EE2703-Assignment7

EE19B094

April 2021

Abstract

This week's assignment involves the analysis of high and low pass filters using laplace transforms. Python's symbolic solving library, sympy is a tool we use in the process to handle our requirements in solving Modified Nodal Analysis equations. And with the help of Scipy's signal module, we analyse the filters by graphing the magnitude response and the output voltage of the system for different forms of input voltage.

1 Low Pass Filter

Consider the circuit given below, The above circuit gives the following ma-

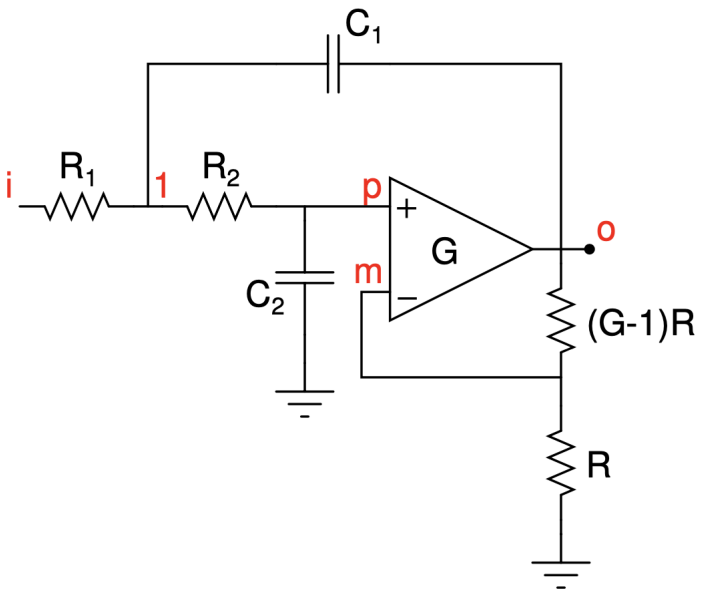

Figure 1: Low Pass Filter Circuit

trix after simplification of Modified Nodal Equations.

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

```
#KCL for low pass filter
def low_pass_filter(R1=10e3,R2=10e3,C1=1e-9,C2=1e-9,G=1.586,Vi=1):
   s = sympy.symbols('s')
   A = sympy.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 = sympy.Matrix([0,0,0,-Vi/R1])
```

```
    V = A.inv()*b
    return (A,b,V)
```

1.1 Magnitude response

The magnitude response of the circuit is,

```
A,b,V = low_pass_filter() #Solve for low pass filter
H_lowpass = V[3]
hf = sympy.lambdify(s,H_lowpass,"numpy") #Convert transfer function
    to python function
v = hf(ss)    #Do freq sweep and plot magnitude response
g1 =
    General_Plotter("frequency$\longrightarrow$","|H(s)|$\longrightarrow$","Bode
    plot of low pass filter",[],1)
g1.plot_loglog(w,abs(v))
g1.show()
```

