# Assignment 7: Analysis of Circuits using Symbolic Algebra

Siddharth D P
EE18B072

March 15, 2020

## 1 Abstract

The goal of this assignment is the following.

- To incorporate Symbolic Algebra in Python.
- Analyze the given circuits using Laplace Transforms.
- To understand how a Lowpass filter and Highpass filter differ in their transfer functions and responses to different inputs.

## 2 Introduction

This assignment explores the use of **Sympy** module in Python. The module has provisions for using symbols in various expressions instead of actual valued variables. Considering the low pass filter who's figure is given, writing the nodal matrix in the form AX = B, we get:

$$\begin{bmatrix} 0 & 0 & 0 & -1/G \\ -1/1+sR_2C_2 & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ -1/R_1-1/R_2-sC_1 & 1/R_2 & 0 & sC_1 \end{bmatrix} \begin{bmatrix} \mathrm{V}_1 \\ \mathrm{V}_p \\ \mathrm{V}_m \\ \mathrm{V}_o(\mathrm{s}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -\mathrm{V}_i(\mathrm{s})/R_1 \end{bmatrix}$$

Now, from the above matrix equation, we can find $V_o$ by using

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

Now, forming the corresponding matrix for the high pass filter, we get the matrix equation as

$$\begin{bmatrix} 0 & 0 & 0 & -1/G \\ -sR_2C_2/(1+s\mathrm{R}_2C_2 & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ -1/\mathrm{R}_1\text{-}1/R_2\text{-}s\mathrm{C}_1 & s\mathrm{C}_2 & 0 & 1/\mathrm{R}_1 \end{bmatrix} \begin{bmatrix} \mathrm{V}_1 \\ \mathrm{V}_p \\ \mathrm{V}_m \\ \mathrm{V}_o(\mathrm{s}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -\mathrm{V}_i(\mathrm{s})/R_1 \end{bmatrix}$$

$V_o$ for the above matrix can be calculated as before.

# 3 Assignment

## 3.1 Importing the standard libraries

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

## 3.2 Question 1

First we have to convert the **sympy** expression to a polynomial expression after simplifying in order to process it the **scipy.signal()** toolbox. To make sure that the rational function is in the simplest possible form, we use the expression

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

After executing the below code, the expression for Vo for the low pass and the high pass filters are as follows :

$$\text{Vo(LowPass)} = \frac{0.0001586}{(2.0e14)s^2+(4.414e9)s+0.0002}$$
$$\text{Vo(HighPass)} = \frac{1.586e14s^2}{(2.0e14)s^2+(4.414e9)s+0.0002}$$

We then use the *sympy.fraction* to obtain the numerator and denominator fractions. Following which, we convert the numerator and denominator polynomials into numpy arrays to enable floating point operations.

```
def convert(V_s):
    V_s=simplify(V_s)
    n,d =fraction(V_s)
    n=poly(n,s).all_coeffs()
    d=poly(d,s).all_coeffs()
    n,d = [float(i) for i in n], [float(i) for i in d]
    Vo_sig = sp.lti(n,d)
    return Vo_sig
```

Therefore, we have the necessary LTI equivalent for Vo. After defining the lowpass filter, signals can now be processed on this and Bode plots can be plotted.

```
s=symbols('s')
#defining the lowpass filter function
def lowpass(R1,R2,C1,C2,G,Vi):
    A=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
    b=Matrix([0,0,0,-Vi/R1])
    V=A.inv()*b
```

```
    return (A,b,V)

#defining the highpass filter function
def highpass(R1,R2,C1,C2,G,Vi):
    s=symbols('s')
    A1=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
    b1=Matrix([0,0,0,-Vi*s*C1])
    V1=A1.inv()*b1
    return (A1,b1,V1)
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)
Vo=V[3]
Ah,bh,Vh=highpass(10000,10000,1e-9,1e-9,1.586,1)
Voh=Vh[3]

#Bode plots of LPF
plt.figure(1)
ww=np.logspace(0,8,801)
ss=1j*ww
hf=lambdify(s,Vo,'numpy')
v=hf(ss)
plt.subplot(2,1,1)
plt.loglog(ww,abs(v),lw=2)
plt.xlabel('ww')
plt.ylabel('f(ss)')
plt.grid()
plt.title('Bode plot of LPF step response')
plt.subplot(2,1,2)
hhf=lambdify(s,Voh,'numpy')
vh=hhf(ss)
plt.loglog(ww,abs(vh),lw=2)
plt.xlabel('ww')
plt.ylabel('f(ss)')
plt.title('Bode plot of HPF step response')
plt.tight_layout()
plt.grid()

#step response of LPF
plt.figure(2)
Vo_t = convert(Vo)
t,step_resp = sp.step(Vo_t,None,np.linspace(1e-5,1e-3,2001),None)
plt.plot(t,step_resp,'b-')
plt.xlabel('Time(s)')
plt.ylabel('Vo(t)')
plt.title('Unit Step response for the Lowpass filter')
```

```
plt.grid(True)

#step response of HPF
plt.figure(3)
Vo_th = convert(Voh)
t,step_resph = sp.step(Vo_th,None,np.linspace(1e-5,1e-3,2001),None)
plt.plot(t,step_resph,'b-')
plt.xlabel('Time(s)')
plt.ylabel('Vo(t)')
plt.title('Unit Step response for the Lowpass filter')
plt.grid(True)
```

