EE2703 Applied Programming Lab
Assignment 8

Name: Neham Hitesh Jain
Roll Number: EE19B084

April 23, 2021

Abstract

In this assignment, we use Sympy to analytically solve a matrix equation governing an analog circuit. We look at two circuits, an active low pass filter and an active high pass filter. We create matrices using node equations for the circuits in sympy, and then solve the equations analytically. We then convert the resulting sympy solution into a numpy function which can be called. We then use the signals toolbox we studied in the last assignment to understand the responses of the two circuits to various inputs.

Low Pass Filter

Consider the circuit given below,

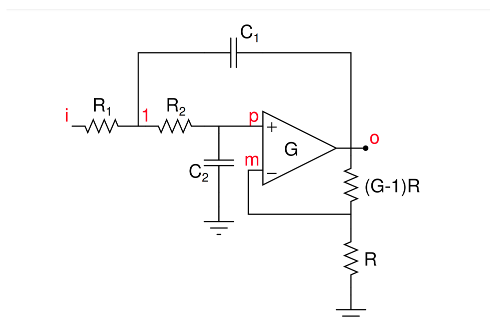


Figure 1: Low Pass Filter Circuit

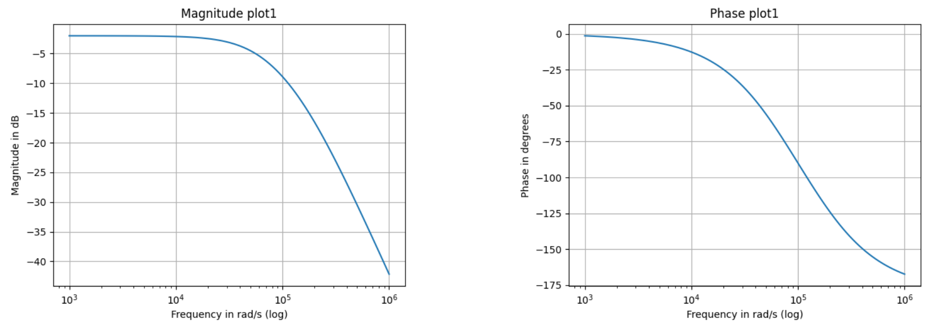
The above circuit gives the following matrix after simplification of Modified Nodal Equations.

$$\begin{bmatrix} 0 & 0 & 1 & -1/G \\ \frac{-1}{sR_2C_2} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ \frac{-1}{R_1} - \frac{1}{R_2} - s*C_1 & \frac{1}{R_2} & 0 & sC_1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{-V_i(s)}{R_1} \end{bmatrix}$$

```python
def lowpass(R1=10e3,R2=10e3,C1=1e-9,C2=1e-9,G=1.586,Vi=1,s=symbols('s')):
    """Solve the given lowpass filter circuit for a given input 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/R2,0,s*C1]])

    b=Matrix([0,0,0,-Vi/R1])
    V=A.inv()*b

    return V
```

Magnitude response

The magnitude response of the circuit is,



(a) Magnitude Plot (b) Phase Plot

Figure 2: Bode Plot of Low Pass Circuit

```python
def get_bode_plot(H,title):
    ''' Plots the magnitude and phase response of a given transfer function '''

    w, mag, phase = H.bode()
    #Plotting Magnitude Response of Transfer Function
    p = General_Plotter("Frequency in rad/s (log)","Magnitude in dB","Magnitude
        plot"+title)
    p.semilogx(w,mag)

    #Phase Response of Transfer Function
    p = General_Plotter("Frequency in rad/s (log)","Phase in degrees","Phase
        plot"+title)
    p.semilogx(w,phase)
```

Clearly, the circuit acts as a low pass filter as the magnitude response drops rapidly after a certain frequency.

