EXPERIMENT 7
CIRCUIT ANALYSIS USING SYMPY AND LAPLACE TRANSFORMS

Abhishek Sekar
EE18B067

March 17,2020

Abstract

The goal of this assignment is the following:

- To analyze Filters using Laplace Transform.
- Utilising python for symbolic Algebra.
- Plotting graphs to get a visual understanding of 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 a low pass function and solves the matrix equation to get the V matrix is shown below:

```
#lowpass filter function definition
def lowpass(R1=10000,R2=10000,C1=10**(-9),C2=10**(-9),G=1.586,Vi=1):
#function to solve for the output voltage given the parameters
    s=sympy.symbols('s') #treats s as a symbol in the below equations
    A=sympy.Matrix([[0,0,1,-1/G],[-1/(1+s*R2*C2),1,0,0],[0,-G,G,1],[(-1/R1)-(1
    b=sympy.Matrix([0,0,0,-Vi/R1])
    V=A.inv()*b #solution is in s-space
    return A,b,V
```

The bode magnitude plot of the transfer function is shown on the next page.

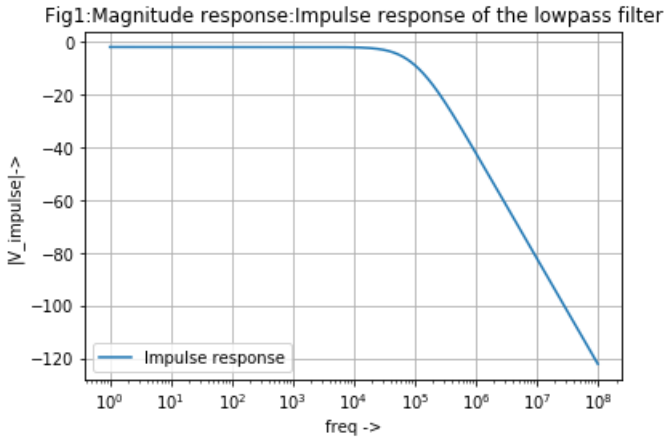

Figure 1: Magnitude bode plot of the low pass filter

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 a high pass function and solves the matrix equation to get the V matrix is shown below:

```
#high pass filter function definition
def highpass(R1,R3,C1,C2,G,Vi):
    s=  sympy.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/
    b = Matrix([0,0,0,-Vi*s*C1])
    V = A.inv()*b
    return A,b,V
```

The bode magnitude plot of the transfer function is shown on the next page.

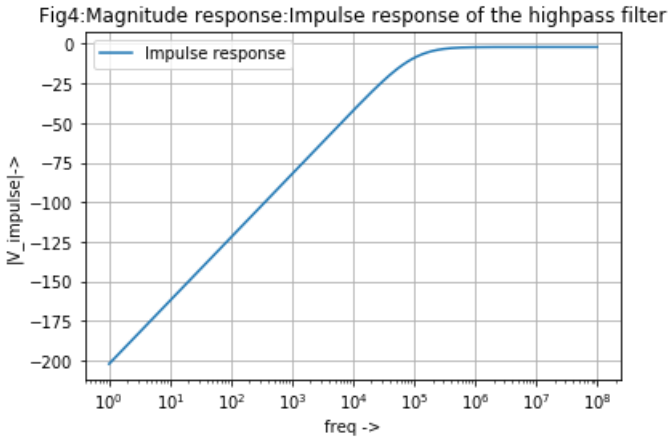

Figure 2: Magnitude bode plot of the high pass filter

Format Change of the Sympy Functions

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

```
The function with detailed description
is given below.
    def laplace_converter(V):
    Vout=sympy.expand(sympy.simplify(V))
    #simplify simplifies the expression
    ie x^2+x becomes x(x+1)
    #expand :expands the simplified expression to
    a polynomial notation like x^2+x
    n,d=sympy.fraction(Vout)
    #writes separately as num and denom
    n,d=sympy.Poly(n,s),sympy.Poly(d,s)
    # expresses n and d as polynomials of s
    num,denom=n.all_coeffs(),d.all_coeffs()
    #extracts the coefficients
    num,denom= [float(f) for f in num], [float(f) for f in denom]
    return num,denom
```

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.*

```
# The below code snippet will calculate and plot the step response for the l
A1,b1,V1 = lowpass(10000,10000,1e-9,1e-9,1.586,1/s)
Vstep = V1[3]
Hstep = laplace_converter(Vstep)
t,vstep = sp.impulse(Hstep,None,np.linspace(0,5e-3,10000))

# The plot for step response of the lowpass filter.
plt.plot(t,vstep,label='Step response')
plt.title("Fig2:Step Response of the lowpass filter")
plt.xlabel('t ->')
plt.ylabel('Vstep ->')
```

```
plt.grid()
plt.legend(loc='best')
plt.show()
```

The plot for the step response of the low pass circuit is shown below.

