ASSIGNMENT 7: CIRCUIT ANALYSIS USING SYMPY AND LAPLACE TRANSFORMS

ROHIT KUMAR EE20B110

April 6, 2022

Abstract

The goal of this assignment is the following:

- To analyze Filters using Laplace Transform.
- To see how python can be used for symbolic Algebra.
- To plot graphs to understand high pass and low pass analog filters.

Low Pass Filter

The low pass filter that we use gives the following matrix equation after simplification of the modified nodal equations.

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

The python code snippet that declares the low pass function and solves the matrix equation to get the V matrix is as shown below:

```
s=symbols('s')
def lowpassfilter(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
```

High Pass Filter

The high pass filter we use gives the following matrix equations after simplification of the modified nodal equations

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

The python code snippet that declares the high pass function and solves the matrix equation to get the V matrix is as shown below:

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

The plot for the magnitude of the transfer function (magnitude bode plot) is as shown below:

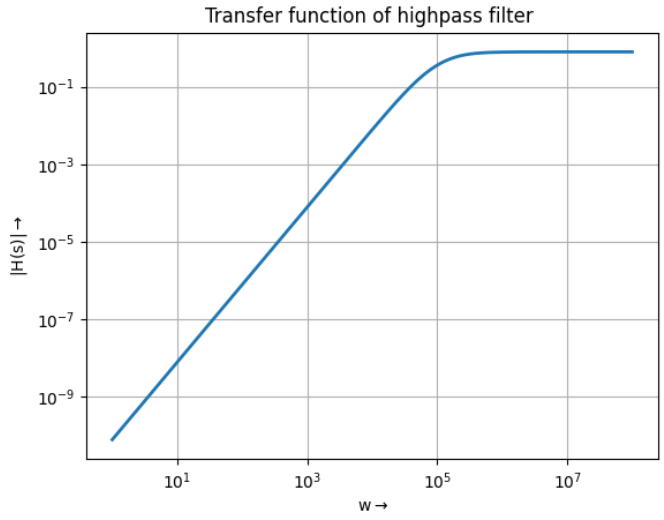

Figure 1: Magnitude bode plot of the high pass filter

Format Change of Sympy Functions

The sympy functions,that are expressed in terms of 's', must be converted to another form that is understood by sp.signal. This is done using the

sympyToTrFn() function.
The python code snippet is as shown:

```
# Creating a function to convert a sympy function into a version
# that is understood by sp.signal
def sympytotrnfn(func):
    num, den = simplify(func).as_numer_denom()
    num_list=Poly(num).all_coeffs()
    num_list = array(num_list, dtype='double')
    den_list=Poly(den).all_coeffs()
    den_list = array(den_list, dtype='double')
    h_lowpass=sp.lti(num_list,den_list)
    return h_lowpass
```

Step Response Of the Low Pass Filter

In order to find the step response of the low pass filter, we need to assign $Vi(s) = 1/s$. The python code below explains it.

```
h_lowpass=sympytotrnfn(H)

t=linspace(0,0.001,1000)
Vi = np.multiply((np.sin(2000*np.pi*t)+np.cos(2000000*np.pi*t)),np.heaviside(t,1))
c=heaviside(t,0)
t,y,svec=sp.lsim(h_lowpass,c,t)
plt.plot(t,c,label='V_input')
plt.plot(t, y,label='V_output')
xlabel("t"+"$\\rightarrow$")
ylabel("V"+"$\\rightarrow$")
title("step response of the circuit")
grid(True)
legend()
show()
```

The plot for the step response of the low pass circuit is as shown:

Response to Sum Of Sinusoids

When the input is a sum of sinusoids like,

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

Then the output response for the lowpass filter can easily be found as shown in the code snippet.