High Pass Filter

Consider the circuit below, The above circuit gives the following matrix after simplification

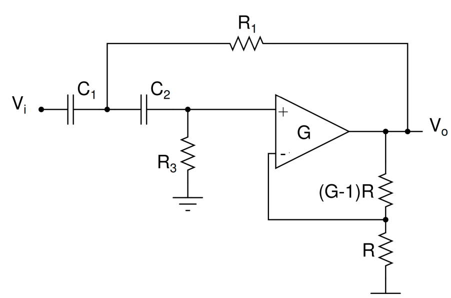


Figure 3: High Pass Filter Circuit

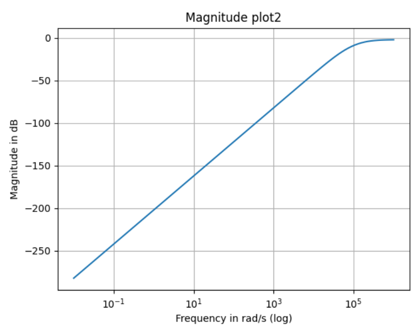
of Modified Nodal Equations.

$$\begin{bmatrix} 0 & -1 & 0 & 1/G \\ \frac{s*C_2*R_3}{1+s*C_2*R_3} & 0 & -1 & 0 \\ 0 & G & -G & 1 \\ -s*C_2-\frac{1}{R_1}-s*C_1 & 0 & s*C_2 & \frac{1}{R_1} \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -V_i(s)*s*C_1 \end{bmatrix}$$

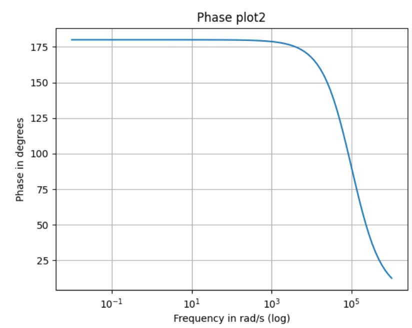
Magnitude response

The magnitude response of the circuit is:

```
#Step Response of the Highpass Circuit
V_h = highpass()[3]
H2 = sympy_to_lti(V_h)
get_bode_plot(H2,title="2")
```



(a) Magnitude Plot



(b) Phase Plot

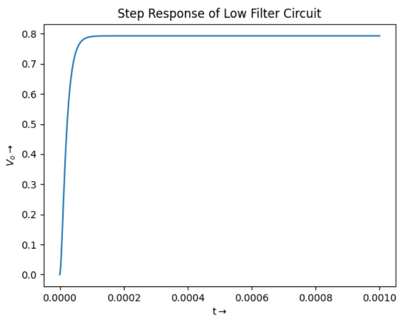
Figure 4: Bode Plot of High Pass Circuit

Clearly, the circuit acts as a high pass filter as the magnitude response starts with low values and then increases and remains constant with frequency after a certain cut-off frequency.

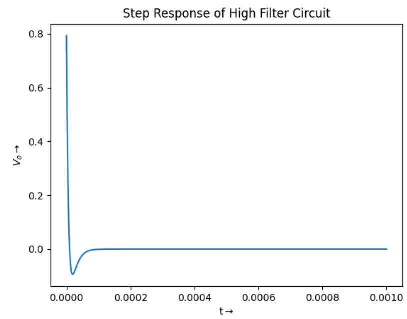
1 Converting Sympy to Scipy

On obtaining the transfer function in it's symbolic representation, we now convert it to a form that can be used by Scipy's Signals package.

```
def sympy_to_lti(xpr, s=symbols('s')):
    """ Convert Sympy transfer function polynomial to Scipy LTI """

    num, den = simplify(xpr).as_numer_denom() # returns the expressions
    p_num_den = poly(num, s), poly(den, s)
    num,den = p_num_den[0].all_coeffs(), p_num_den[1].all_coeffs()

    num=list(map(float, num))
    den=list(map(float, den))

    return sp.lti(num, den)
```