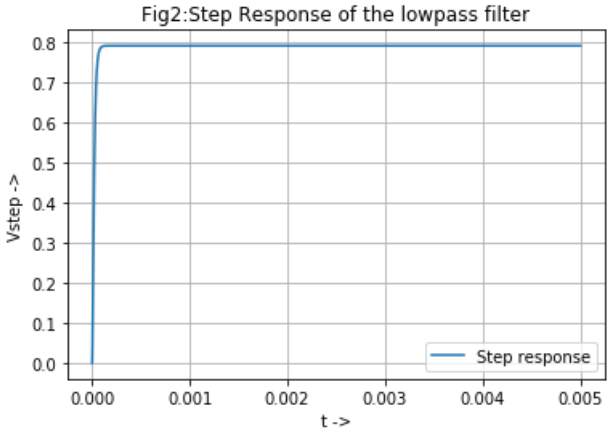

Figure 3: Step response of low pass filter.

Response to a 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 using sp.lsim ,as shown in the code snippet below.

```
def inputfn(t):   #function for the input
    return (np.sin(2000*np.pi*t)+np.cos(2e6*np.pi*t))

# The below code snippet will calculate and plot the transfer function for t
s = symbols('s')
A,b,V = lowpass(10000,10000,1e-9,1e-9,1.586,1)
Vimp = V[3]#output is in third element of V (its in terms of s)
H = laplace_converter(Vimp)
ww = np.logspace(0,8,801)
ss = 1j*ww
hf = lambdify(s,Vo,'numpy')#like a short function definition for replacing s
vimp = hf(ss)

print('The transfer function of the lowpass filter is:',Vimp)


# The response is also calculated and plotted for sum of sinusoids.
t,Vsum,svec = sp.lsim(H,inputfn(t),t)

# The plot for output response for sum of sinusoids of the lowpass filter.
plt.plot(t,Vsum,label='output response')
plt.title("Fig3:Output response for sum of solonoids of the lowpass filter")
plt.xlabel('t ->')
plt.ylabel('Vout ->')
plt.grid()
plt.legend(loc='best')
plt.show()
```

The output response to the sum of sinusoids for the low pass filter is shown below.

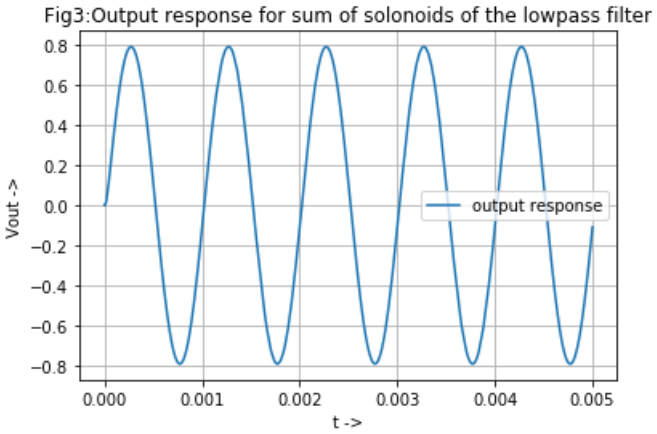

Figure 4: Output response to the sum of sinusoids.

