EE2703 : Applied Programming Lab
Assignment 7
Symbolic Algebra and Circuit Analysis

Aditya Nanda Kishore
EE20B062

April 8, 2022

Introduction

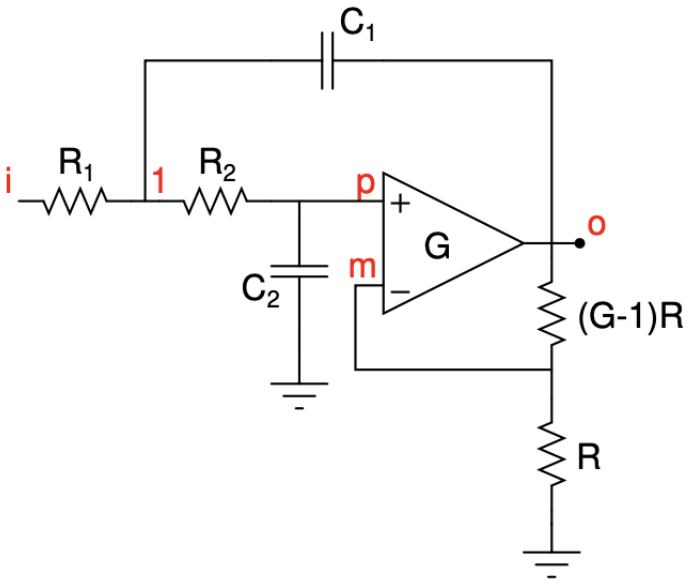
Python has a SymPy toolkit for computing symbolic algebra that will help us calculate and solve for transfer functions easily. We will be exploring that in this assignment.

Importing Necessary Libraries

```
import sympy as sym
import scipy.signal as sp
import numpy as np
import matplotlib.pyplot as plt
```

Low Pass Butterworth Filter

So, Now Let's consider a Low Pass Butter-worth Filter which is something like this



where $G = 1.586$ and $R_1 = R_2 = 10k\Omega$ and $C_1 = C_2 = 10pF$.The circuit equations for this circuit are

$$V_m = \frac{V_o}{G} \tag{1}$$

$$V_p = V_1 \frac{1}{1 + j\omega R_2 C_2} \tag{2}$$

$$V_o = G(V_p - V_m) \tag{3}$$

$$\frac{V_i - V_1}{R_1} + \frac{V_p - V_1}{R_2} + j\omega C_1(V_o - V_1) = 0 \tag{4}$$

So the Matrix form of this set of linear equations is

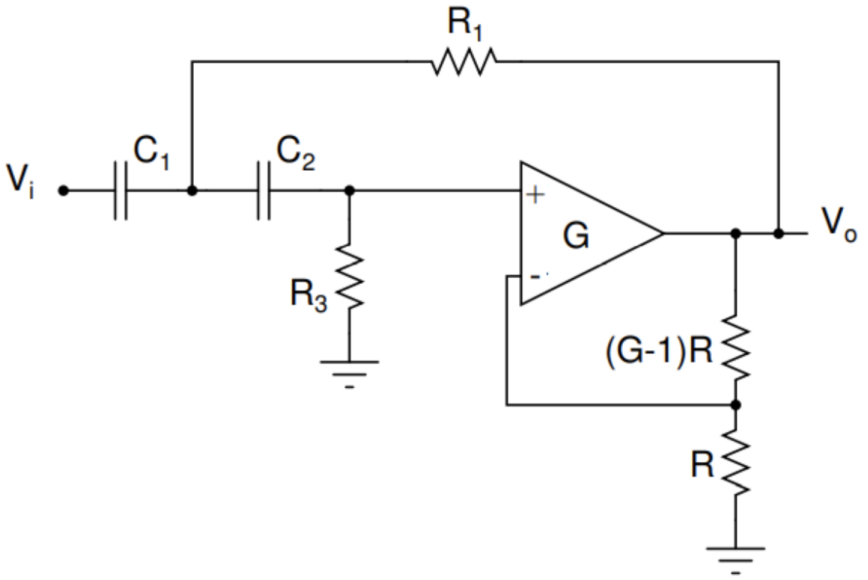
$$\begin{pmatrix} 0 & 0 & 1 & -\frac{1}{G} \\ -\frac{1}{1+sR_2C_2} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ -\frac{1}{R_1}-\frac{1}{R_2}-sC_1 & \frac{1}{R_2} & 0 & sC_1 \end{pmatrix} \begin{pmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ V_i(s)/R_1 \end{pmatrix}$$

We can now solve this matrix using **A.inv()*B** command.Code for this filter is

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

High Pass Butterworth Filter

Similarly, We are given a highpass butterworth filter too.