(a) Low Pass Filter Circuit



(b) High Pass Filter Circuit

Figure 5: Step Response

Step Response

We plot the outputs of the two systems to a unit step below:

```
Steady state value of low pass filter step response: 0.7930
Steady state value of high pass filter step response: 0.0000
Initial value of low pass filter step response: 0.0000
Initial value of high pass filter step response: 0.7930
```

- We observe that the low pass filter inverts the step and attenuates it by a factor of 0.793. We observe a transient which decays quite fast, again as we expected as the system is almost critically damped.
- The initial value of the low pass response to the step is 0 as the AC gain of the low pass filter is 0. We can interpret the step as an infinte frequency signal, so the response to it would be according to the AC gain.
- The transient of the high pass response is also similar. There are two differences. Firstly, the steady state response of the high pass filter to the step is 0. This is because it only allows frequencies higher than the cutoff to pass through. Since DC inputs have a frequency of 0, it is completely filtered out. In other words, the DC gain is 0. This is similar to what is done when two systems are coupled for AC signals.
- The other difference is that the response overshoots the steady state value of 0, reaches an extremum, then settles back to 0, unlike the response of the low pass filter which steadily approaches the steady state value with no extrema. This means that the impulse response must equal zero at one or more time instants. Since the impulse response is the derivative of the step response, this therefore means that the step response must have at least one extremum. This explains the behaviour of the step response of the high pass filter.

Sum of Low and High Frequency Sinusoids

We analyse the responses to the following input:

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

```python
def mixed_freq_sinusoid(t):
''' Function that gives the sum of sinusoid with different frequencies '''
    return (np.sin(2000*np.pi*t)+np.cos(2000000*np.pi*t))*np.heaviside(t,0.5)

#Plotting sum of sinusoids and the responses of circuits to the sum Of sinusoids
t_ls = np.linspace(0,0.001,100000)
t_hs = np.linspace(0,0.00001,100000)

Vi = mixed_freq_sinusoid(t_ls)
Vi_h = mixed_freq_sinusoid(t_hs)

p.general_plot(t_ls,Vi)

t,Vo,_ = sp.lsim(H1,Vi,t_ls)
p.general_plot(t,Vo)

t,Vo,_ = sp.lsim(H2,Vi_h,t_hs)
p.general_plot(t,Vo)
```

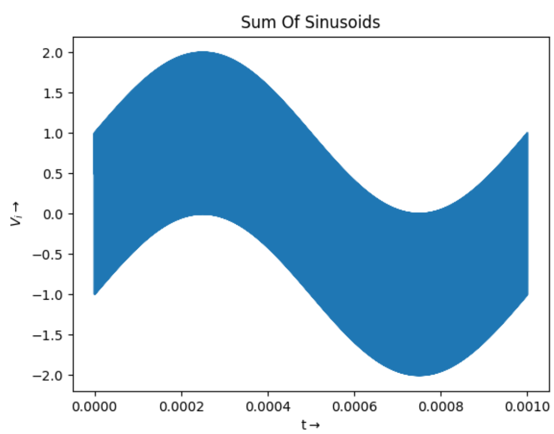


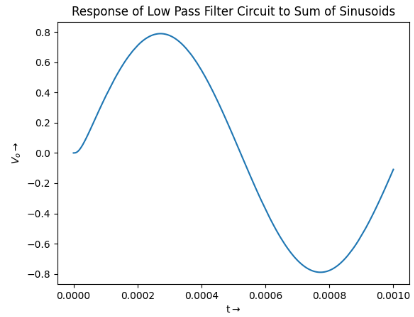
Figure 6: Sum of Sinusoids

We first plot the response of the low pass filter in a very short time range and the high pass filter in a very large time range.

