Assignment 7 :Circuit analysis using sympy

Gudivada Harshad Kumar
EE20B038

April 8, 2022

Abstract

The main aim of the assignment 7 is

- Analyze the output for different kind of filters
- To understand the working of *sympy* in Python
- Plotting the output response of filter for given input

Let us define both low pass as well as high pass filter matrices

Low Pass Filter

The matrix form for the low pass filter is

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

The python code snippet that declares the low pass filter is given by

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

The matrix form for the high pass filter is

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

The python code snippet that declares the high pass filter is given by

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

We now define a function to get numpy array from polynomials. The following python codemsnippet does it.

```
def get_Poly(nr,dr):
    isFloat = False
    try:
        nr = Poly(nr).all_coeffs()
    except GeneratorsNeeded:
        nr = nr
        isFloat = True
    dr = Poly(dr).all_coeffs()
    dr2 = []
    nr2 = []
    for i in dr:
        dr2.append(float(i))
    dr2 = np.array(den2)
    if (isFloat):
        nr2 = nr
    else:
        for i in nr:
            nr2.append(float(i))
        nr2 = np.array(nr2)
    return nr2,dr2
```

We now plot the magnitude and Phase plots of low pass filter. The following python code snippet helps to do it.

```
A_1,b_1,V_1 = lowpass_filter(10000,10000,1e-9,1e-9,1.586,1)
Vo1=V_1[3]
w=mp.logspace(0,8,801)
ss=1j*w
hf=lambdify(s,Vo1,'numpy')
v=(hf(ss))
phi = np.angle(v)
mp.figure(num=0,figsize = (7,7))
mp.subplot(2,1,1)
mp.loglog(w,abs(v),lw=2)
mp.title("Magnitude Plot")
mp.xlabel(r'$\omega\rightarrow$',loc ='left')
mp.ylabel('Magnitude in log scale')
mp.grid(True)

mp.subplot(2,1,2)
mp.semilogx(w,phi,lw=2)
mp.title("Phase plot")
mp.xlabel("Frequency in log scale",loc ='left')
mp.ylabel("Phase")
mp.grid(True)
mp.show()
```

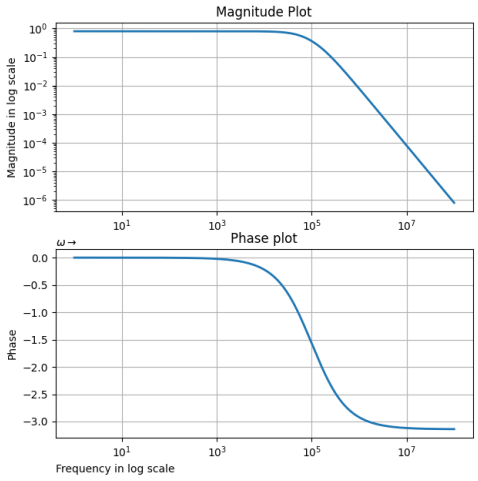

Figure 1: Bode plots

Question 1

Now we obtain the magnitude plot for step input to low pass filter using the following python code snippet

```
Vi = 1/s
A_1,b_1,V_1 = lowpass_filter(10000,10000,1e-9,1e-9,1.586,Vi)
Y=V_1[3]
w=mp.logspace(0,8,801)
ss=1j*w
hf=lambdify(s,Y,'numpy')
v=(hf(ss))
phi = np.angle(v)
mp.figure(num=1,figsize = (7,7))
mp.loglog(w,abs(v),lw=2)
mp.title("Magnitude Plot")
mp.xlabel(r'$\omega\rightarrow$',loc ='left')
mp.ylabel('Magnitude in log scale')
mp.grid(True)
mp.show()
```

