ECE 351 - 52

Signals and Systems 1

Lab 10

Submitted By :
Owen Blair

Contents

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

# 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 not 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

2 Part 1 Deliverables

2.1 Part 1

Part 1 is developing a frequency response to the RLC circuit shown below and present it as a Bode plots.

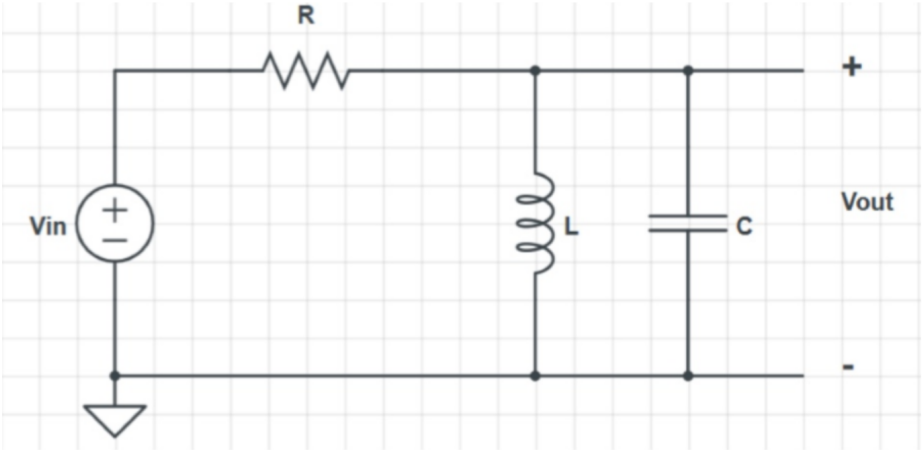


Figure 1: R = 1kΩ, L = 27 mH, C = 100 nF

This circuit has a transfer function of

$$H(s) = \frac{\frac{1}{RC}s}{s^2 + \frac{1}{RC}s + \frac{1}{LC}}$$

The Bode plots for this part are plotted in the rage of frequencies around $10^3 \leq \omega \leq 10^6$ in $\frac{rad}{s}$. The semilogx function in matplotlib.pyplot should be used instead of the regular plot function so that the Bode plot displays correctly. The code used to make the Bode plot can be found in *Listing 1: User Generated Bode Plot Code without Phase Adjustment.* This section of code originally gave a plot that had a jump from 180° to 0°. This jump can be seen in *Figure 1: np.arctan vs np.arctan2.*

The arctan 2 function has an output that is wider than np.arcta. The np.arctan function's output is limited to values between 0° and 180° where np.arctan2 can output values that are negative. This discontinuity was fixed with a piece of code shown in *Listing 2: np.acrtan Loop Adjustment* and is compared to np.arctan2 in *Figure 2: np.arctan with Loop Adjustment vs. np.arctan2.* After adjusting the phase of the Bode plot the code in *Listing 2* was added after the program calculated the output of the phase as to adjust the output of the phase before it was plotted.

It is also useful to plot the Bode plot in terms of Hertz instead of radians per second. This was done using the control package and two lists, one for the numerator of the transfer function and one for the denominator of the transfer function. This code used to do this is available in *Listing 3: Control Package Bode Plot in Hertz*. The plot of the user generated Bode plot can be seen in *Figure 3: User Generated Bode Plot* and the Control package generated Bode plot can be seen in *Figure 4: Control Package Generated Bode Plot*.

2.2 Part 2

Part 2 is using the frequency response of the RLC circuit to filter a multi-band input signal. The signal being put through the filter is $x(t) = cos(2\pi * 100t) + cos(2\pi * 3024t) + sin(2\pi 50000t)$. This signal is plotted in the time domain of $0 \leq t \leq 0.01$ seconds. This transfer function must be converted into its z-domain equivalent before using it to filter the signal. This is done using the scipy.signal.bilinear() function with the inputs of the numerator of the transfer function, denominator of the transfer function, and the sampling frequency. Then the signal can be filtered using the scipy.signal.lfilter() function with inputs of the transfer function's z-domain numerator and denominator, and the signal. The plots of the input signal and the filtered input signal can be seen in *Figure 5: Input Signal and Filtered Signal Output*. The code used to generate the plots and the input and output signals can be seen in *Listing 4: Signal Filtering and Plotting Code*.

