Assignment 7

EE2703- APPLIED PROGRAMMING LANGUAGE - SUHAS C EE20B132

Date - 08/04/22

Abstract

This week's assignment involves the analysis of
lters using laplace transforms. Python's symbolic solving library, sympy is a tool we use in the process to handle our requirements in solving Modi
ed Nodal Analysis equations. Besides this the library also includes useful classes to handle the simulation and response to inputs. Coupled with scipy's signal module, we are able to analyse both High pass and low pass
lters, both second order, realised using a single op amp

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:

```
# Defifning Low Pass Filter
def lowpass(R1,R2,C1,C2,G,Vi):

    A=Matrix([[0,0,1,-1/G],                                    # Conductance Matrix
              [-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])                                    # Source Matrix
```

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

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

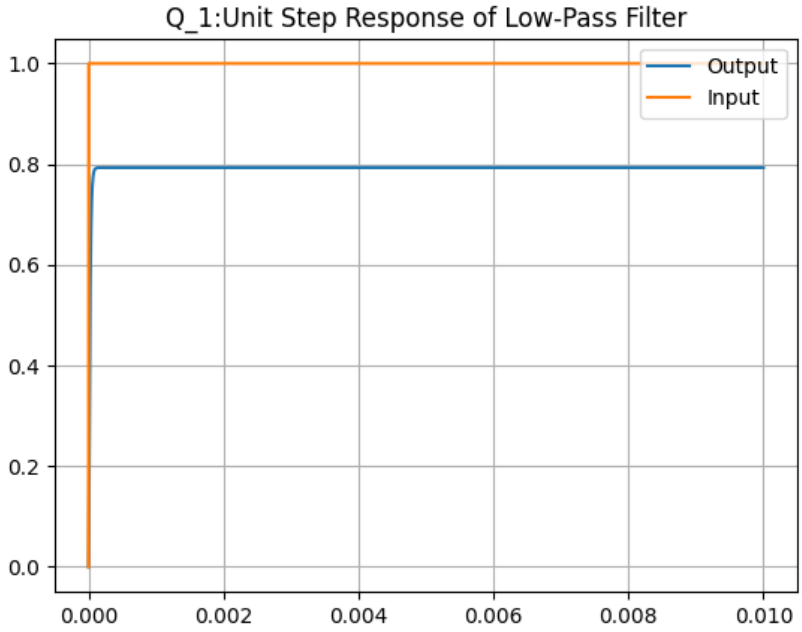

Figure 1: Low pass filter Magnitude response

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:

```
# Defining High Pass Filter
```

```
def highpass(R1,R3,C1,C2,G,Vi):

    A = Matrix([[0,0,1,-1/G],                                    # Conductance Matrix
                [0,G,-G,-1],
                [s*C2*R3/(1+s*C2*R3),-1,0,0],
                [(s*C2+s*C1+1/R1),-s*C2,0,-1/R1]])

    b = Matrix([0,0,0,s*C1*Vi])                                  # Source Matrix

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

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

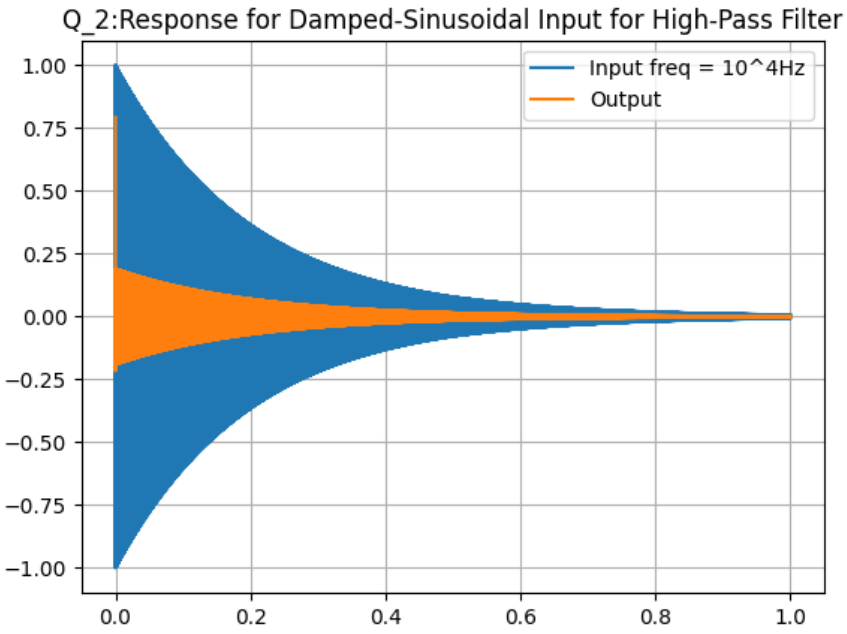

Figure 2: High pass filter Magnitude response

Change of Format in 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:

```
# Function for converting SymPy symbols to Polynomials
def Symbols_to_Polynomial(H):
n,d=fraction(simplify(H))
return (np.array(Poly(n,s).all_coeffs(),dtype=float),np.array(Poly(d,s).all_coeffs()
```

Low Pass Filter Step Response

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.