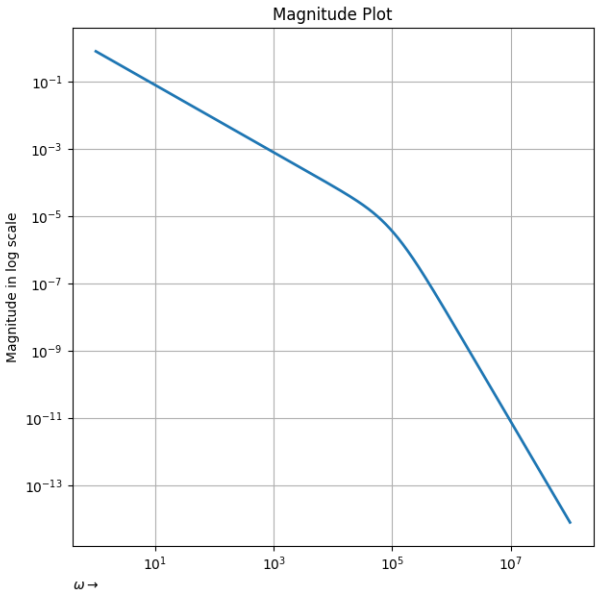

Figure 2: Magnitude plot

Now we obtain the time domain output plot for step input to low pass filter using the following python code snippet

```
simpY = simplify(Y)
num = fraction(simpY)[0]
den = fraction(simpY)[1]
num2,den2 = get_Poly(num,den)
num2 = np.poly1d(num2)
den2 = np.poly1d(den2)
Y = sp.lti(num2,den2)
t = np.linspace(0.0,4e-3,10001)
t,y=sp.impulse(Y,None,t)
mp.figure(num=2,figsize = (7,7))
mp.plot(t,y)
mp.title('Time domain output to step input')
mp.xlabel('Time')
mp.ylabel('y(t)')
mp.ylim(-1,1)
mp.grid(True)
mp.show()
```

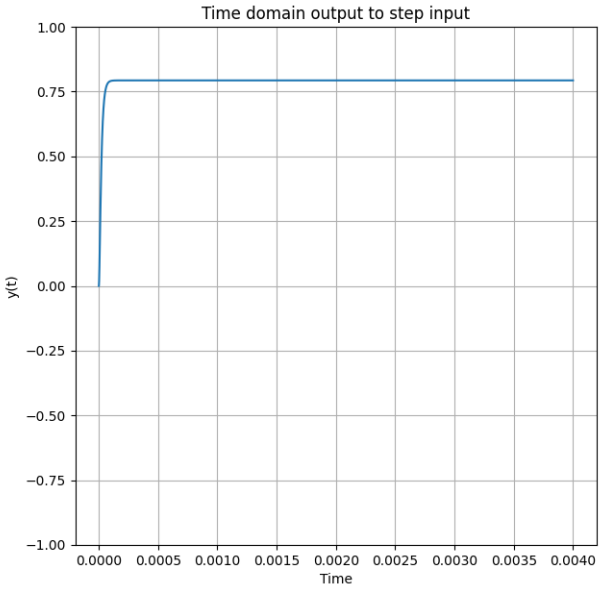

Figure 3: Output plot for step input

Question 2

Now for the given input of sum of two sinusoidal functions we plot the output throught low pass filter

$$V_i(t) = (\ \sin(2000\pi t) + \cos(2 * 10^6 \pi t)\ ) u_o(t)\ Volts$$

We use the following python code snippet to get the output for given input

```
t_2 = np.linspace(0.0,4e-3,100001)
x_2 = np.sin(2000*np.pi*t_2) + np.cos(2*(10**6)*np.pi*t_2)
simpH = simplify(Vo1)
num = fraction(simpH)[0]
den = fraction(simpH)[1]
num2,den2 = get_Poly(num,den)

num2 = np.poly1d(num2)
den2 = np.poly1d(den2)

H_2 = sp.lti(num2,den2)
t_2,y_2,sec=sp.lsim(H_2,x_2,t_2)
mp.figure(num=3,figsize=(7,7))
mp.subplot(2,1,1)
mp.plot(t_2,y_2)
mp.title('Output of lowpass filter')
mp.xlabel("t")
mp.ylabel("y(t)")
mp.grid(True)

mp.subplot(2,1,2)
mp.plot(t_2,x_2)
mp.title('Input to lowpass filter')
mp.xlabel('t')
mp.ylabel('x(t)')
mp.grid("True")
mp.show()
```