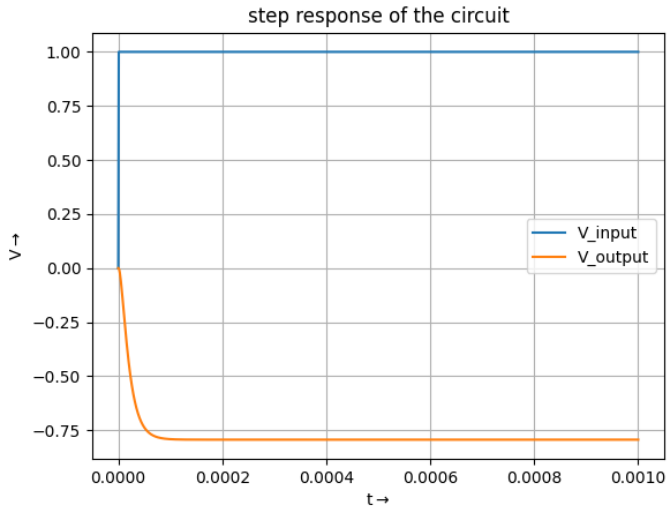

Figure 2: Step response of low pass filter.

```
t,y,svec=sp.lsim(h_lowpass,Vi,t)
plt.plot(t,Vi,label='V_input')
plt.plot(t,y,label='V_output')
xlabel("t"+"$\\rightarrow$")
ylabel("V"+"$\\rightarrow$")
title("response of the circuit to vi(t) =(sin(2000t)+cos(2*10^6t))*u(t)")
plt.grid(True)
legend()
show()
```

The output response to the sum of sinusoids for the low pass filter is as shown:

Response to Damped Sinusoids

In this case we assign the input voltage as a damped sinusoid like,
Low frequency,

$$V_i(t) = e^{-500t}(\ \cos(2000\pi t)\ )u_o(t)\ Volts$$

High frequency,

$$V_i(t) = e^{-500t}(\ \cos\left(2 * 10^6\pi t\right)\ )u_o(t)\ Volts$$

The high frequency and low frequency plots of the input damped sinusoids are as shown:

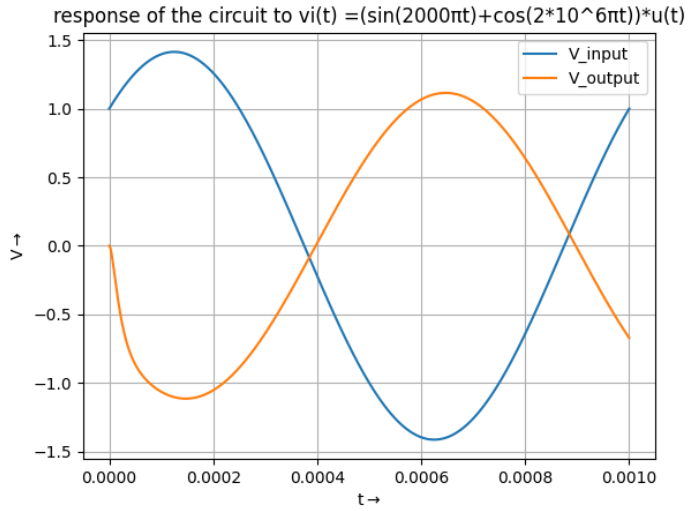

Figure 3: Output response to the sum of sinusoids.

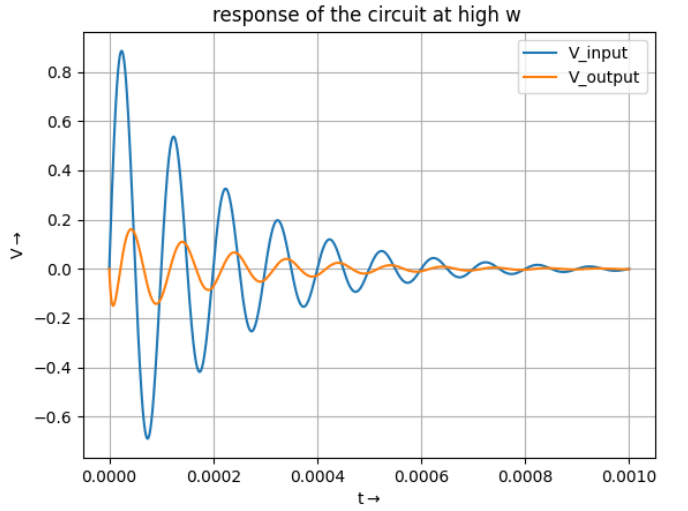

Figure 4: High frequency damped sinusoid

The python code snippet to execute the above is as shown:

Step Response of High Pass Filter

In order to find the step response of the high pass filter, we need to assign $Vi(s) = 1/s$. The python code below explains it.

