ASSIGNMENT 7 - EE2703

EE17B047 - KOMMINENI ADITYA

March 19, 2019

1 Introduction

This assignment involves the use of **Sympy** module in Python. The module allows us to use symbols in various expressions and thereby enable us to represent filters and much complex systems.

Now, considering the low pass filter. On writing the equations in the AX = B form through nodal analysis, the equations we obtain are as follows :

$$\begin{vmatrix} 0 & 0 & 0 & -1/G \\ -1/(1+sR_2C_2) & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ -1/R_1 - 1/R_2 - sC_1 & 1/R_2 & 0 & sC_1 \end{vmatrix} \begin{vmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ 0 \\ -V_i(s)/R_1 \end{vmatrix}$$

Now, from the above matrix equation, we can find V_o by using the below code :

```
V = A.inv()*b
Vo = V[3]
```

Now, we must form the corresponding matrix for the high pass filter. The matrix equation is as below :

$$\begin{vmatrix} 0 & 0 & 0 & -1/G \\ -sR_2C_2/(1+sR_2C_2) & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ -1/R_1 - sC_2 - sC_1 & 1/R_2 & 0 & sC_1 \end{vmatrix} \begin{vmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ 0 \\ -V_i(s)sC_1 \end{vmatrix}$$

The V_o for the above matrix can be calculated using the same piece of vode as that for low pass filter.

2 Question 1

In order to convert the V_o expression into the signal.lti form, we need to extract the coefficients of the numerator and the denominator. Firstly, we must make sure that the V_o is in the simplest possible fraction form. We use the sympy.simplify() function to ensure the stated functionality.

```
Vo = sympy.simplify(Vo)
```

After performing the below code, the expression of V_o for the low pass and the high pass filters are as follows :

$$V_o(LowPass) = \frac{0.0001586}{(2.0e-14)s^2 + (4.414e-9)s + 0.0002}$$

$$V_o(HighPass) = \frac{1.586e-14s^2}{(2.0e-14)s^2 + (4.414e-9)s + 0.0002}$$

Then, we use the sympy.fraction() in order to obtain the numerator and denominator fractions. Following which, we convert the numerator and denominator polynomials into numpy arrays to enable floating point operations.

```
num,den = sy.fraction(Vs)
num = np.array(sy.Poly(num,s).all_coeffs(),dtype = np.complex64)
den = np.array(sy.Poly(den,s).all_coeffs(),dtype = np.complex64)
Vo_sig = sp.lti(num,den)
```

Therefore, we have the necessary lti equivalent for V_o. Thereby, we can plot the bode magnitude and phase functions. Also, by using the signal.impulse() and signal.step() functions, we can find the impulse and step responses for the low pass and high pass filters respectively.