3 Questions

3.1 Question 1

The filters output makes sense given the Bode plots because the filter has a peak at zero. This means that when an input signal has a frequency at a frequency that corresponds to the Bode plot's peak then the signal will go through. If the frequency is above or below the peak then it will be filtered out. This can be seen in the filtered signal by the first few oscillations of the wave being dampened than the rest of the function. This is the cosine parts of the input signal being filtered out leaving the sine parts of the input signal alone.

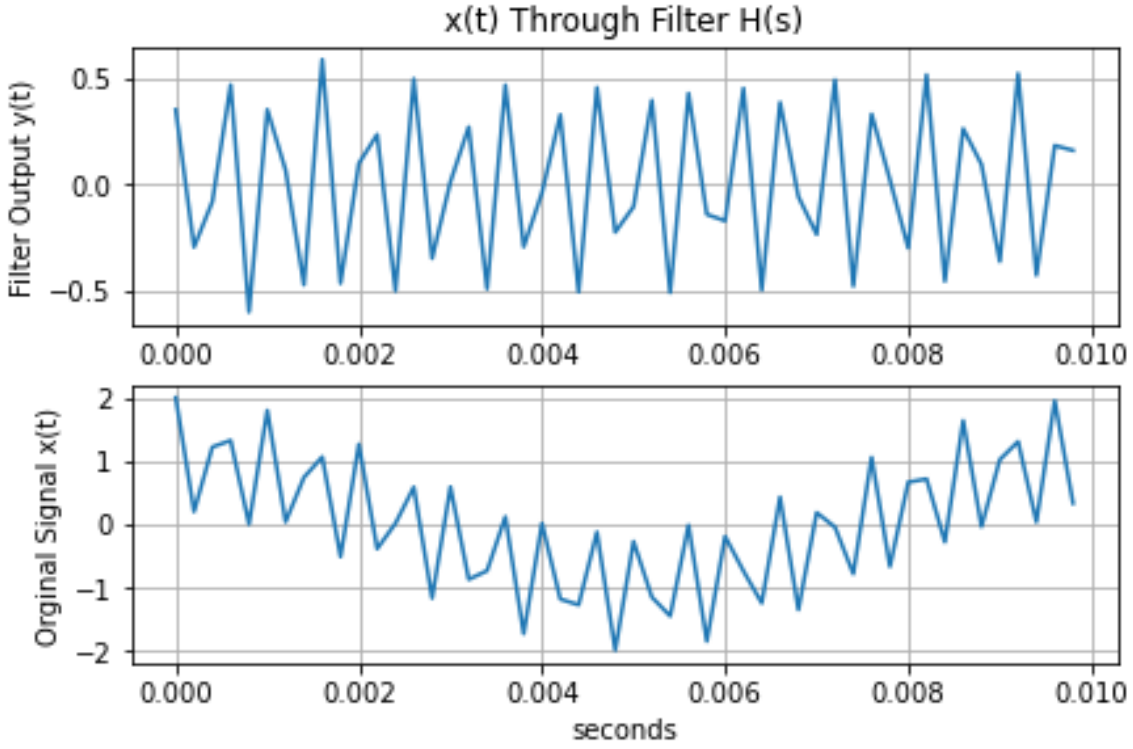
3.2 Question 2

The purpose of scipy.signal.bilinear() is to convert a transfer function in the Laplace domain to the equivalent z-domain function. The purpose of scipy.signal.lifilter() is

to filter a signal using a z-domain transfer function. This allows for a user to take a Laplace domain transfer function and use it to filter signals in Python.

