually check that all frequencies besides the position sensor signal's have been properly attenuated. Another inturesting piece of information is having the input and output of the filter compared. This what the figure after the *Filtered Signal FFT* figure is. This provides a nice secondary check to see if the output still looks noisy compared to the input signal.

5.5 Filtered Signal FFT



6 Questions

6.1 1

I feel like I got what I wanted out of this course. I am much more confident in my Python skills and I feel like I could actually apply them to a problem in the real world.

7 Listings

7.1 Listing 1: Fast Fourier Transform Function

```
"""User defined fast Fourier transform function.
INPUTS:
    X, a function array
    fs, frequency of sampling rate

OUTPUTS:
    X_freq, frequency array corresponding to FFT
    X_mag, array of FFT magnitudes from fast fourier transform
    X_phi, array of FFT angles from fast fourier transform

```

```
NOTES:
    Any magnitude that is less than 1e-10 will also have a corresponding angle
    of zero. This serves to "clean up" the phase angle plot.
"""
def FFT(X, fs):

    # Length of input array
    n = len(X)

    # Preform fast fourier transform
    X_fft = fft.fft(X)

    """
    Will not use shifted because the frequencies that are needed will be real
    and won't have a negative value!!!!!

    # shift zero frequency to center of the spectrum
    X_fft_shift = fft.fftshift(X_fft)
    """

    # Calculate frequencies for output. fs is sampling frequency
    X_freq = np.arange(0, n) * fs / n

    # Calculate magnitude and phase
    X_mag = np.abs(X_fft)/n
    X_phi = np.angle(X_fft)

    # Clean up the phase array!
    for i in range(len(X_phi)):
        if ( X_mag[i] < 1e-10):
            X_phi[i] = 0

    # Return values!
    return X_freq, X_mag, X_phi
```

7.2 Listing 2: Make Stem Workaround

```
"""
This is a faster version of matplot.pyplot.stem() plotting. This is from the
lab handout and is provided from the TA.

MODIFIED FROM LAB HANDOUT BY OWEN BLAIR 11/22/2021
    This addition of the return statement allows the programmer to set an axis
    from the return of this function. I was having issues with the
```

```
    code from
     the lab handout not working. The error I was getting was

     ValueError: Single argument to subplot must be a three-digit
    integer, not
     AxesSubplot(0.125,0.536818;0.775x0.343182)"

     I would also get random "list has no attribute 'min'" errors
    with:
     ax.set_ylim ([1.05 * y.min(), 1.05 * y.max()])

     After a bit of googleing, sometimes there are issues with the
    list.min()
     function. No issues yet with the min(list) function change
    ..... yet.....
"""
def make_stem(ax ,x,y,color='k',style='solid',label='',linewidths
    =1.5 ,** kwargs):
     ax.axhline(x[0],x[-1],0, color='r')
     ax.vlines(x, 0 ,y, color=color , linestyles=style , label=label
      , linewidths= linewidths)

     # This has been modified
     ax.set_ylim ([1.05 * min(y), 1.05 * max(y)])

     # This line has been added
     return ax
```