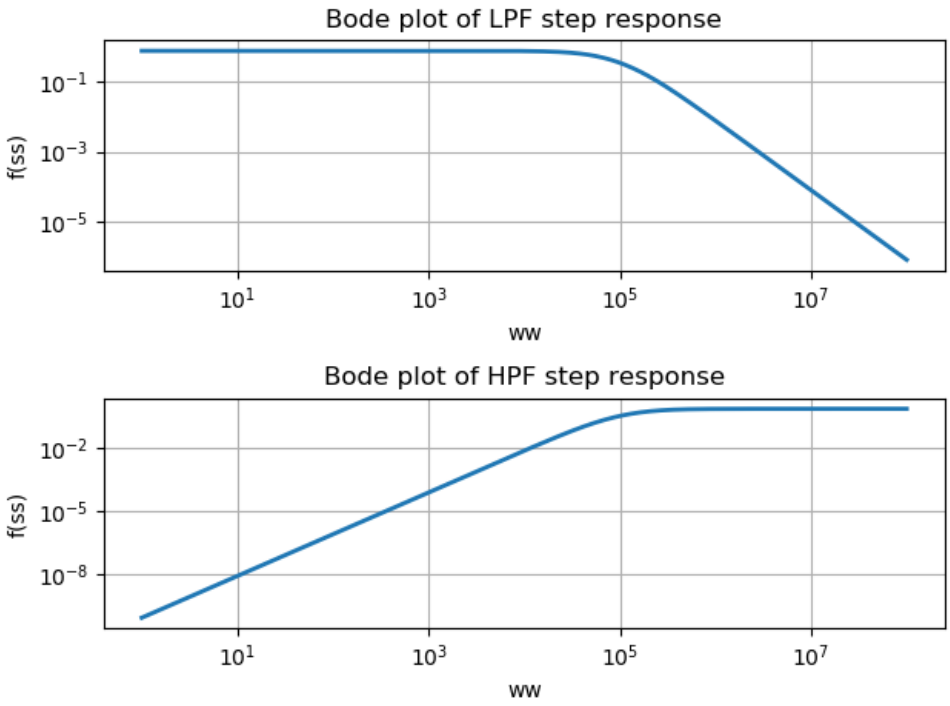


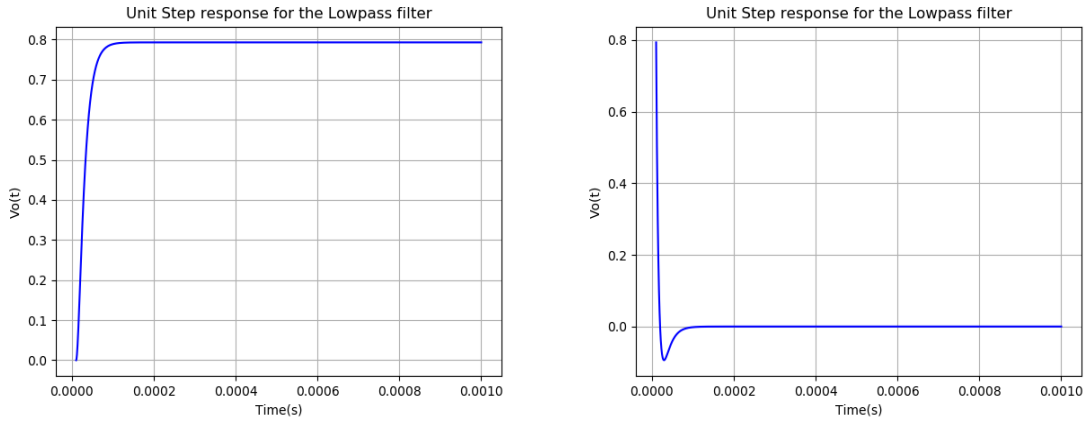

Figure 1: Bode Plots for both the Lowpass and Highpass filters

### 3.3 Question 2

Given the input as

$$V_i(t) = (sin(2000\pi t) + cos(2 \times 10^6 \pi t))u(t)$$

we calculate the output voltage by using the **scipy.signal.lsim()** function by convolving the impulse response with the given time domain input.