3.3 Question 3

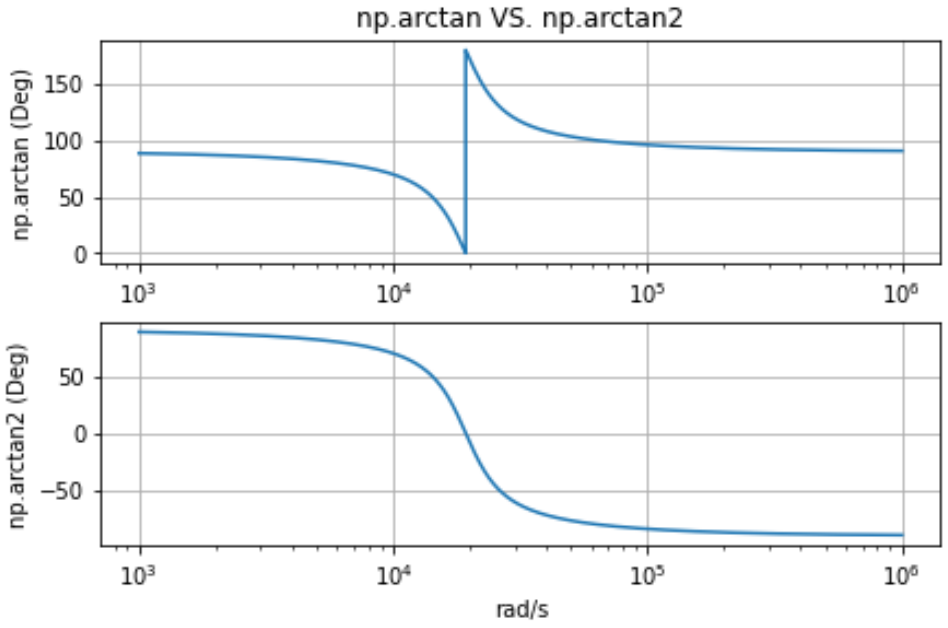
If a diffrent sampling frequency is used in the scipy.signal.bilinear() function then the output from the function may not be calculated currently. For example, the following image is of the input and output signal when the sampling frequency is lower than one of the sine function's frequency.



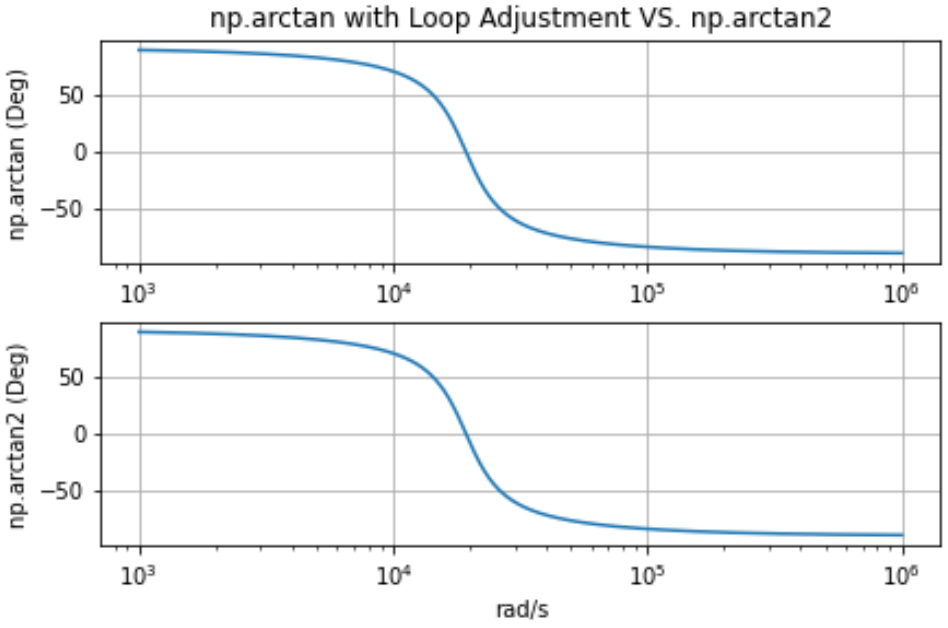
Looking at the graph, the sampling frequency is not high enough to accurately capture the higher frequencies. At extremely high sampling frequencies, like say 10^{10}, Spyder chugs through the calculations and it takes a long time to compute the output.