Equations for this filter are

$$V_m = \frac{V_o}{G} \tag{5}$$

$$V_p = V_1 \frac{j\omega C_2 R_3}{1 + j\omega R_3 C_2} \tag{6}$$

$$V_o = G(V_p - V_m) \tag{7}$$

$$\frac{-V_o}{R_1} - j\omega C_2 V_p - j\omega C_1 V_i + V_1(j\omega C_2 + j\omega C_1 + \frac{1}{R_1}) = 0 \tag{8}$$

So, The matrix form of this set of equations is

$$\begin{pmatrix} 0 & 0 & 1 & -\frac{1}{G} \\ -\frac{-sR_3C_2}{1+sR_3C_2} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ -1-(sR_1C_1)-(sR_3C_2)) & sC_2R_1 & 0 & 1 \end{pmatrix} \begin{pmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -V_i(s)sR_1C_1 \end{pmatrix}$$

Just as above,we can now solve this matrix using **A.inv()*B** command. Code for this filter is

```
def highpass(R1,R3,C1,C2,G,Vi):
    s=sym.symbols('s')
    A = sym.Matrix([[0,0,-1,1/G],
    [0,G,-G,-1],
    [s*C1+s*C2+1/R1,-s*C2,0,-1/R1],
    [-s*C2*R3/(1+s*C2*R3),1,0,0]])
    b = sym.Matrix([0,0,s*C1*Vi,0])
    V = A.inv()*b
    return (A,b,V)
```

Converting Sympy function to LTI Transfer function

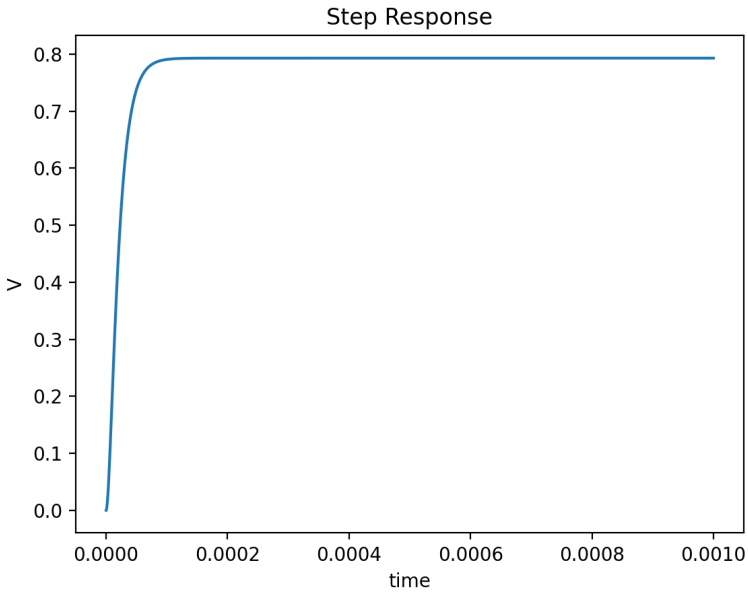
In order to use **scipy.signal** commands we need list of function's numerator's coefficients and denominator's as well.

```
def sympyToH(V):
    n,d=sym.fraction(V)
    n,d=sym.Poly(n,s),sym.Poly(d,s)
    num,den = n.all_coeffs(), d.all_coeffs()
    num,den = [float(f) for f in num], [float(f) for f in den]
    H = sp.lti(num,den)
    return H
```