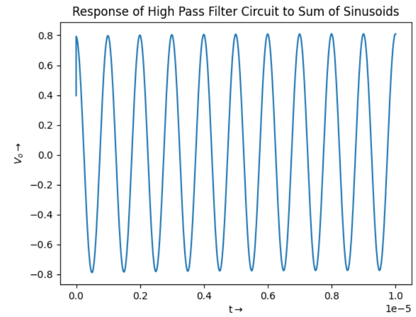
- These time ranges allow us to notice the heavily attenuated part of the input better. We can see that an extremely low amplitude high frequency sinusoid rides on top of a slow response in the output of the low pass filter. This is because the low pass filter has heavily attentuated the high frequency component.
- Similaraly, in the high pass filter, a heavily attenuated low frequency sinusoid modulates a high frequency sinusoid. This is because the high pass filter highly attenuated the low frequency component.

System Response to Damped Sinusoid

We analyse the Low Pass and High Pass filter using a damped sinusoid of low frequency and high frequency.



(a) Low Pass Filter Circuit



(b) High Pass Filter Circuit

Figure 7: System Response to Sum of Sinusoids

```
def damped_sinusoid(t,decay,freq):
''' Function that gives a damped sinusoid '''

    return np.cos(freq*t)*np.exp(-decay*t) * (t>0)
```

High Frequency Damped Sinusoid

We analyse the outputs to a high frequency damped sinusoid given as:

$$v_i(t) = \cos(10^7 t)e^{-3000t}u(t)$$

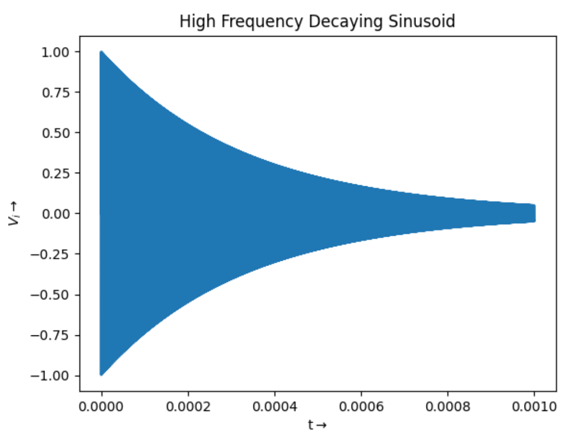
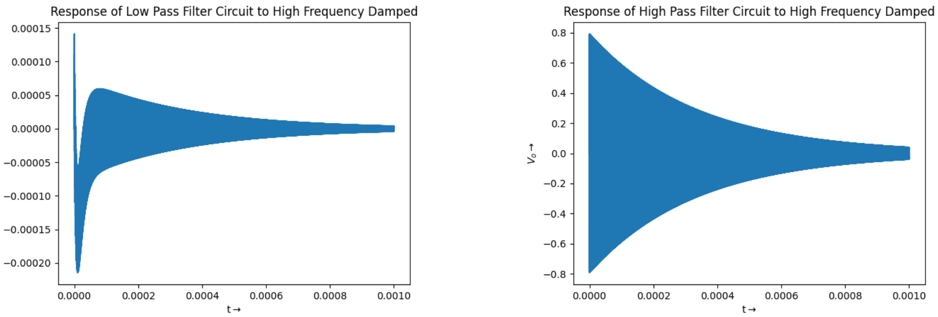


Figure 8: High Frequency Damped Circuit

```
Vi_h = damped_sinusoid(t_h,decay=3e3,freq=1e7)

p=General_Plotter(r't$\rightarrow$',r'$V_{i}\rightarrow$',"High Frequency Decaying
    Sinusoid")
p.general_plot(t_h,Vi_h)

t,Vo,_ = sp.lsim(H1,Vi_h,t_h)
```



(a) Low Pass Filter Circuit

(b) High Pass Filter Circuit

Figure 9: System Response to High Frequency Damped Sinusoid

```
p=General_Plotter(r't$\rightarrow$',r'$V_{o}\rightarrow$',"Response of Low Pass
    Filter Circuit to High Frequency Damped")
p.general_plot(t,Vo)

t,Vo,_ = sp.lsim(H2,Vi_h,t_h)
p=General_Plotter(r't$\rightarrow$',r'$V_{o}\rightarrow$',"Response of High Pass
    Filter Circuit to High Frequency Damped")
p.general_plot(t,Vo)
```

- The low pass filter responds fast and heavily attenuates the high frequency sinusoid. The output decays as the input also decays.
- The high pass filter responds by more or less letting the input pass through as is, with only a slight attenuation. So the output decays as the input does.