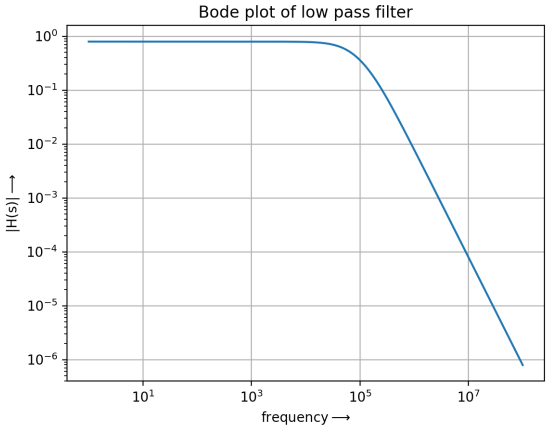

Figure 2: Low pass filter Magnitude response

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

2 High Pass Filter

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

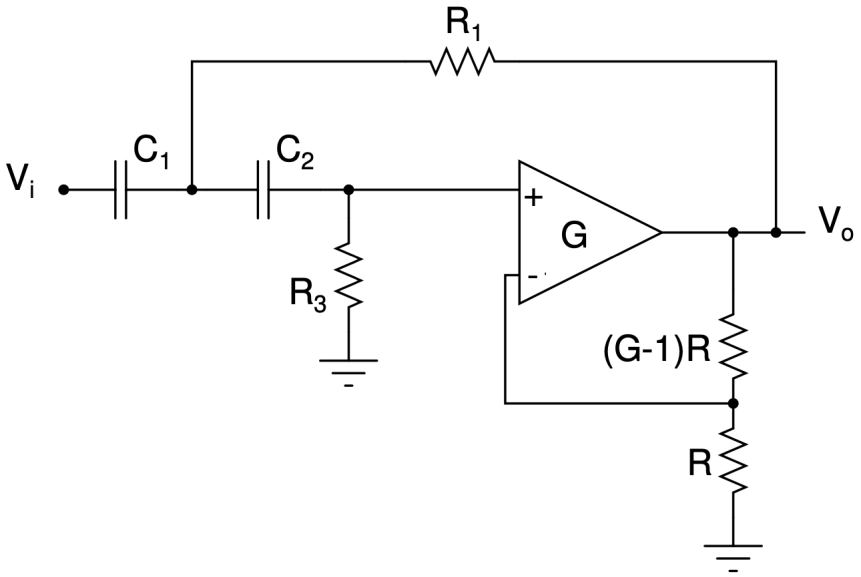

Figure 3: High Pass Filter Circuit

simplification of Modified Nodal Equations.

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

2.1 Magnitude response

The magnitude response of the circuit is,

```
A,b,V = high_pass_filter() #Solve for high pass filter
H_highpass = V[3]
hf = sympy.lambdify(s,H_highpass,"numpy")  #Convert transfer
    function to python function
v = hf(ss)    #Do freq sweep and plot magnitude response
g1 =
    General_Plotter("frequency$\longrightarrow$","|H(s)|$\longrightarrow$","Bode
    plot of high pass filter",[],2)
g1.plot_loglog(w,abs(v))
g1.show()
```

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.

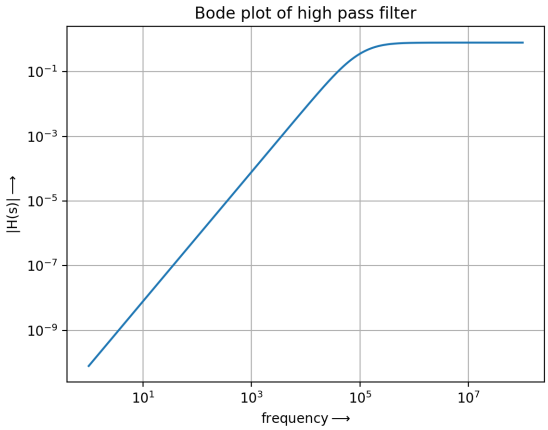

Figure 4: High pass filter Magnitude response

3 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.

```
#Convert transfer function in sympy to scipy
def convert_sympy_to_scipy(H):
   H = sympy.simplify(H)
   n,d = sympy.fraction(H)  #Get numerator and denominator from H
   num,den = sympy.Poly(n,s), sympy.Poly(d,s) #Convert them into
       polynomial in 's'
   num,den = num.all_coeffs(), den.all_coeffs() #Get coefficients
       of polynomials
   num,den = [float(f) for f in num], [float(f) for f in den]
       #Store them in list
   return num,den
```

We then use this numerator and denominator coefficients with scipy's various functions to get output for various inputs.

4 Step Responses

We now calculate and plot the output for both filters to step input.

```
#Calculate step response to given transfer function in sympy
def step_response(H):
   num,den = convert_sympy_to_scipy(H)  #Sympy to scipy
   den.append(0)                  #Multiply H by 1/s for step input
   H = sp.lti(num,den)
   t,x = sp.impulse(H,T = np.linspace(0,1e-3,100000)) #Calculate
```

```
      Impulse response
   return t,x

t,x = step_response(H_lowpass) #Get stepresponse for low pass
    filter and plot
g1 =
    General_Plotter("time$\longrightarrow$","$V_o(t)\longrightarrow$","Step
    response for low pass filter",[],3)
g1.plot_points(t,x)
g1.show()

t,x = step_response(H_highpass) #Get stepresponse for high pass
    filter and plot
g1 =
    General_Plotter("time$\longrightarrow$","$V_o(t)\longrightarrow$","Step
    response for high pass filter",[],4)
g1.plot_points(t,x)
g1.show()
```