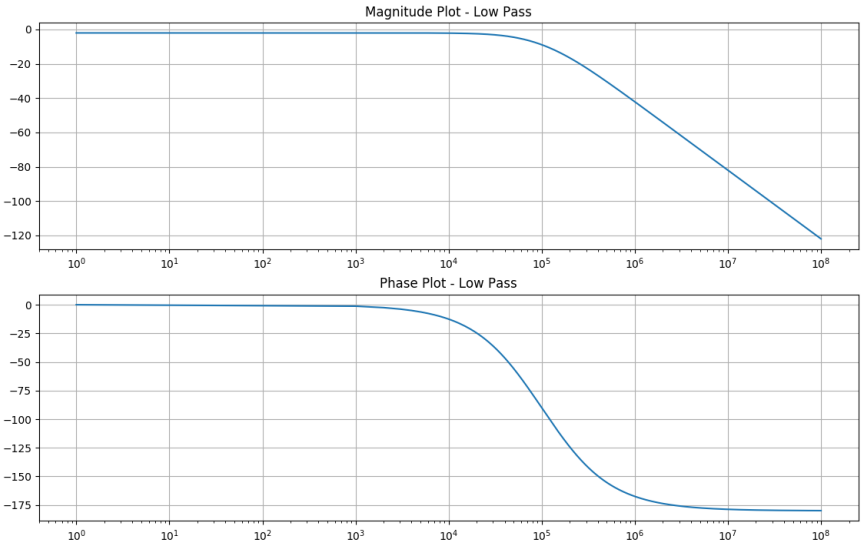

Figure 1: Bode plot for Low Pass Filter

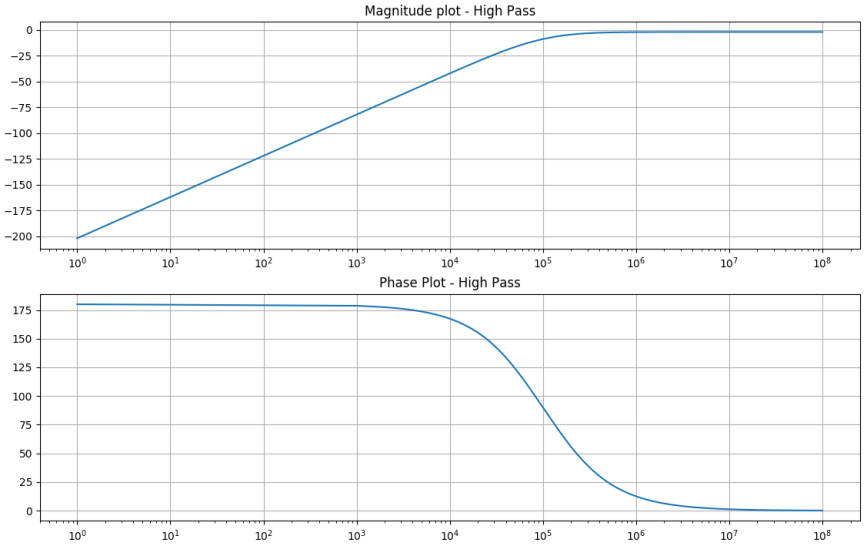

Figure 2: Bode plot for High Pass Filter

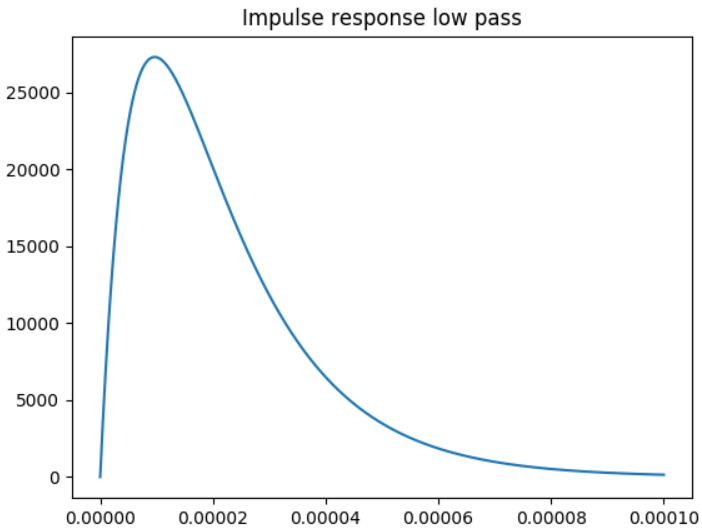

Figure 3: Impulse Response of Low Pass Filter

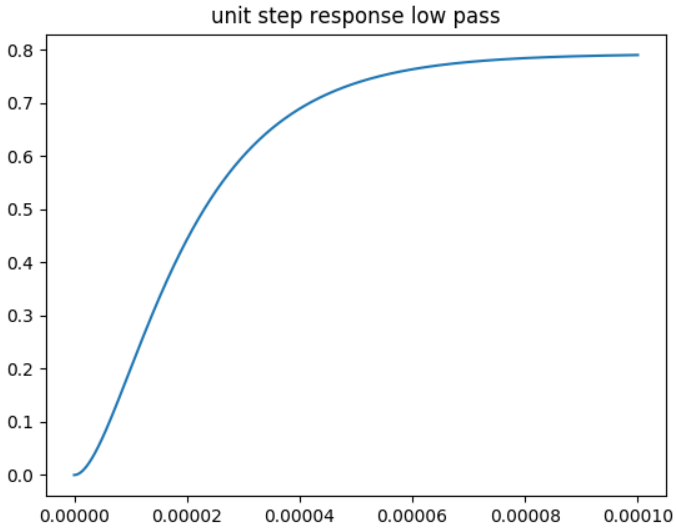

Figure 4: Unit Step Response of Low Pass Filter

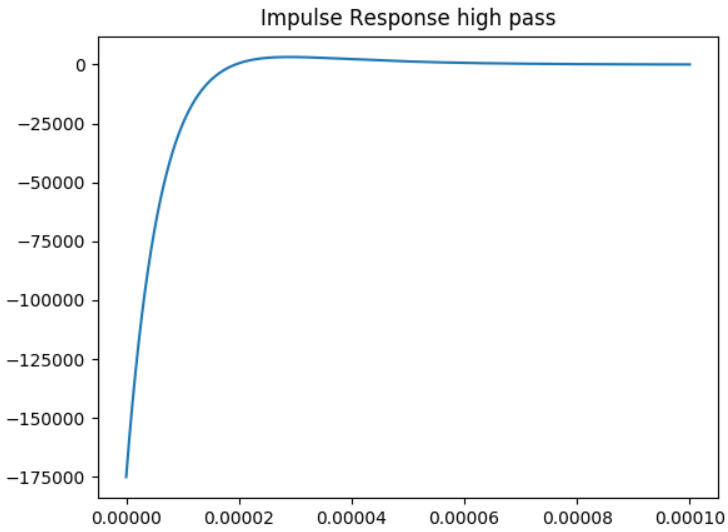

Figure 5: Impulse Response of High Pass Filter

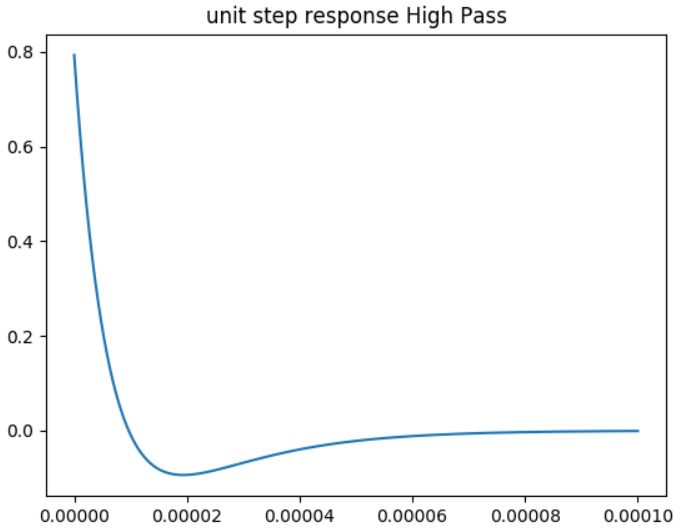

Figure 6: Step Response of High Pass Filter

3 Question 2

This question requires us to find the output of the low pass and high pass filters to the input :

$$V_i(t) = (cos(2e6\pi t) + sin(2000\pi t))u(t)$$