Low Frequency Damped Sinusoid

We analyse the outputs to a low frequency damped sinusoid given as:

$$v_i(t) = \cos(10^3 t)e^{-10t}u(t)$$

The input waveform is plotted below:

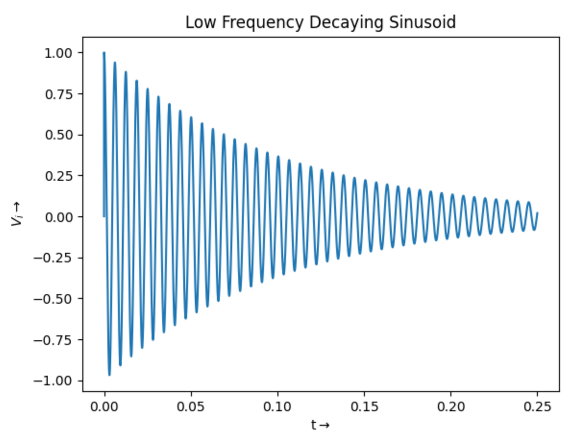
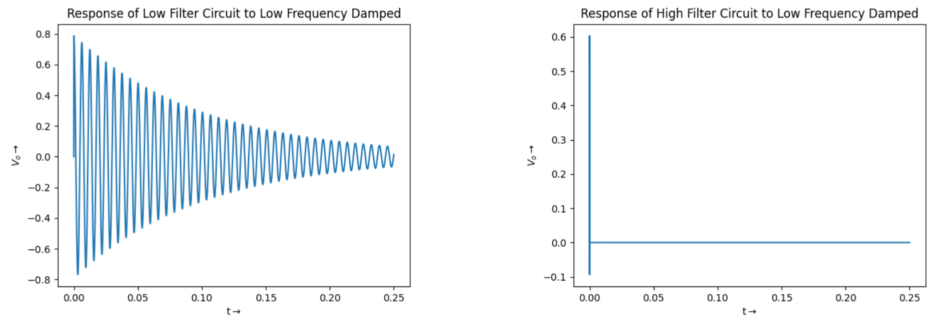


Figure 10: Low Frequency Damped Sinusoid



(a) Low Pass Filter Circuit

(b) High Pass Filter Circuit

Figure 11: System Response to Low Frequency Damped Sinusoid

```
#Plotting the Low Frequency damped Sinusoid and the response of circuits to this
Vi_l = damped_sinusoid(t_l,decay=1e1,freq=1e3)

p=General_Plotter(r't$\rightarrow$',r'$V_{i}\rightarrow$',"Low Frequency Decaying
    Sinusoid")
p.general_plot(t_l,Vi_l)

t,Vo,_ = sp.lsim(H1,Vi_l,t_l)
p=General_Plotter(r't$\rightarrow$',r'$V_{o}\rightarrow$',"Response of Low Filter
    Circuit to Low Frequency Damped")
p.general_plot(t,Vo)

t,Vo,_ = sp.lsim(H2,Vi_l,t_l)
p=General_Plotter(r't$\rightarrow$',r'$V_{o}\rightarrow$',"Response of High Filter
    Circuit to Low Frequency Damped")
p.general_plot(t,Vo)
```

Conclusions

- The low pass filter responds by letting the low frequency sinusoid pass through without much additional attenuation. The output decays as the input also decays.
- The high pass filter responds by quickly attenuating the input. Notice that the time scales show that the high pass filter response is orders of magnitudes faster than the low pass response. This is because the input frequency is below the cutoff frequency, so the output goes to 0 very fast.
- 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.