Step Response of Butterworth Low Pass Filter

We will use the previous function and get the transfer function and use it to execute **lsim** command and the second argument here would be **ones** array so that we will get a step response

```
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)
Vo_Q1 = V[3]
H_low = sympyToH(Vo_Q1)
t = np.linspace(0, 1e-3, 1001)
t,x,svec = sp.lsim(H_low , np.ones(1001),t)
plt.plot(t,x)
plt.title('Step Response')
plt.xlabel('time')
plt.ylabel('V')
plt.show()
```



Action of Butterworth Low Pass Filter

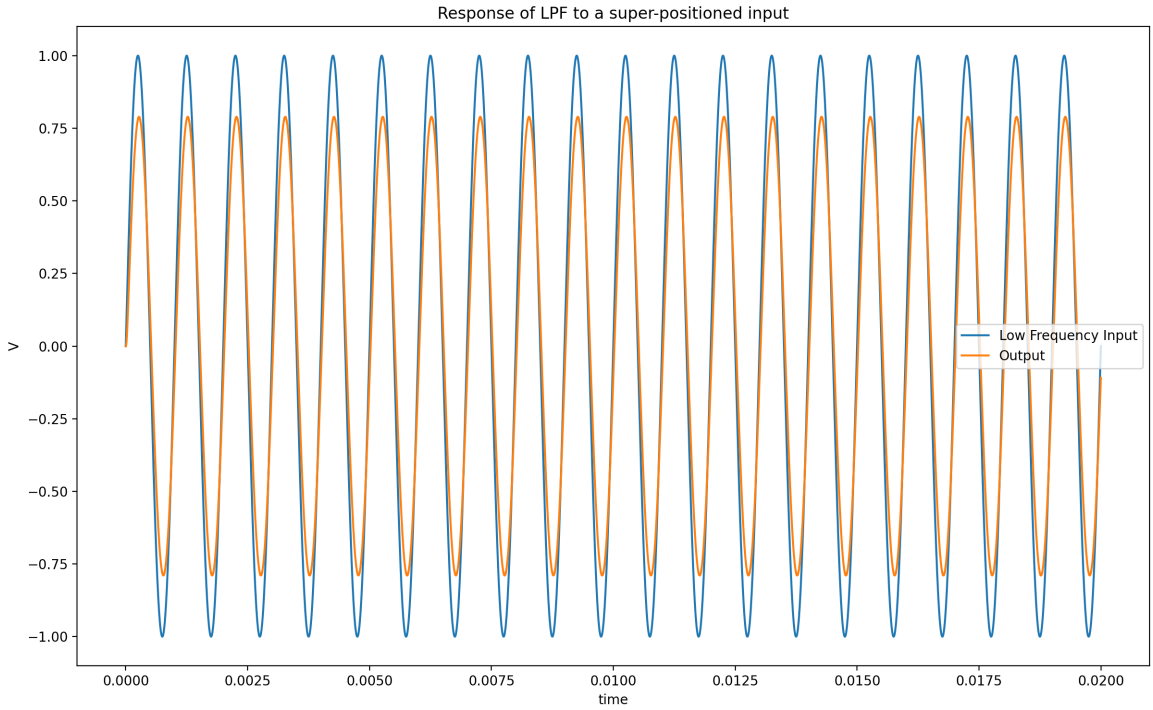
Now Input is given as

$$V_i = sin(2000\pi t) + cos(2.10^6\pi t)u(t) \tag{9}$$

So using the same transfer function but with input in **lsim** command as the above equation, we get an output where the higher frequency got filtered.I have plotted the lower frequency input to depict this.

```
t = np.linspace(0,2e-2,100001)
t,x,svec=sp.lsim(H_low, np.sin(2000*np.pi*t)+np.cos(2e6*np.pi*t),t)
plt.plot(t,np.sin(2000*np.pi*t))
plt.plot(t,x)
plt.title('Response of LPF to a super-positioned input')
```

```
plt.xlabel('time')
plt.ylabel('V')
plt.legend(['Low Frequency Input','Output'])
plt.show()
plt.plot(t,x)
plt.title('Response of LPF to a super-positioned input')
plt.xlabel('time')
plt.ylabel('V')
plt.legend(['Output'])
plt.show()
```