8 Appendix A

```
################################################################
#
# Owen Blair
# ECE351-52
# Lab #12
# 12/9/2021
#
################################################################

import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sig
import scipy.fftpack as fft
import math as m
import pandas as pd

# The following package is not included with the Anaconda
```

```
distribution
# and needed to be installed separately. The control package also has issues
# working on macs and a PC or a linux distribution is needed
import control as con

#-----------FUNCTIONS!!!!!!!----------------------------

"""
This is a faster version of matplot.pyplot.stem() plotting. This is from the
lab handout and is provided from the TA.

MODIFIED FROM LAB HANDOUT BY OWEN BLAIR 11/22/2021
    This addition of the return statement allows the programmer to set an axis
    from the return of this function. I was having issues with the code from
    the lab handout not working. The error I was getting was

    ValueError: Single argument to subplot must be a three-digit integer, not
    AxesSubplot(0.125,0.536818;0.775x0.343182)"

    I would also get random "list has no attribute 'min'" errors with:
    ax.set_ylim ([1.05 * y.min(), 1.05 * y.max()])

    After a bit of googleing, sometimes there are issues with the list.min()
    function. No issues yet with the min(list) function change ..... yet.....
"""
def make_stem(ax ,x,y,color='k',style='solid',label='',linewidths=1.5 ,** kwargs):
    ax.axhline(x[0],x[-1],0, color='r')
    ax.vlines(x, 0 ,y, color=color , linestyles=style , label=label , linewidths= linewidths)

    # This has been modified
    ax.set_ylim ([1.05 * min(y), 1.05 * max(y)])

    # This line has been added
    return ax


"""User defined fast fourier transform funnction.
INPUTS:
    X, a function array
```

```
    fs, frequency of sampleing rate

OUTPUTS:
    X_freq, frequency array coresponding to FFT
    X_mag, array of FFT magnatudes from fast fourier transform
    X_phi, array of FFT angles from fast fourier transform

NOTES:
    Any magnatude that is less than 1e-10 will also have a coresponding angle
    of zero. This serves to "clean up" the phase angle plot.
"""
def FFT(X, fs):

    # Length of input array
    n = len(X)

    # Preform fast fourier transorm
    X_fft = fft.fft(X)

    """
    Will not use shifted because the frequencies that are needed will be real
    and won't have a negative value!!!!!

    # shift zero frequency to center of the spectrium
    X_fft_shift = fft.fftshift(X_fft)
    """

    # Calculate frequnecies for output. fs is sampling frequency
    X_freq = np.arange(0, n) * fs / n

    # Calculate magnatude and phase
    X_mag = np.abs(X_fft)/n
    X_phi = np.angle(X_fft)

    # Clean up the phase array!
    for i in range(len(X_phi)):
        if ( X_mag[i] < 1e-10):
            X_phi[i] = 0

    # Return values!
    return X_freq, X_mag, X_phi


# ----------------------------------------------------------------------