4 Figures

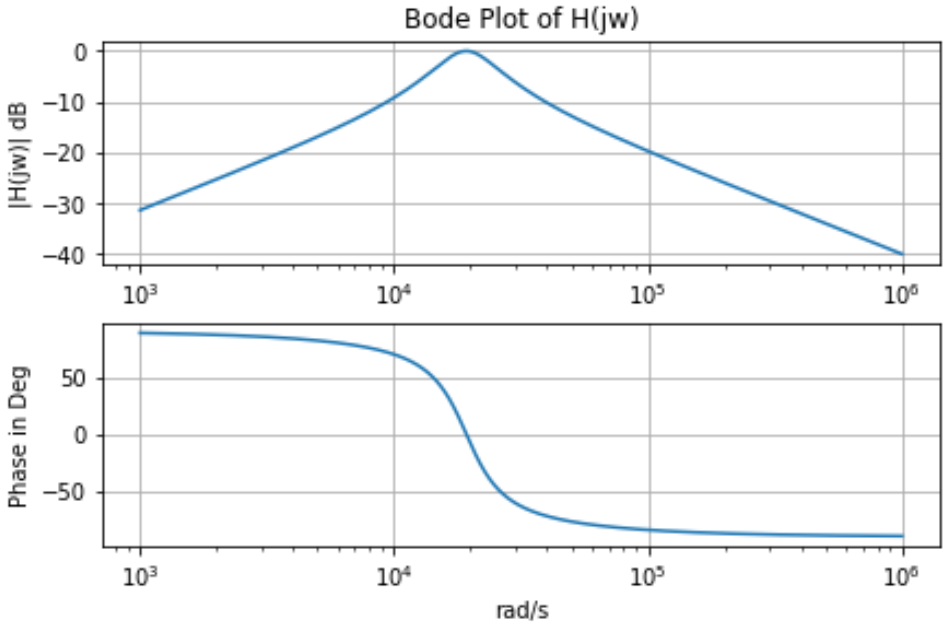
4.1 Figure 1: np.arctan vs np.arctan2



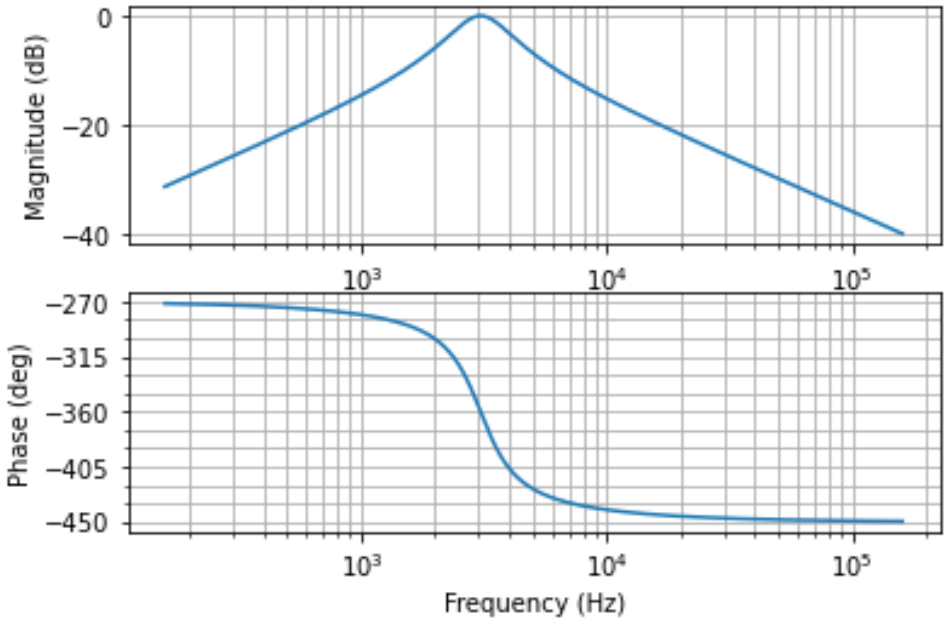
4.2 Figure 2: np.arctan Loop Adjustment vs. np.arctan2