Magnitude of frequency response of Butterworth High Pass Filter

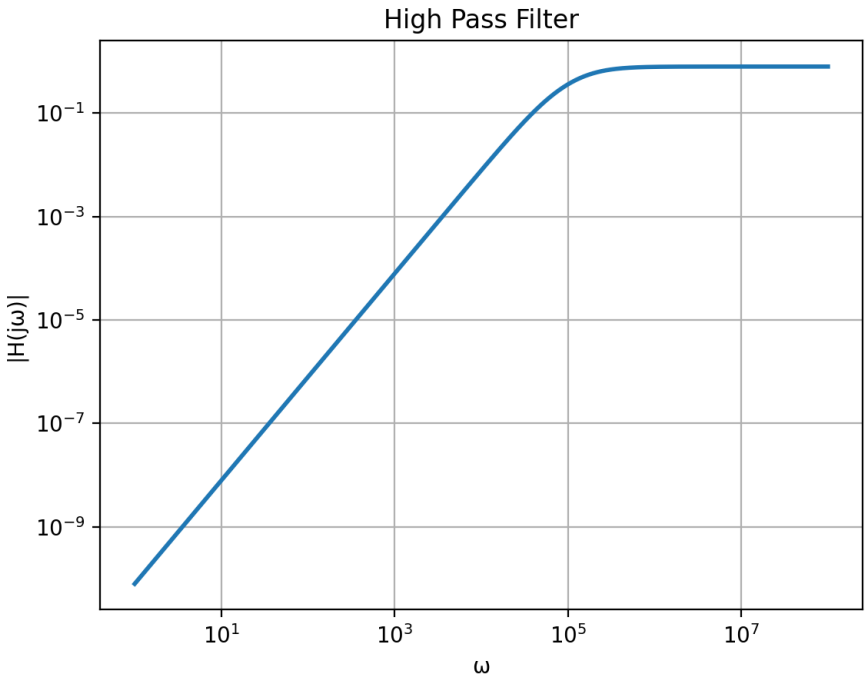
We use lambdify command as shown in the assignment to draw the bode plot instead of scipy commands.Using the above high pass filter code we would get something like this

```
A,b,Vh=highpass(10000,10000,1e-9,1e-9,1.586,1)
H_high=Vh[3]

ww=np.logspace(0,8,801)
```

```
ss=1j*ww
hf=sym.lambdify(s,H_high,"numpy")
v=hf(ss)
plt.loglog(ww,abs(v),lw=2)
plt.grid(True)
plt.xlabel('')
plt.ylabel('|H(j)|')
plt.title('High Pass Filter')
plt.show()


H_high = sympyToH(H_high)
```

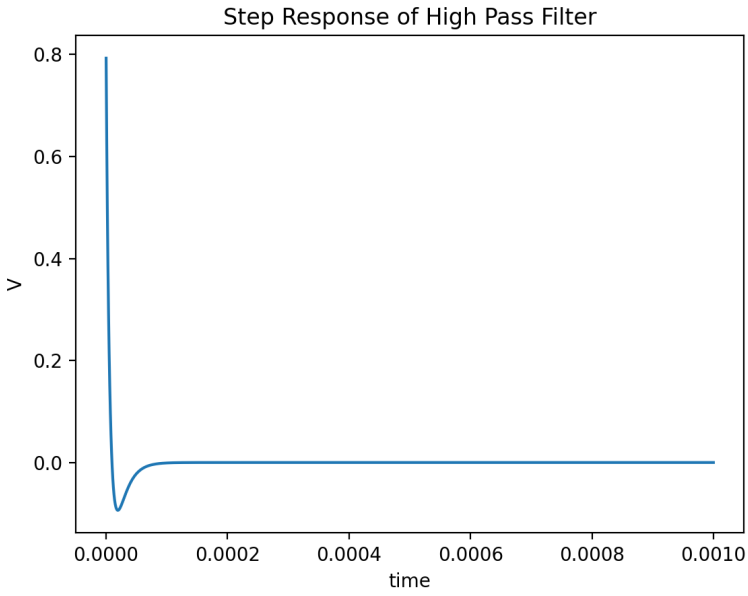


Step Response of Butterworth High Pass Filter

We will use the **sympyToH** function and get the transfer function and use it to execute **lsim** command and the second argument here would be **ones** array so that we will get a step response.

```
t=np.linspace(0,1e-3,1001)
t,x,svec=sp.lsim(H_high,np.ones(1001),t)
plt.plot(t,x)
plt.title('Step Response of High Pass Filter')
```

```
plt.xlabel('time')
plt.ylabel('V')
plt.show()
```