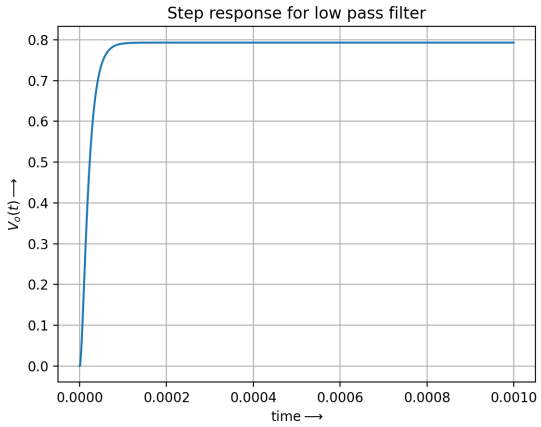

Figure 5: Low pass filter step response

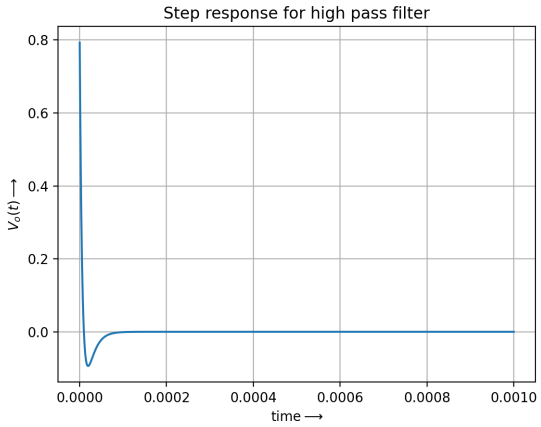

Figure 6: High pass filter step response

The unit step response,for the high pass filter, as expected is high at t=0 when there is an abrupt change in the input. For the steady value of the input Vi, the capacitor C1 in the circuit acts as an open switch and allows no current to pass through. Consequently, we have a zero voltage at the output node.

5 Two frequency Sinusoids

We now plot the output for both the filters corresponding to a input with two sinuoids with one small frequency and one big frequency. The input signal is given by:

$$vi = (sin(2000 * \pi * t) + cos(2000000 * \pi * t))u(t)$$

```
#Sum of sinusoids of two frequencies
def sum_of_sinuosids(t):
   return (np.sin(2000*np.pi*t)+np.cos(2e6*np.pi*t))

#Function to calculate output corresponding to given transfer
    function and input function
def general_input(H,input_time,max_time=1e-3):
   num,den = convert_sympy_to_scipy(H)  #Sympy to scipy
   H = sp.lti(num,den)
   t = np.linspace(0,max_time,100000)  #Time range
   t,y,svec = sp.lsim(H,input_time(t),t) #Calculate output
   return t,y

#Plotting input signal
t = np.linspace(0,1e-3,1000000)
```

```
g1 =
    General_Plotter("time$\longrightarrow$","$V_i(t)\longrightarrow$","Two
    frequency signal",[],5)
g1.plot_points(t,sum_of_sinuosids(t))
g1.show()

#Low pass filter on TFS
t1,y1 = general_input(H_lowpass,sum_of_sinuosids) #Solve for low
    pass filter
g1 =
    General_Plotter("time$\longrightarrow$","$V_o(t)\longrightarrow$","Low
    pass filter on Two frequency signal",[],6)
g1.plot_points(t1,y1)
g1.show()

#High pass filter on TFS
t1,y1 = general_input(H_highpass,sum_of_sinuosids,1e-5) #Solve for
    high pass filter
g1 =
    General_Plotter("time$\longrightarrow$","$V_o(t)\longrightarrow$","High
    pass filter on Two frequency signal",[],7)
g1.plot_points(t1,y1)
g1.show()
```

The input signal along side the output signals are, We see that the high

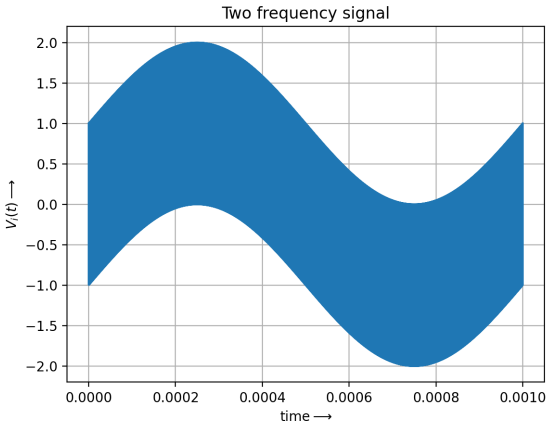

Figure 7: Input two frequency sinusoid signal

frequency part of the signal has been attenuated and only the low frequency remains. This is what we expected from a low pass filter. Similarly for the high pass filter, only the high frequency signal remains while the other gets attenuated. Thus, we can confirm that the cut-off frequencies for both the circuits lie in between the frequencies of the input signal.