```
# Question_1

A1,b1,V1 = lowpass(10000,10000,1e-9,1e-9,1.586,1)          # Finding the Tansfer Fu
Nr_H1,Dr_H1 = Symbols_to_Polynomial(V1[3])
H1 = sp.lti(Nr_H1,Dr_H1)                                   # Converting Sympy Funct
t  = np.linspace(0,0.01,50001)                             # Time steps for plottin

Vi1 = np.ones(50001)                                       # Unit Step input functi
Vi1[0] = 0
t,Vo1,svec = sp.lsim(H1,Vi1,t)                             # Finding the output in


py.figure(0)
py.plot(t,abs(Vo1),label = 'Output')                       # plotting the input and
py.plot(t,Vi1,label = 'Input')
py.legend(loc = 'upper right')
py.title(r'Q_1:Unit Step Response of Low-Pass Filter')
py.grid(True)
py.savefig("suhas/figure0.png")
py.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.

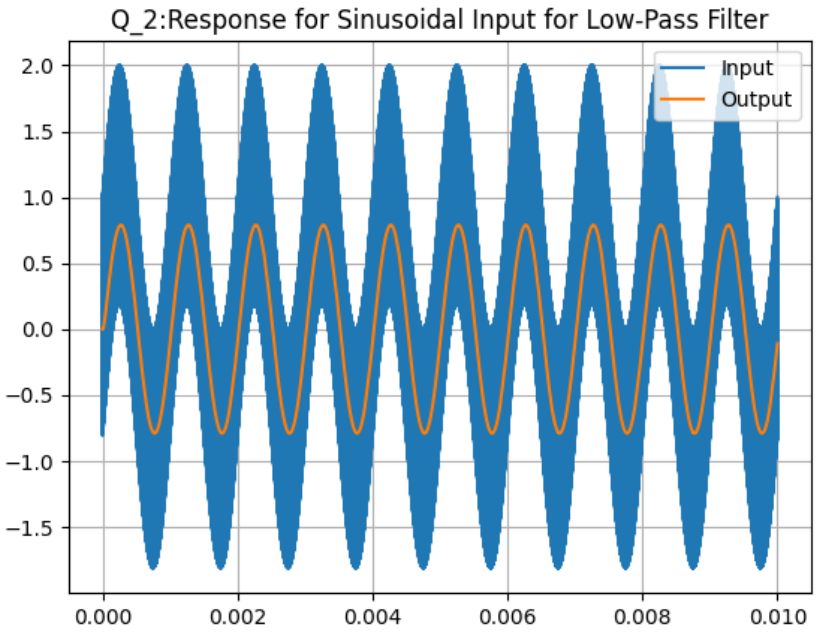

Figure 3: System Response of Low Pass Filter with Decay = 0.5.

```
# Question_2

Vi2 = np.sin(2*(10**3)*(np.pi)*t)+np.cos(2*(10**6)*(np.pi)*t)
t,Vo2,svec = sp.lsim(H1,Vi2,t)

py.figure(1)
py.plot(t,Vi2,label = 'Input')
py.plot(t,-Vo2,label = 'Output')
py.legend(loc = 'upper right')
py.title(r'Q_2:Response for Sinusoidal Input for Low-Pass Filter')
py.grid(True)
py.savefig("suhas/figure1.png")
py.show()
```