This can be done by taking the impulse response of the corresponding transfer function and convolving them with time domain input. Hence, for Lowpass filter, we get the following output.

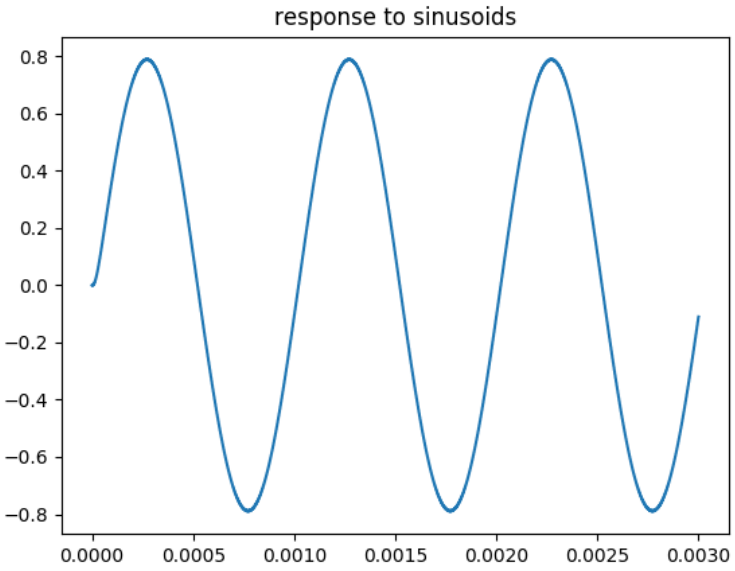

Figure 7: Response to Input from Lowpass Filter

From the above, we can infer that from the two frequencies i.e. 10^3 and 10^6, the lowpass filter has rejected the higher frequency therefore we get the sinusoid of the lower frequency as the output.

Conversely, for the highpass filter, the output consists of the sinusoid of the higher frequency as the filter masks the lower frequency. Therefore, the output for the highpass filter is :