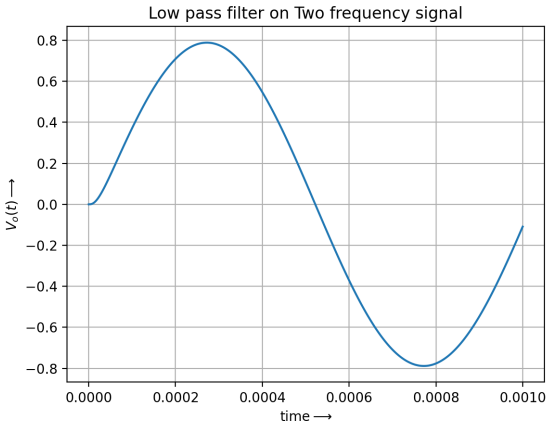

Figure 8: Low pass filter response

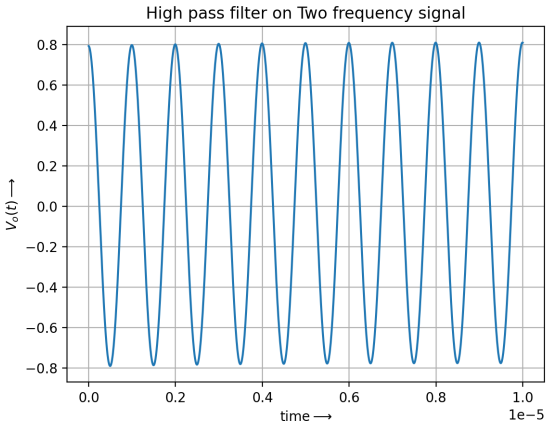

Figure 9: High pass filter response

6 Damped Sinusoids

We now observe and plot outputs to damped sinusoids to different frequencies. Just like the two frequency sinusoids case, we expect the low pass filter to pass low frequency signal while the opposite for high frequency signal.

6.1 Low frequency damped sinusoid

The low frequency input signal is,

$$f(t) = sin(10^3 * t) * e^{-10t} \tag{1}$$

```
#Low frequency damped sinusoid
def damped2(t,decay=1e1,freq=1e3):
   return np.cos(freq*t)*np.exp(-decay*t) * (t>0)

#Low pass filter on low freq damped
t = np.linspace(0,0.5,1000000)
t3,y3 = general_input(H_lowpass,damped2,0.5) #Solve for low pass
    filter
g1 =
    General_Plotter("time$\longrightarrow$","Voltage$\longrightarrow$","Low
    pass filter on low freq damped",[],8)
g1.plot_points(t,damped2(t),"Input signal") #Plotting input signal
g1.plot_points(t3,y3,"Output signal")  #Plotting output signal
g1.show(1)

#High pass filter on low freq damped
t3,y3 = general_input(H_highpass,damped2,0.5) #Solve for high pass
    filter
g1 =
    General_Plotter("time$\longrightarrow$","Voltage$\longrightarrow$","High
    pass filter on low freq damped",[],9)
g1.plot_points(t,damped2(t),"Input Signal") #Plotting input signal
g1.plot_points(t3,y3,"Output Signal")  #Plotting output signal
g1.show(1)
```

The output for both the filters are, Just as expected, the low pass filter

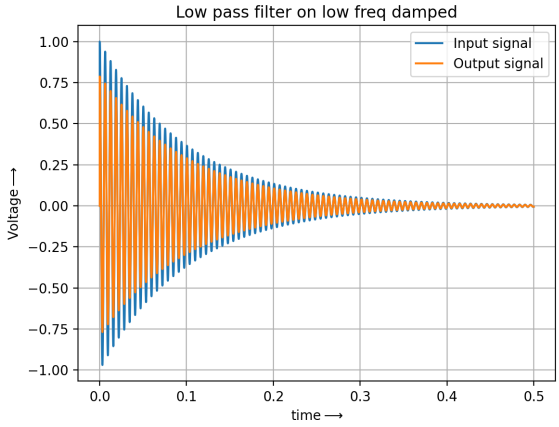

Figure 10: Low pass filter response to low freq damped

allowed the signal to pass while the high pass signal attenuated it.