(a) Lowpass filter

(b) Highpass filter

Figure 2: Unit step responses of both the filters

```
#response to sum of sinusoids LPF
plt.figure(4)
Vo_t = convert(Vo)
t=np.linspace(0,1e-2,2001)
V_inp=np.sin(2000*np.pi*t)+np.cos(2e6*np.pi*t)
t,resp,svec = sp.lsim(Vo_t,V_inp,t)
plt.plot(t,resp,'b-')
plt.xlabel('Time(s)')
plt.ylabel('Vo(t)')
plt.title('Response for the sum of sinusoids for the Lowpass filter')
plt.grid(True)

#response to the sum of sinusoids HPF
plt.figure(5)
Vo_th = convert(Voh)
th = np.linspace(0,1e-5,2001)
V_inp=np.sin(2000*np.pi*th)+np.cos(2e6*np.pi*th)
th,resph,svec = sp.lsim(Vo_th,V_inp,th)
plt.plot(th,resph,'b-')
plt.xlabel('Time(s)')
plt.ylabel('Vo(t)')
plt.title('Response for the sum of sinusoids for the Highpass filter')
plt.grid(True)
```

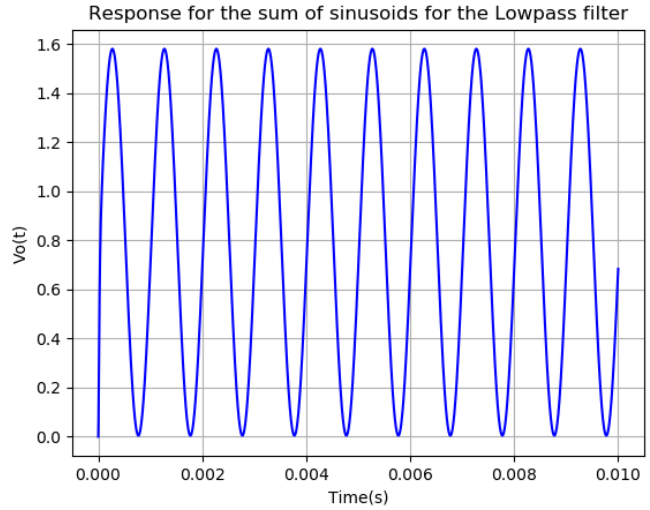



Figure 3: Response for a Lowpass filter

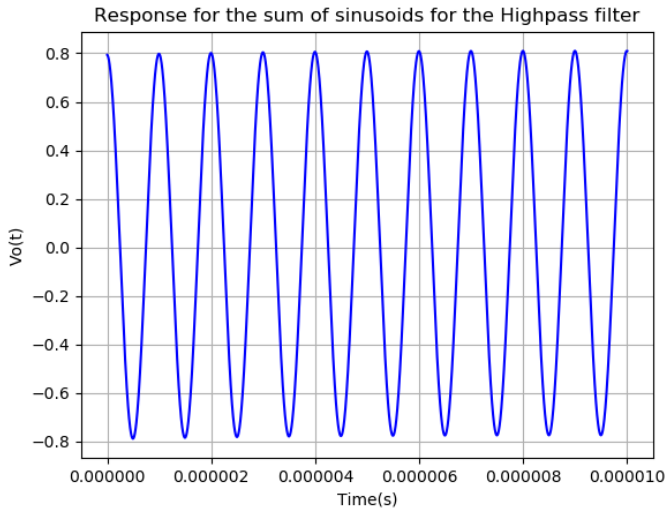



Figure 4: Response for a Highpass filter

From the above, we can infer that from the two frequencies $10^3$ and $10^6$, the lowpass filter rejects the higher frequency and hence, only the sinusoid of the lower frequency is present in the output. Conversely, for the highpass filter, the output consists of the sinusoid of the higher frequency as the filter suppresses the lower frequency. For the graphs to look similar, the time scale for the lowpass filter is in the range of 0.01s while the highpass filter is in the range of 0.00001s.