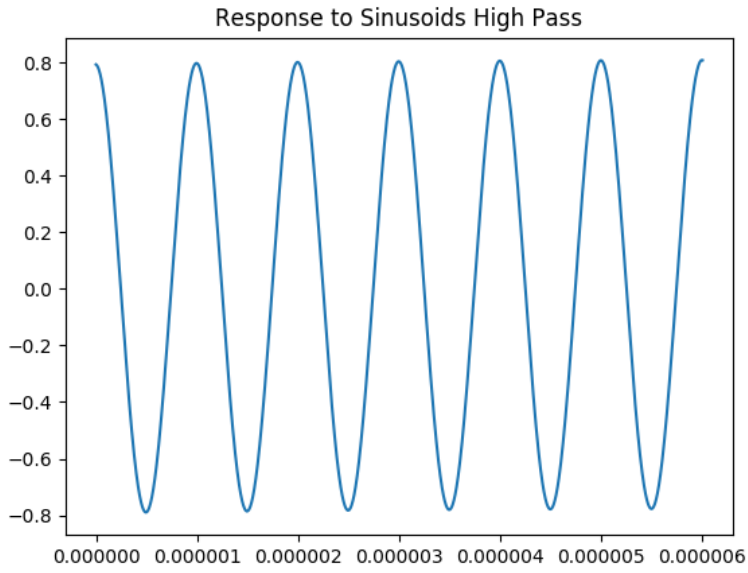

Figure 8: Response to the input from Highpass Filter

4 Question 3

Using the matrix function defined in the Introduction, we can define the function highpass which takes the values of the resistors, capacitors and input and gives the corresponding voltages at nodes as the outputs. The function is defined as :

```
def highpass(R1,R2,C1,C2,G,Vi):
    s=sy.symbols('s')
    A=sy.Matrix([[0,0,1,-1/G],[-(s*R2*C2)/(1+s*R2*C2),1,0,0], [0,-G,G,1]
        ,[-1/R1-s*C2-s*C1,s*C2,0,1/R1]])
    b=sy.Matrix([0,0,0,-Vi*s*C1])
    V = A.inv()*b
    return (A,b,V)
```

Therefore, we can use the above function to define our highpass filter.

5 Question 4

In order to find the response to a damped sinusoid, we should make sure that the sinusoid frequency is in the passband of the filter so that the sinusoid isn't clipped.

Here, for the lowpass filter, my input is :

$$V_i(t) = e^{-100t}cos(2000\pi t)$$

For this input, the corresponding output is :

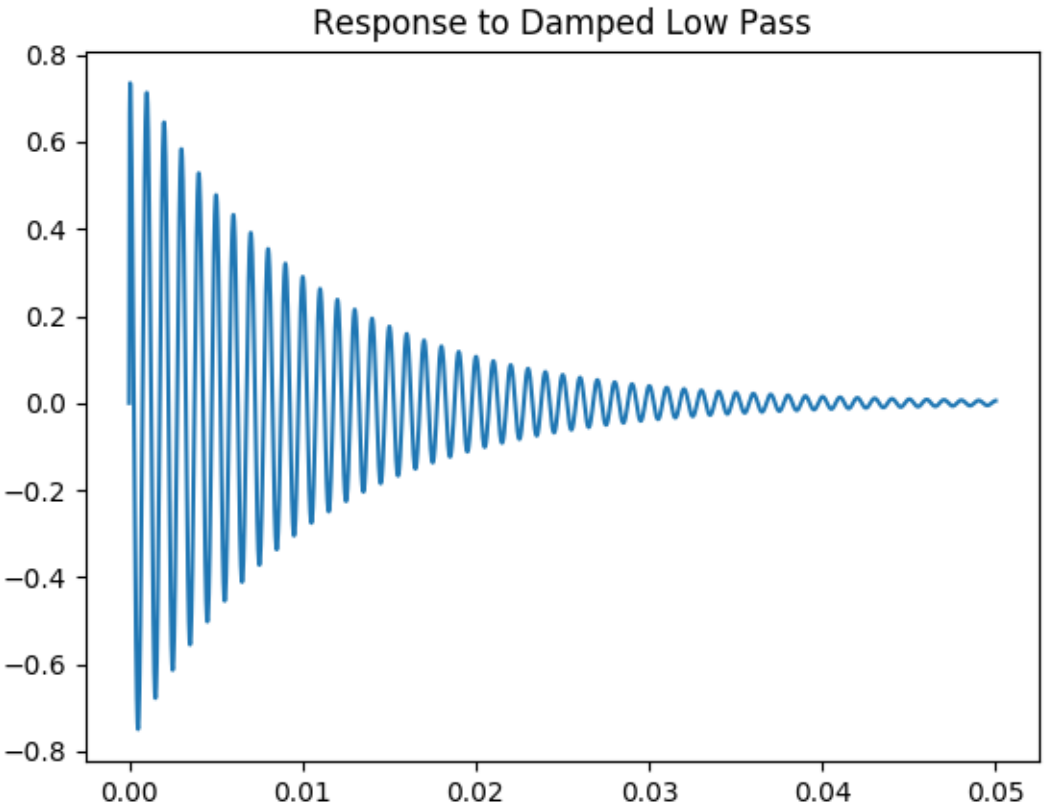

Figure 9: Respose of Lowpass Filter to Damped sinusoid

Now, for the highpass filter, the input is :

$$V_i(t) = e^{-10^5 t} cos(2 * 10^6 \pi t)$$

The input has sinusoid frequencies in the pass band which means that the output has damped sinusoids in it. The corresponding plot is :