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

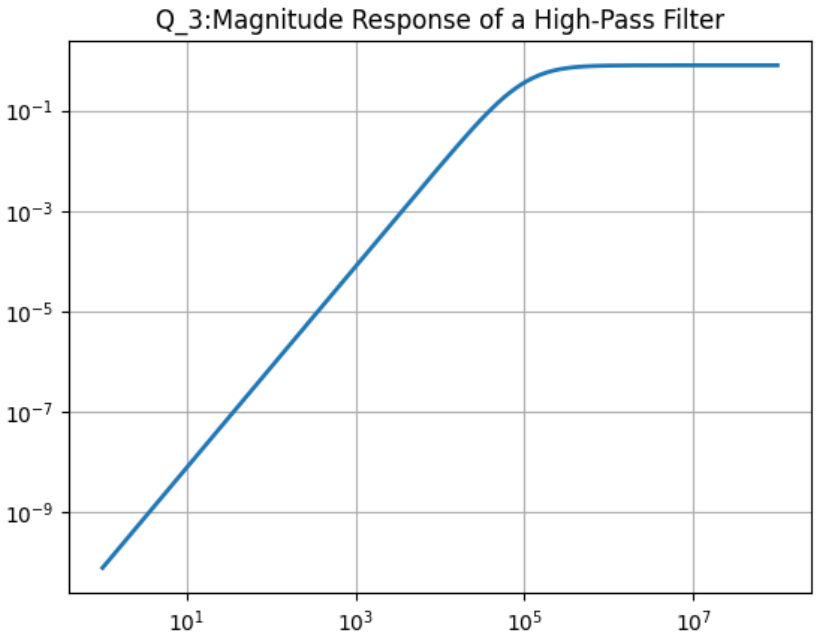

Figure 4: Output for sum of sinusoids

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:

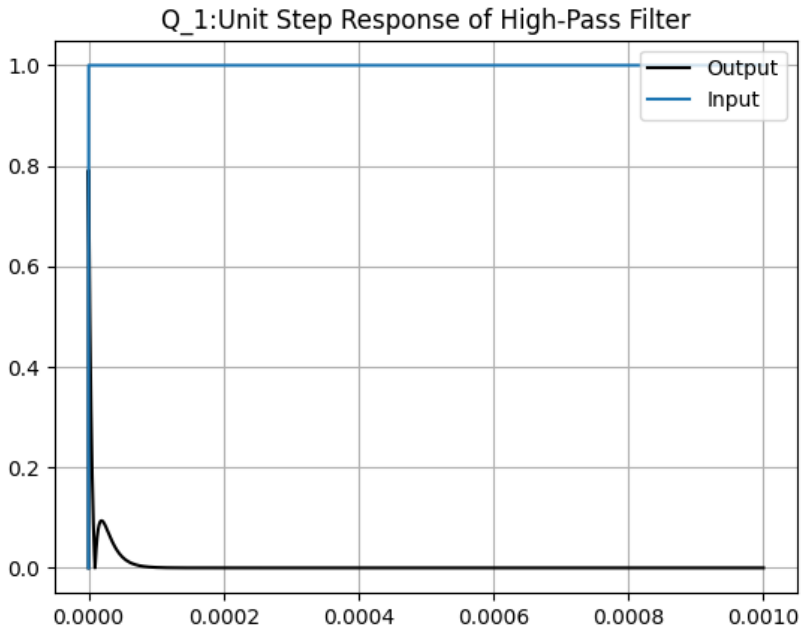

Figure 5: High frequency damped sinusoid

Thus, the high-pass behaviour of the circuit is verified. The change in the exponential would only affect the rate at which the sinusoid amplitude decays to zero. The python code snippet to execute the above is as shown:

```
# Question_3

A3,b3,V3 = highpass(10000,10000,1e-9,1e-9,1.586,1)
Vo3 = V3[3]
w  = py.logspace(0,8,801)
ss = 1j*w
hf = lambdify(s,Vo3,'numpy')
v  = hf(ss)

py.figure(2)
py.loglog(w,abs(v),lw=2,label = 'Magnitude Response of an High-Pass Filter')
py.title(r'Q_3:Magnitude Response of a High-Pass Filter')
py.grid(True)
py.savefig("suhas/figure2.png")
py.show()
```

```
# Question_4

t4 = np.linspace(0,1,100000)
Vi4 = (np.cos(2*(np.pi)*(10**4)*t4))*(np.exp(-5*t4))

Nr_H4,Dr_H4 = Symbols_to_Polynomial(Vo3)
H4 = sp.lti(Nr_H4,Dr_H4)

t4,Vo4,svec = sp.lsim(H4,Vi4,t4)

py.figure(3)
py.plot(t4,Vi4,label = 'Input freq = 10^4Hz')
py.plot(t4,Vo4,label = 'Output')
py.legend(loc = 'upper right')
py.title(r'Q_2:Response for Damped-Sinusoidal Input for High-Pass Filter')
py.grid(True)
py.savefig("suhas/figure3.png")
py.show()
```

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 shows:-

```
# Question_5

A5,b5,V5 = highpass(10000,10000,1e-9,1e-9,1.586,1/s)
Vo5 = V5[3]
t5  = np.linspace(0,0.001,50001)

Vi5 = np.ones(50001)
Vi5[0] = 0
t5,Vo5,svec = sp.lsim(H4,Vi5,t5)

py.figure(4)
py.plot(t5,abs(Vo5),color='black',label = 'Output')
py.plot(t5,Vi5,label = 'Input')
py.legend(loc = 'upper right')
py.title(r'Q_1:Unit Step Response of High-Pass Filter')
py.grid(True)
py.savefig("suhas/figure4.png")
py.show()
```

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

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 ana- lyzing complicated systems like the active

lters in this assignment.