### 3.4 Question 3

Using the matrix function defined in the Introduction, we go on to define the highpass filter function by replacing all $1/R_1$ with $sC_1$ and $1/R_2$ with $sC_2$ to give the corresponding voltages at nodes as the outputs. Everything else in the nodal matrix remains the same.

```
def highpass(R1,R2,C1,C2,G,Vi):
    s=symbols('s')
    A1=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]])
    b1=Matrix([0,0,0,-Vi*s*C1])
    V1=A1.inv()*b1
    return (A1,b1,V1)
```

### 3.5 Question 4

We have to ensure that the response to a damped sinusoid is in the pass-band of the filter so that it isn't clipped. For both the lowpass and the highpass filters, a low frequency and high frequency input taken is taken. The functions defined are as follows :

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

$$V_i(t) = cos(2 \times 10^5 \pi t)e^{-100t}$$

```
#response to a damped sinusoid LPF
plt.figure(6)
Vo_t = convert(Vo)
t = np.linspace(0,1e-2,2001)
V_inp=np.cos(2*1e3*np.pi*t)*np.exp(-100*t) # cos(2*10^3*pi*t)e0.05t
t,resp,svec = sp.lsim(Vo_t,V_inp,t)
plt.plot(t,resp,label='Frequency = 1kHz')
thf = np.linspace(0,1e-2,2001)
V_inphf=np.cos(2*1e5*np.pi*thf)*np.exp(-100*thf) # cos(2*10^5*pi*t)e0.05t
thf,resphf,svec = sp.lsim(Vo_t,V_inphf,thf)
plt.plot(t,resphf,label='Frequency = 1MHz')
plt.xlabel('Time(s)')
plt.ylabel('Vo(t)')
plt.legend()
plt.title('Response to a damped sinusoid for the Lowpass filter')
plt.grid(True)

#response to a damped sinusoid HPF
plt.figure(7)
```

```
Vo_th = convert(Voh)
th = np.linspace(0,0.02,4001)
V_inp=np.cos(2*1e3*np.pi*th)*np.exp(-100*th) # cos(2*10^3*pi*t)e0.05t
th,resph,svec = sp.lsim(Vo_th,V_inp,th)
plt.plot(th,resph,label='Frequency = 1kHz')
thhf = np.linspace(0,1e-3,4001)
V_inphhf=np.cos(2*1e5*np.pi*thhf)*np.exp(-100*thhf) # cos(2*10^5*pi*t)e0.05t
thhf,resphhf,svec = sp.lsim(Vo_th,V_inphhf,thhf)
plt.plot(thhf,resphhf,label='Frequency = 1MHz')
plt.xlabel('Time(s)')
plt.ylabel('Vo(t)')
plt.legend()
plt.title('Response to a damped sinusoid for the Highpass filter')
plt.grid(True)
```

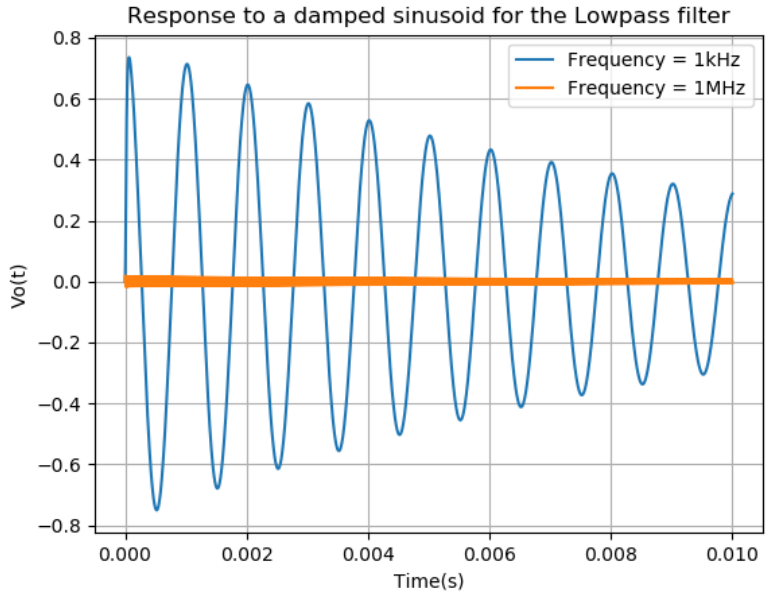



Figure 5: *Response of Lowpass filter to a damped sinusoid of different frequencies*

### 3.6 Question 5

The graph is identical to the step response as obtained in Question 2 since the only difference between that and this method is the usage of 1/s in the s-domain instead of convolving with 1 in the time domain using the *signal.lsim* function in Question 2.