```

```

# Import signal
df = pd.read_csv('NoisySignal.csv')

t = df['0'].values
sensor_sig = df['1'].values

"""
Uncomment the follwoing to plot the input signal
"""
# Plott the signal!
plt.figure(figsize = (10, 7))
plt.plot(t, sensor_sig)
plt.grid()
plt.title("Input signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplatude (V)")
plt.show()



"""
This part uses the fast fourier transform to identify the noise and
    the
signal's frequencies and magnatudes (Modified from lab 9)
"""

    # Set sampling frequency
fs=1e6

lowFrq = []
lowMag = []

dataFrq = []
dataMag = []

highFrq = []
highMag = []

    # Run through signal with FFT
X_freq, X_mag, X_phi = FFT(sensor_sig, fs)

for i in range(len(X_freq)):

    if (X_freq[i] < 1.8e3):
        lowFrq.append(X_freq[i])
        lowMag.append(X_mag[i])

    if ((X_freq[i] <= 2e3) and (X_freq[i] >= 1.8e3)):
```

```python
        dataFrq.append(X_freq[i])
        dataMag.append(X_mag[i])

    if (X_freq[i] > 2e3):
        highFrq.append(X_freq[i])
        highMag.append(X_mag[i])

    # Plot FFT stuff!
gridSize = (5,1)
fig = plt.figure(figsize = (10, 10), constrained_layout = True)

    # Magnatude of input signal
inputFFTMagAx = plt.subplot2grid(gridSize, (0,0))# Make axis object to modify
inputFFTMagAx = make_stem(inputFFTMagAx, X_freq, X_mag)
inputFFTMagAx.set_title("Input Signal Magnatudes")
inputFFTMagAx.set_ylabel("Magnatude (V)")
inputFFTMagAx.set_xlabel("Frequency (Hz)")
inputFFTMagAx.set_xlim(0, 9e5)
inputFFTMagAx.grid()

    # Low ( < 1.8 kHz) frequency zoom
lowFreqAx = plt.subplot2grid(gridSize, (1,0))
lowFreqAx = make_stem(lowFreqAx, lowFrq, lowMag)
lowFreqAx.set_title("Low Frequency Noise Magnatudes")
lowFreqAx.set_ylabel("Magnatude (V)")
lowFreqAx.set_xlabel("Frequency (Hz)")
#inputFFTMagAx.set_xlim(0, 1.8e3)
lowFreqAx.grid()

    # Data frequency (1.8 kHz < frq < 2.0 kHz)
dataFreqAx = plt.subplot2grid(gridSize, (2,0))
dataFreqAx = make_stem(dataFreqAx, dataFrq, dataMag)
dataFreqAx.set_title("Data Signal Magnatudes")
dataFreqAx.set_ylabel("Magnatude (V)")
dataFreqAx.set_xlabel("Frequency (Hz)")
#inputFFTMagAx.set_xlim(1.8e3, 2e3)
dataFreqAx.grid()

    # High frequency (between 2 kHz and 50 kHz) magnatudes
highFreqAx = plt.subplot2grid(gridSize, (3,0))
highFreqAx = make_stem(highFreqAx, highFrq, highMag)
highFreqAx.set_title("High Frequency Noise Magnatudes")
highFreqAx.set_ylabel("Magnatude (V)")
highFreqAx.set_xlabel("Frequency (Hz)")
inputFFTMagAx.set_xlim(2e3, 9e5)
highFreqAx.grid()

plt.show()
```

```


# FILTER
    !-------------------------------------------------------------------
    # Numerator and denominator for transfer function H(s)
# The following was from a previous filter that was complacated but
     had a bode
# plot that sugested that it should work but still had a spike at
    10e6 above
# 0.05 V

#filterNUM = [9e8, 0, 0]
#filterDEN = [18, 2.7e6, 2.34e10, 3.75e14, 1.5625e18]

"""
    The follwing is calculated using the series bandpass filter
    from ECE-212 (Karen's circuits class) and is used to calculate
    component values
"""
    # Fist number is in Hz but is multiplied by 2pi for rads/sec
bandwidth = 800 * (2*np.pi)
centerFrq = 1.9e3 * (2 * np.pi)

#R = 10
#L = R / bandwidth
#C = bandwidth / (centerFrq**2 * R)
#print(R, L, C)

R = 10
L = 1.989e-3
C = 3.527e-6

filterNUM = [0, R/L, 0]
filterDEN = [1, R/L, 1/(L*C)]


    # Transfer funciton object, num and den are defined above
H_s = con.TransferFunction(filterNUM, filterDEN)

# Define an omega for transfer function
    # Define step size
steps = 1

    # w for part 1
start = 0
stop = 9e5
    #Define a range of w, with a stepsize of step
w = np.arange(start, stop, steps)
```

```

# Make and show Bode plots!
    # Make the Bode plot