Response to Damped Sinusoids

In this case we assign the input voltage in the form of a damped sinusoid like,
Low frequency,

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

High frequency,

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

The high frequency and low frequency plots of the input damped sinusoids are shown in the next page:

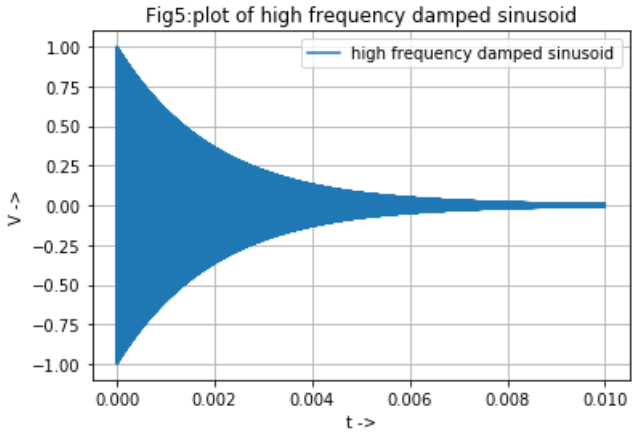

Figure 5: High frequency damped sinusoid

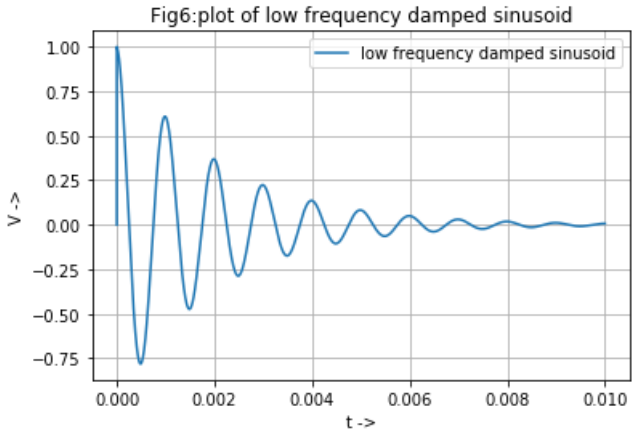

Figure 6: Low frequency damped sinusoid

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

```
def damped(t,freq=2e6,decay=500):  #default is for high freq function
    return np.cos(freq*np.pi*t)*np.exp(-decay*t) * (t>0)

# The below snippet will calculate and the transfer function for the highpas
A2,b2,V2 = highpass(10000,10000,1e-9,1e-9,1.586,1)
V_imp = V2[3]
H2 = laplace_converter(V_imp)
hf2 = lambdify(s,V_imp,'numpy')
v_imp = hf2(ss)
# The output response is calculated and plotted when the input is a damped s
# High frequency.
t2 = np.linspace(0,1e-2,1e5)
t2,Vhf,svec = sp.lsim(H2,damped(t2),t2)

# The plot for high frequency damped sinusoids.
plt.plot(t2,damped(t2),label='high frequency damped sinusoid')
plt.title("Fig5:plot of high frequency damped sinusoid")
plt.xlabel('t ->')
plt.ylabel('V ->')
plt.grid()
plt.legend(loc='best')
plt.show()

# The plot for high frequency damped sinusoid response from highpass filter.
plt.plot(t2,Vhf,label='high frequency damped sinusoid response')
plt.title("Fig7:plot of high frequency damped sinusoid response from highpas
plt.xlabel('t ->')
plt.ylabel('Vout ->')
plt.grid()
plt.legend(loc='best')
plt.show()


```

```
# Low frequency.
t3 = np.linspace(0,1e-2,1e5)
t3,Vlf,svec = sp.lsim(H2,damped(t3,2e3),t3)

# The plot for low frequency damped sinusoids.
plt.plot(t3,damped(t3,2e3),label='low frequency damped sinusoid')
plt.title("Fig6:plot of low frequency damped sinusoid")
plt.xlabel('t ->')
plt.ylabel('V ->')
plt.grid()
plt.legend(loc='best')
plt.show()


# The plot for low frequency damped sinusoid response from  highpass filter.
plt.plot(t2,Vlf,label='low frequency damped sinusoid response')
plt.title("Fig8:plot of low frequency damped sinusoid response from highpass
plt.xlabel('t ->')
plt.ylabel('Vout ->')
plt.grid()
plt.legend(loc='best')
plt.show()
```

The output responses for both cases in a high pass filter is shown in the next page.