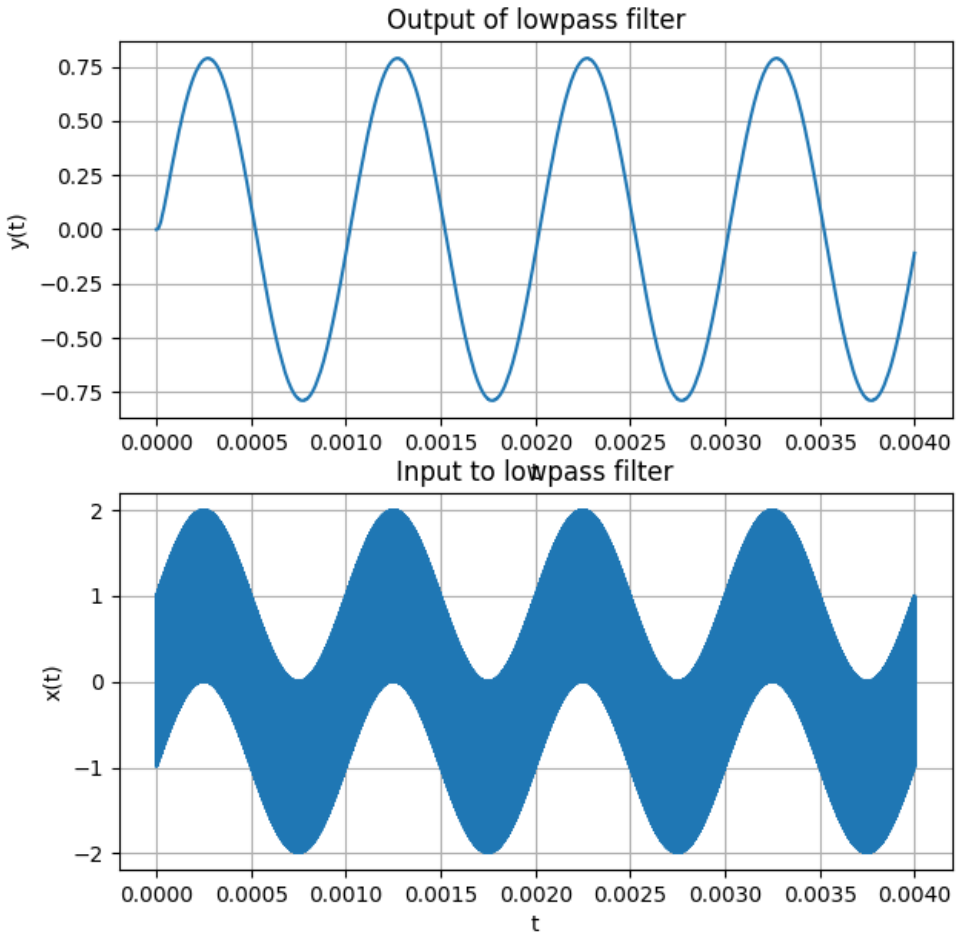

Figure 4: Time domain output for given inputs

Question 3

We already define the matrix for high pass input above, now we plot the magnitude and phase plots for it, the following code snippet helps to do it.

```
Vi = 1
A_3,b_3,V_3 = highpass_filter(10000,10000,1e-9,1e-9,1.586,Vi)
Vo2 = V_3[3]
w=np.logspace(0,8,801)
ss=1j*w
hf=lambdify(s,Vo2,'numpy')
v=hf(ss)
phi = np.angle(v)
mp.figure(num=4,figsize=(7,7))
mp.subplot(2,1,1)
mp.loglog(w,abs(v),lw=2)
mp.title('Bode plot high pass filter')
mp.xlabel(r'$\omega\rightarrow$',loc = "left")
mp.ylabel("Magnitude in log scale")
mp.grid(True)

mp.subplot(2,1,2)
mp.semilogx(w,phi,lw=2)
mp.title('Phase plot')
mp.xlabel('Frequency in log scale')
mp.ylabel('Phase in radians')
mp.grid("True")
mp.show()
```