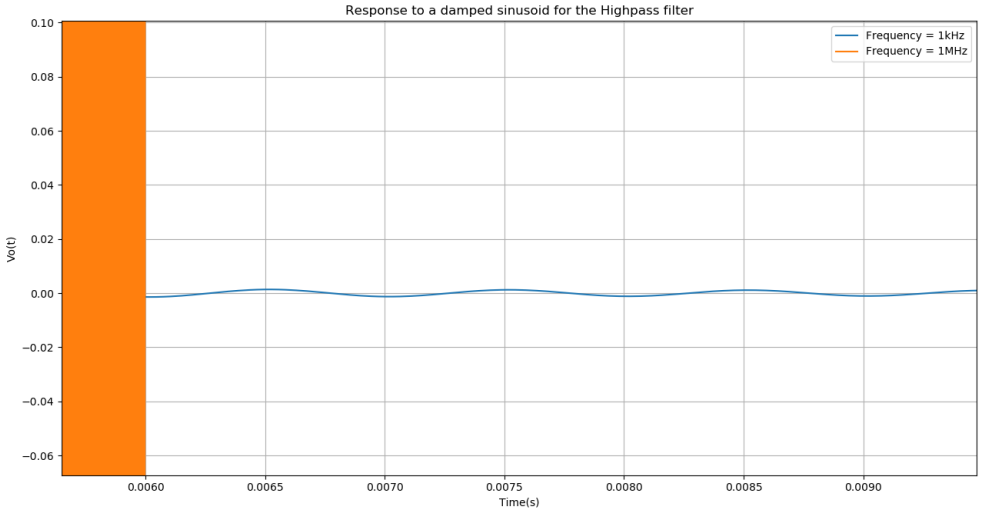



Figure 6: *A highly zoomed version to show the Response of Highpass filter to both a low and high frequencies for a damped sinusoid*

```
plt.figure(8)
Ah,bh,Vh=highpass(10000,10000,1e-9,1e-9,1.586,1/s)
Voh=Vh[3]
Vo_th = convert(Voh)
th,step_resph = sp.impulse(Vo_th,None,np.linspace(1e-5,1e-3,2001),None)
plt.plot(th,step_resph,'b-')
plt.xlabel('Time(s)')
plt.ylabel('Vo(t)')
plt.title('Unit Step response for the Highpass filter using V_i=1/s')
plt.grid(True)
plt.show()
```

# 4 Conclusion

- We obtained a method by which we can represent electrical components such as filters, opamps etc in Python.
- We analyzed the lowpass and highpass Butterworth filters.
- We saw a how symbolic algebra can be a powerful tool for computing various signals to the system. We must remember that though **Sympy**

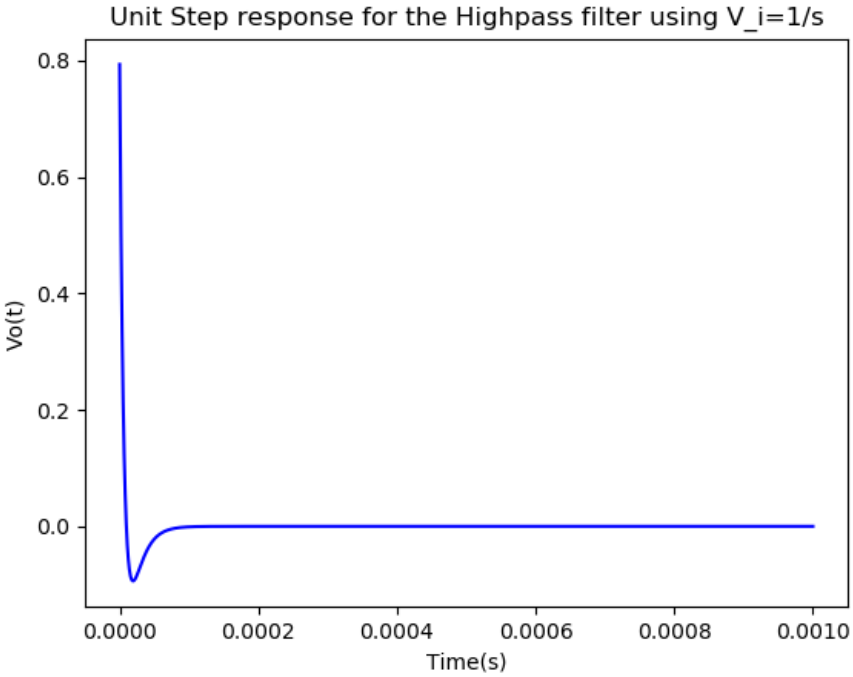



Figure 7: *Step Response of Highpass filtern*

comes with enormous computational abilities, it must be converted to its **Scipy** equivalent before any signal processing can be done on it.