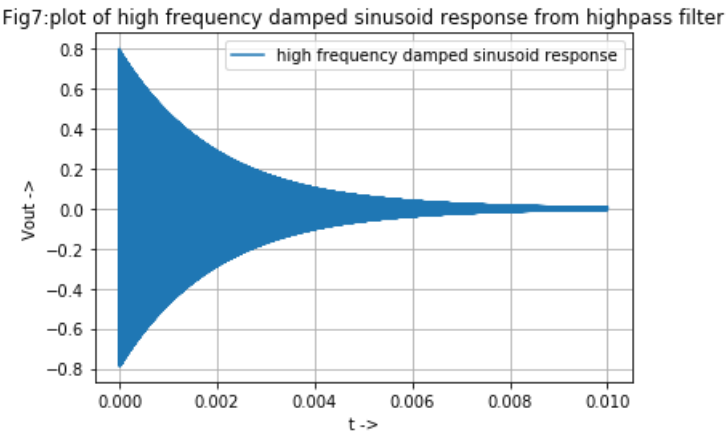

Figure 7: Output response of a high frequency damped sinusoid

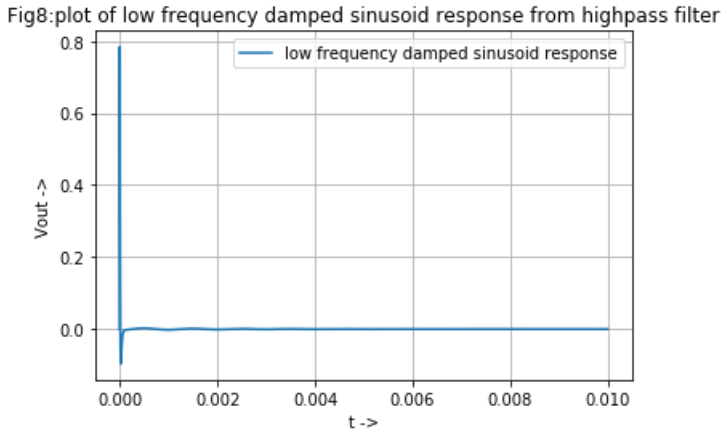

Figure 8: Output response of a low frequency damped sinusoid

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.*

```
# The step response is calculated for a highpass filter.
A3,b3,V3 = highpass(10000,10000,1e-9,1e-9,1.586,1/s)
V_step = V3[3]
H_step = laplace_converter(V_step)
t1,v_step = sp.impulse(H_step,None,np.linspace(0,5e-3,10000))

# The plot for step response of the highpass filter.
plt.plot(t1,v_step,label='Step response')
plt.title("Fig9:Step Response of the highpass filter")
plt.xlabel('t ->')
plt.ylabel('Vstep ->')
plt.grid()
plt.legend(loc='best')
plt.show()
```

The plot of the step response for a high pass filter is shown in the next page.

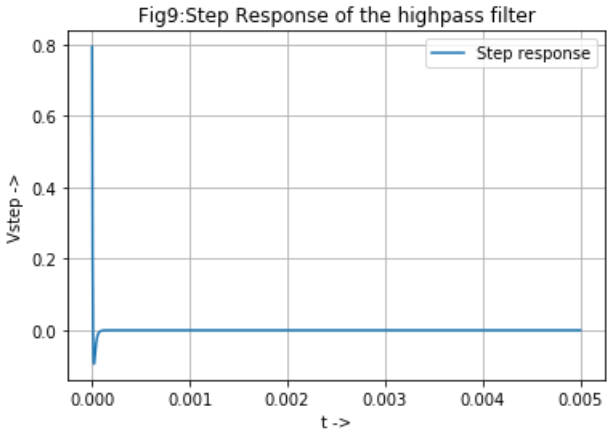

Figure 9: Step response for a high pass filter

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.

Results

Output obtained from the code:

*The transfer function of the lowpass filter is: -0.0001586/((1.0e-5*s + 1)*(-2.0e-9*s + 1.586e-9*s/(1.0e-5*s + 1) - 0.0004 + 0.0002/(1.0e-5*s + 1)))*

*The transfer function of the highpass filter is: 1.586e-14*s**2/((1.0e-5*s + 1)*(-2.0e-14*s**2/(1.0e-5*s + 1) + 4.0e-9*s - 1.586e-9*s/(1.0e-5*s + 1) + 0.0002))*

Conclusion

The sympy module has given us a simple and novel way to analyse complicated circuits by analytically solving their node equations. We then interpreted the solutions by plotting time domain responses using the signals toolbox introduced in the previous experiment. Thus, sympy combined with the scipy.signal module is a very useful toolbox for analyzing complicated systems like the active filters in this assignment.