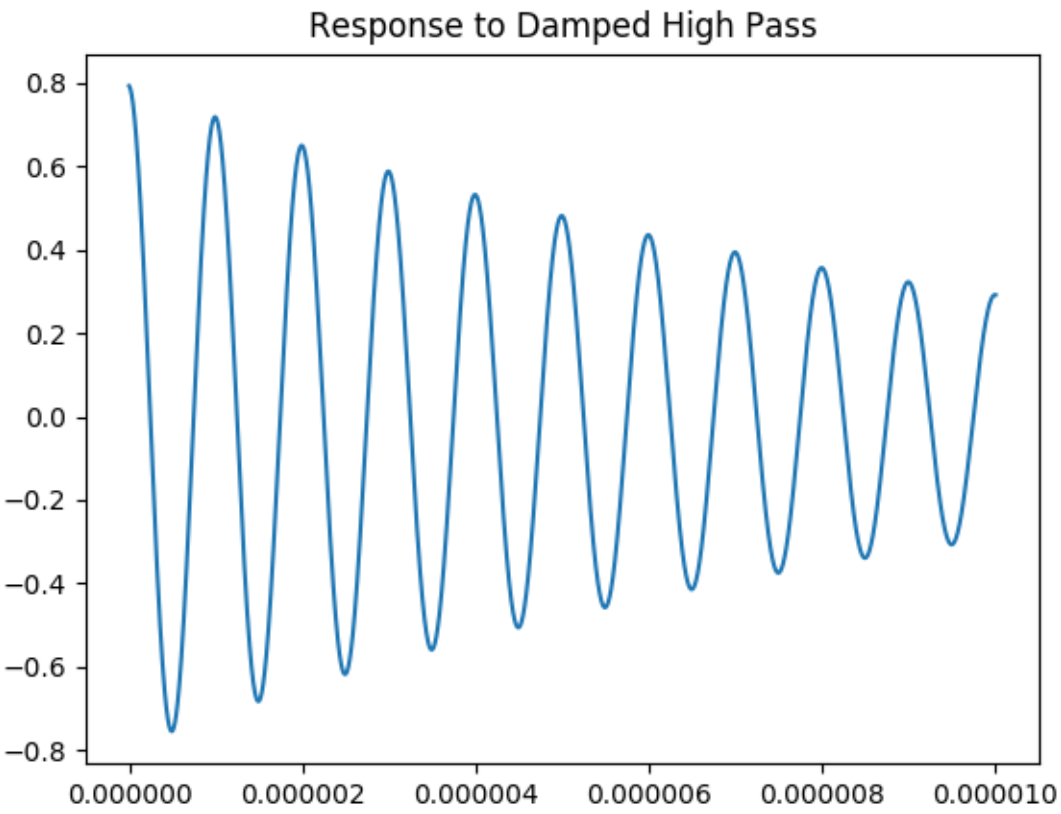

Figure 10: Response of the Highpass Filter to Damped sinusoid

6 Question 5

This question requires us to give the V_i to the function as 1/s i.e.(unit step). Then, we must obtain the step response in time domain. The corresponding code is :

```
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1/s)
Vo=V[3];Vo_signal = convert(Vo)
t,Vo_time = sp.impulse(Vo_signal,None,np.linspace(0,100e-6,20001))
plt.plot(t,Vo_time);plt.grid()
plt.title('step response using lowpass(1/s)')
plt.plot();plt.show()

A,b,V=highpass(10000,10000,1e-9,1e-9,1.586,1/s)
Vo=V[3];Vo_signal = convert(Vo)
t,Vo_time = sp.impulse(Vo_signal,None,np.linspace(0,100e-6,20001))
plt.plot(t,Vo_time);plt.grid()
plt.title('step response using Highpass(1/s)')
plt.plot();plt.show()
```

The graphs are identical to the step resposes which have been obtained using the signal.lsim function in Question2.

7 Inferences

- Firstly, we have obtained a method by which we can represent complec electrical components such as filters, opamps, etc.
- Using the Sympy module, we can find the transfer function in frequency domain.
- Although Sympy provides qith an effective way to find the symbolic expressions for transfer functions, it poses several difficulties in signal manipulaions as it isn't compatible with Scipy.