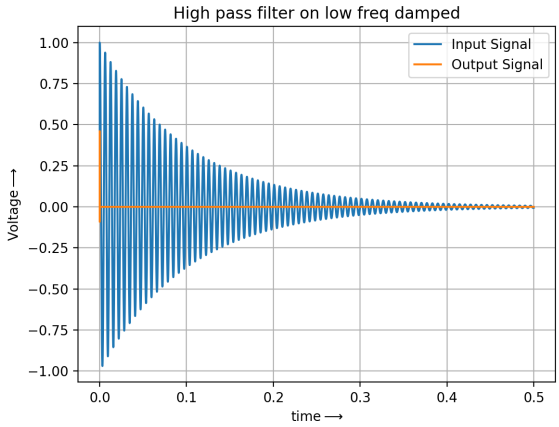

Figure 11: High pass filter response to low freq damped

6.2 High frequency damped sinusoid

The high frequency input signal is,

$$f(t) = sin(10^7 * t) * e^{-3*10^3 t} \tag{2}$$

```
#Low pass filter on high freq damped
t = np.linspace(0,1e-3,1000000)
t2,y2 = general_input(H_lowpass,damped1) #Solve for low pass filter
g1 =
    General_Plotter("time$\longrightarrow$","damped$\longrightarrow$","Low
    pass filter on high freq damped",[],10)
g1.plot_points(t,damped1(t),"Input Signal") #Plotting input signal
g1.plot_points(t2,y2,"Output Signal")  #Plotting output signal
g1.show(1)

#High pass filter on high freq damped
t2,y2 = general_input(H_highpass,damped1) #Solve for high pass
    filter
g1 =
    General_Plotter("time$\longrightarrow$","damped$\longrightarrow$","High
    pass filter on high freq damped",[],11)
g1.plot_points(t,damped1(t),"Input Signal") #Plotting input signal
g1.plot_points(t2,y2,"Output Signal")  #Plotting output signal
g1.show(1)
```

The corresponding outputs are, Now, this time as expected the low pass filter attenuated the signal to zero, while high pass filter allowed it to pass.

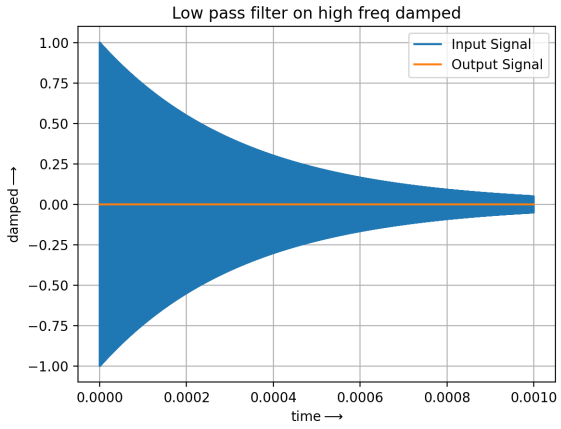

Figure 12: Low pass filter response to high freq damped

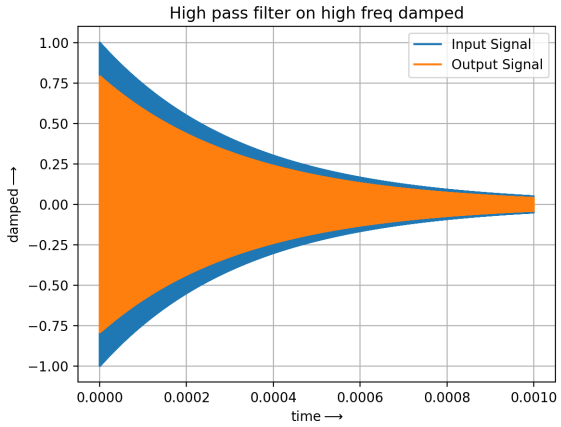

Figure 13: Low pass filter response to high freq damped

7 Conclusion

In conclusion, the sympy module has allowed us to analyse quite complicated circuits by analytically solving their node equations. We then interpreted the circuit behaviour by plotting time domain responses of them to various inputs 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.