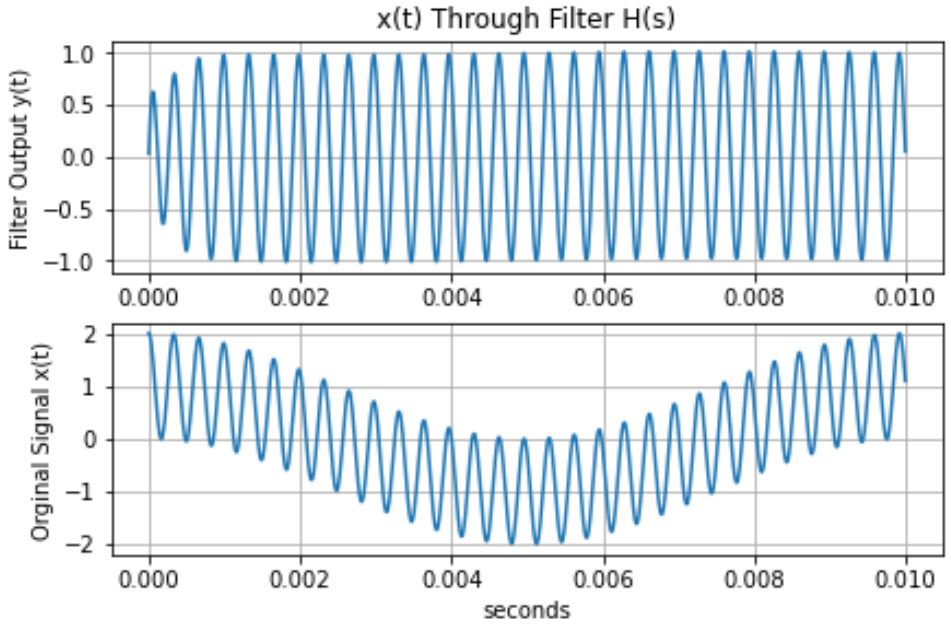
4.3 Figure 3: User Generated Bode Plot



4.4 Figure 4: Control Package Generated Bode Plot



4.5 Figure 5: Signal Filtering and Plotting Code



5 Listings

5.1 Listing 1: User generated Bode Plot Code Without Phase Adjustment

```
#---------PART 1-----------------------------------#
    #Define step size
steps = 1

    #t for part 1
start = 1e3
stop = 1e6
    #Define a range of w, with a stepsize of step
w = np.arange(start, stop, steps)

#------------------------------TASK 1, Part 1---------------#
    # Data from RLC circuit
R=1e3
L=27e-3
C=100e-9
    # Magnatude function and phase function. Keep in mind that
    # np.arctan will return a value in radians
H_mag = (w/(R*C)) / np.sqrt( w**4 + ( (1/(R*C))**2 - (2/(L*C))) * (
    w**2) + (1/(L*C))**2)
```

```
H_phi = (np.pi/2) - np.arctan( (w/(R*C)) / (-1*(w**2) + (1/(L*C))) )

    # Convert to Db and deg
H_magDb = 20 * np.log10(H_mag)
H_phiDeg = (180/np.pi) * H_phi

    # Plot the magnatude
plt.figure(figsize=(10, 7))
plt.figure(constrained_layout=True)
plt.subplot(2,1,1)
plt.title("Bode Plot of H(jw)")
plt.ylabel("|H(jw)| dB")
plt.semilogx(w, H_magDb)
plt.grid()

plt.subplot(2,1,2)
plt.ylabel("Phase in Deg")
plt.xlabel("rad/s")
plt.semilogx(w, H_phiDeg)
plt.grid()
```

5.2 Listing 2: np.arctan Loop Adjustment Code

```
    # loop to fix the jump in the phase plot
for i in range(0,len(H_phiDeg)):
    #Do stuff
    if(H_phiDeg[i]>90):
        H_phiDeg[i] = H_phiDeg[i] - 180
```

5.3 Listing 3: Control Package Bode Plot

```
    # Transfer function numerator and den.
num = [(1/(R*C)),0]
den = [1,1/(R*C),1/(L*C)]

    # Need a name change for omega array
w1 = w
sigW, sigBodeMag, sigBodePhi = sig.bode(([(1/(R*C)),0], [1,1/(R*C),1/(L*C)]), w=w1)

    # Start plot!
plt.figure(figsize=(10,7))
plt.figure(constrained_layout=True)
plt.subplot(2,1,1)
plt.title("scipy.signal.bode Bode Plot of H(s)")
plt.ylabel("dB")
```

```
plt.semilogx(w, sigBodeMag)
plt.grid()

plt.subplot(2,1,2)
plt.ylabel("deg")
plt.semilogx(w, sigBodePhi)
plt.xlabel("rad/s")
plt.grid()
```

5.4 Listing 4: Signal Filtering and Plotting Code

```
    # Sampleing size
fs = 50000*2*np.pi
    #Define step size
steps = 1/fs

    #t for part 1
start = 0
stop = 1e-2
    #Define a range of t, with a stepsize of step
t = np.arange(start, stop, steps)

    # Make input signal array!
x_t = np.cos(2*np.pi*100*t) + np.cos(2*np.pi*3024*t) + np.sin(2*np.
    pi*5e4)

    # Move transfer function into z-domain
numZ, denZ = sig.bilinear(num,den,fs)

    # Pass signal through filter
y_t = sig.lfilter(numZ, denZ, x_t)

    # Do some ploting!
plt.figure(figsize=(10,7))
plt.figure(constrained_layout=True)
plt.subplot(2,1,1)
plt.title("x(t) Through Filter H(s)")
plt.ylabel("Filter Output y(t)")
plt.plot(t,y_t)
plt.grid()

plt.subplot(2,1,2)
plt.ylabel("Orginal Signal x(t)")
plt.xlabel("seconds")
plt.plot(t, x_t)
plt.grid()
```