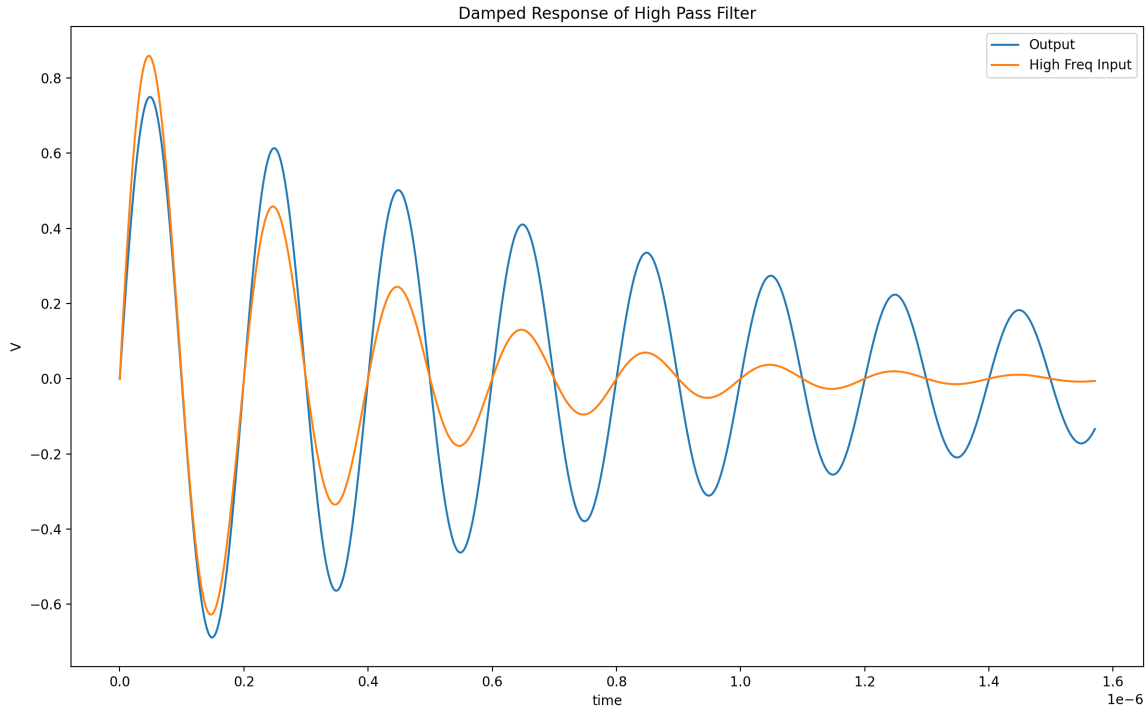
Butterworth High Pass Filter's response to a Damped sinusoid

Now for this I have taken an input of

$$V_i = sin(10^7 \pi t)e^{-10^6 t} + cos(10^3 \pi t)e^{-10^6 t} \tag{10}$$

to show the high frequency filtered. As we can see in the output, it's frequency is matching with the output's frequency

```
t=np.linspace(0,5e-7*np.pi,5001)
t,x,svec=sp.lsim(H_high,np.sin(1e7*np.pi*t)*np.exp(-1e6*t) + np.sin(1e3*np.pi*t)*np.exp(
plt.plot(t,x)
plt.plot(t,np.sin(1e7*np.pi*t)*np.exp(-1e6*np.pi*t))
plt.title('Damped Response of High Pass Filter')
plt.xlabel('time')
plt.ylabel('V')
plt.legend(['Output', 'High Freq Input'])
plt.show()
```



Conclusions

Sympy can be used to solve complex nodal equations using symbols quite easily. Hence, we can see that when we use **scipy.signal** toolbox together with it, circuit analysis is easier. It helped me understand the two Butterworth filters better.