plt.figure(figsize = (10, 7))
plt.title("All Frequencies")
Mag, Phi, bodeW = con.bode(H_s, w * 2 * np.pi, dB=True, Hz=True,
    deg=True, plot=True)

plt.figure(figsize = (10, 7))
plt.title("Frequencies Below Data")
Mag, Phi, bodeW = con.bode(H_s, np.arange(1, 1.8e3, 10)* 2 * np.pi,
     dB=True, Hz=True, deg=True, plot=True)

plt.figure(figsize = (10,7))
Mag, Phi, bodeW = con.bode(H_s, np.arange(1.8e3, 2e3, 1)* 2 * np.pi
    , dB=True, Hz=True, deg=True, plot=True)

plt.figure(figsize = (10,7))
Mag, Phi, bodeW = con.bode(H_s, np.arange(2e3, 1e6, 10)* 2 * np.pi,
     dB=True, Hz=True, deg=True, plot=True)


# Filter the signal!
    # Move transfer function into z-domain
numZ, denZ = sig.bilinear(filterNUM, filterDEN,fs)

    # Pass signal through filter
filteredSignal = sig.lfilter(numZ, denZ, sensor_sig)

# Plott the filtered signal!
plt.figure(figsize = (10, 7))
plt.title("Filtered Sensor Signal")
plt.ylabel("Filtered Signal Amplatude")
plt.xlabel("Time (s)")
plt.plot(t,filteredSignal)
plt.grid()

plt.show()

ratioMult = 1

    # Do some ploting!
plt.figure(figsize=(20*ratioMult, 10*ratioMult))
plt.figure(constrained_layout=True)
plt.subplot(2,1,1)
plt.title("Filtered Vs. Unfiltered Signal Comparison")
plt.ylabel("Filtered Signal Amplatude")
plt.xlabel("Time (s)")
plt.plot(t,filteredSignal)
```

```
plt.grid()

plt.subplot(2,1,2)
plt.plot(t, sensor_sig)
plt.grid()
plt.title("Unfiltered Input Signal")
plt.xlabel("Time (s)")
plt.ylabel("Unfiltered Signal Amplatude")

plt.show()



#Put filtered signal through FFT
    -------------------------------------------
    # Run through signal with FFT
filteredFreq, filteredMag, filteredPhi = FFT(filteredSignal, fs)


"""
    The following uses the make stem function and subplot2grid
    functions
    to make a figure with multiple subplots. The variable gridSize
    controls
    how the subplots are arranged within the figure. The figsize
    variable
    controls the ratio of the figure size.
"""


# Ploting filtered signal through FFT
    # Plot FFT stuff!
gridSize = (4,1) # This is the size of the grid
fig = plt.figure(figsize = (12, 8), constrained_layout = True)


    # All frequency magnatudes
filteredFreqAx = plt.subplot2grid(gridSize, (0,0))
filteredFreqAx = make_stem(filteredFreqAx, filteredFreq,
    filteredMag)
filteredFreqAx.set_title("All Filtered Frequency Noise Magnatudes")
filteredFreqAx.set_ylabel("Magnatude (V)")
filteredFreqAx.set_xlabel("Frequency (Hz)")
filteredFreqAx.set_xlim(0, 9e5)
filteredFreqAx.grid()

    # Low ( < 1.8 kHz) frequency zoom
filteredLowFreqAx= plt.subplot2grid(gridSize, (1,0))
filteredLowFreqAx= make_stem(filteredLowFreqAx, filteredFreq,
```

```
    filteredMag)
filteredLowFreqAx.set_xlim(0,1.8e3)
filteredLowFreqAx.set_title("Filtered Low Frequency Noise Magnatudes")
filteredLowFreqAx.set_ylabel("Magnatude (V)")
filteredLowFreqAx.set_xlabel("Frequency (Hz)")
filteredLowFreqAx.grid()

    # Data frequency (1.8 kHz < frq < 2.0 kHz)
filteredDataFreqAx = plt.subplot2grid(gridSize, (2,0))
filteredDataFreqAx = make_stem(filteredDataFreqAx, filteredFreq, filteredMag)
filteredDataFreqAx.set_xlim(1.79e3, 2e3)
filteredDataFreqAx.set_title("Filtered Data Signal Magnatudes")
filteredDataFreqAx.set_ylabel("Magnatude (V)")
filteredDataFreqAx.set_xlabel("Frequency (Hz)")
filteredDataFreqAx.grid()

    # High frequency (above 2kHz) magnatudes
filteredHighFreqAx = plt.subplot2grid(gridSize, (3,0))
filteredHighFreqAx = make_stem(filteredHighFreqAx, filteredFreq, filteredMag)
filteredHighFreqAx.set_title("Filtered High Frequency Noise Magnatudes")
filteredHighFreqAx.set_ylabel("Magnatude (V)")
filteredHighFreqAx.set_xlabel("Frequency (Hz)")
filteredHighFreqAx.set_xlim(2.1e3, 9e5)
filteredHighFreqAx.grid()

plt.show()
```