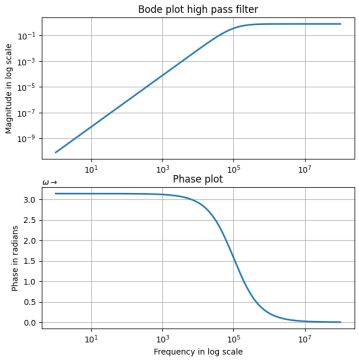

Figure 5: Bode plots for high pass filter

Question 4

Now we obtain the ouput for decaying sinusoid input of high and low frequencies after passing through high pass filter. The following Python code snippets helps to do it.
For low frequency,

```
t_4 = np.linspace(0.0,4e-3,100001)
x_4 = (np.sin(2000*np.pi*t_4))*np.exp((-10**3)*t_4)
simpVo2 = simplify(Vo2)
num = fraction(simpVo2)[0]
den = fraction(simpVo2)[1]
num2,den2 = get_Poly(num,den)

num2 = np.poly1d(num2)
den2 = np.poly1d(den2)

H = sp.lti(num2,den2)
t_4,y_4,sec=sp.lsim(H,x_4,t_4)
mp.figure(num=6,figsize=(7,7))
mp.subplot(2,1,1)
mp.plot(t_4,y_4)
mp.title('Output of highpass filter')
mp.xlabel('t',loc = 'left')
mp.ylabel('y(t)')

mp.subplot(2,1,2)
mp.plot(t_4,x_4)
mp.title('Input to Highpass filter')
mp.xlabel('t')
mp.ylabel('x(t)')
mp.grid("True")
mp.show()
```

For high frequency,

```
t_44 = np.linspace(0.0,3e-5,100001)
x_44 = (np.sin(2*(10**6)*np.pi*t_44))*np.exp((-10**5)*t_44)
simpVo2 = simplify(Vo2)
num = fraction(simpVo2)[0]
den = fraction(simpVo2)[1]
num2,den2 = get_Poly(num,den)

num2 = np.poly1d(num2)
den2 = np.poly1d(den2)
```

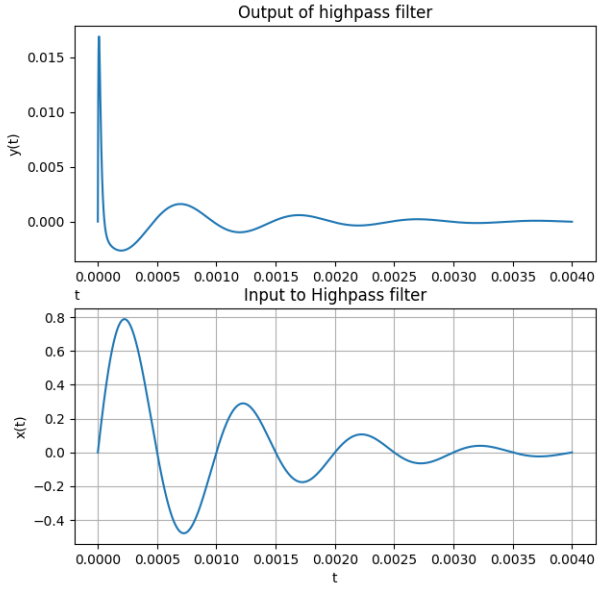

Figure 6: output ofdecaying sinusoids through high pass filter

```
H = sp.lti(num2,den2)
t_44,y_44,sec=sp.lsim(H,x_44,t_44)
mp.figure(num=7,figsize=(7,7))
mp.subplot(2,1,1)
mp.plot(t_44,y_44)
mp.title('Output of highpass filter')
mp.xlabel('t')
mp.ylabel('y(t)')

mp.subplot(2,1,2)
mp.plot(t_44,x_44)
mp.title('Input to Highpass filter')
mp.xlabel('t')
mp.ylabel('x(t)')
mp.grid("True")
mp.show()
```