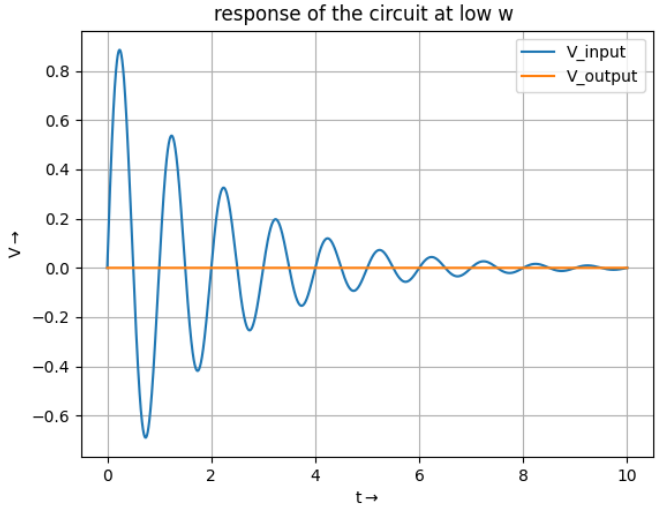

Figure 5: Low frequency damped sinusoid

```
t=linspace(0,0.001,10000)
vi=heaviside(t,0)
plt.plot(t,vi,label='V_input')
t,y,svec=sp.lsim(h_highpass,vi,t)
plt.plot(t,y,label='V_output')
xlabel("t"+"$\\rightarrow$")
ylabel("V"+"$\\rightarrow$")
title("step response of the circuit")
plt.grid(True)
legend()
show()
```

The plot of the step response for a high pass filter is as shown:

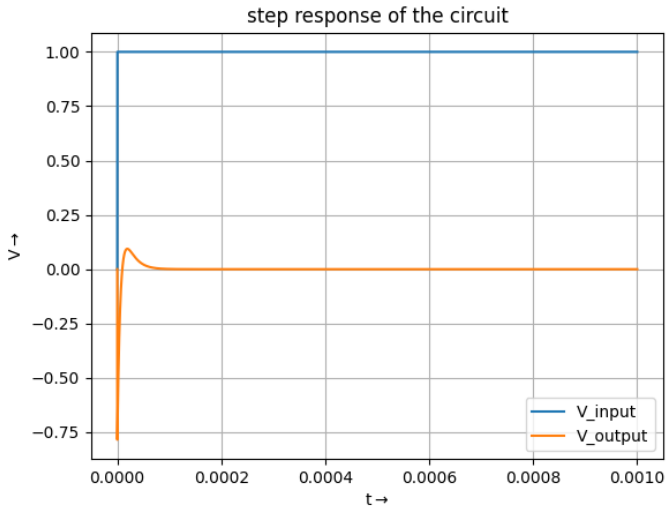

Figure 6: Step response for a high pass filter

The unit step response, as expected is high at t=0 when there is an abrupt change in the input. Since there is no other change at large time values outside the neighbourhood of 0, the Fourier transform of the unit step has high values near 0 frequency, which the high pass filter attenuates.

Conclusion

In conclusion, the sympy module has allowed us to analyse quite complicated circuits by analytically solving their node equations. We then interpreted the solutions by plotting time domain responses using the signals toolbox. Thus, sympy combined with the scipy.signal module is a very useful toolbox for analyzing complicated systems like the active filters in this assignment.