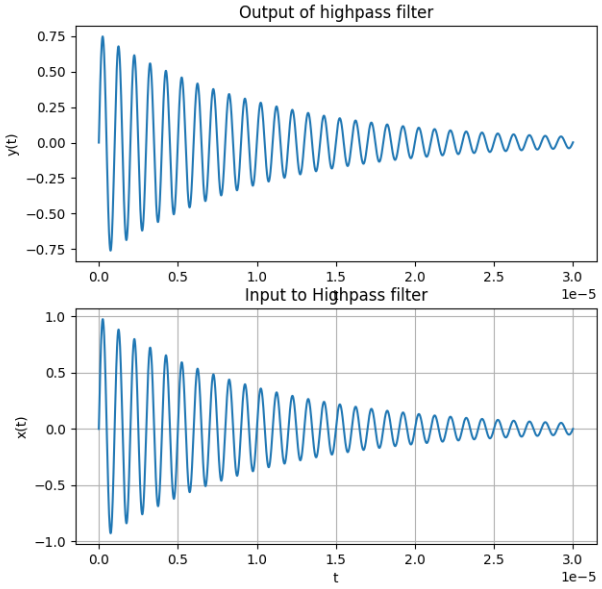

Figure 7: output ofdecaying sinusoids through high pass filter

Question 5

Now we obtain output when the unit step response is feeded to high pass filter. The following python code snippet helps to get the magnitude response plot for step input

```
Vi = 1/s
A_5,b_5,V_5 = highpass_filter(10000,10000,1e-9,1e-9,1.586,Vi)
Y_h=V_5[3]
w=mp.logspace(0,8,801)
ss=1j*w
hf=lambdify(s,Y_h,'numpy')
v=(hf(ss))
phi = np.angle(v)
mp.figure(num=8,figsize = (7,7))
mp.loglog(w,abs(v),lw=2)
mp.title("Magnitude Plot")
mp.xlabel(r'$\omega\rightarrow$',loc ='left')
mp.ylabel('Magnitude in log scale')
mp.grid(True)
mp.show()
```

Now, the following python code snippet helps to give the time domain output

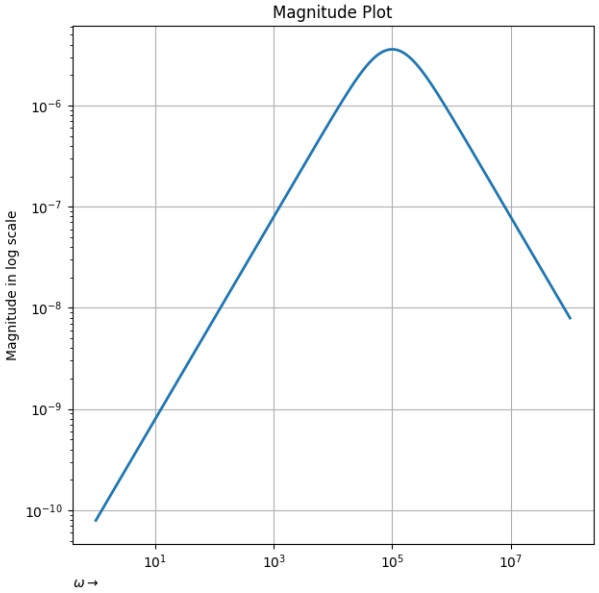

Figure 8: magnitude response plot for given step input

for step input to high pass filter

```
simpY_h = simplify(Y_h)
num = fraction(simpY_h)[0]
den = fraction(simpY_h)[1]
num2,den2 = get_Poly(num,den)
num2 = np.poly1d(num2)
den2 = np.poly1d(den2)
Y = sp.lti(num2,den2)
t_h = np.linspace(0.0,4e-3,10001)
t_h,y_h=sp.impulse(Y,None,t_h)
mp.figure(num=9,figsize = (7,7))
mp.plot(t_h,y_h)
mp.title('Time domain output to step input')
mp.xlabel('Time')
mp.ylabel('y(t)')
mp.ylim(-1,1)
mp.grid(True)
mp.show()
```

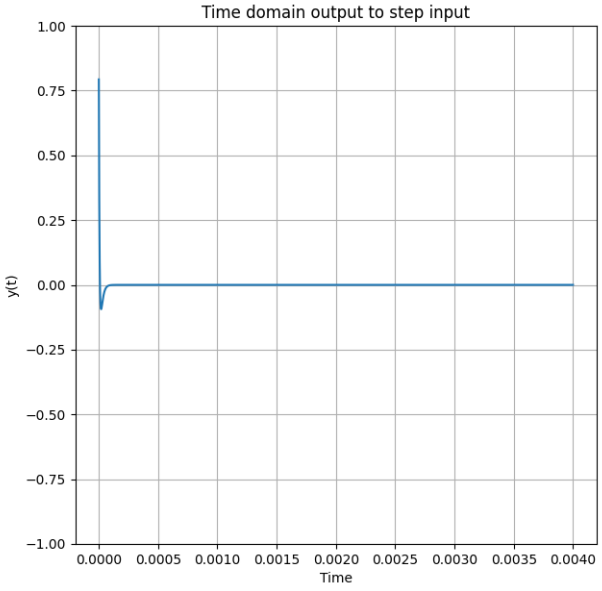

Figure 9: magnitude response plot for given step input

Conclusion

In conclusion, the sympy module has allowed us to analyze some quite complicated circuits and has been a very useful toolbox in Python for solving many circuit operations.