EE2703: Applied Programming Lab
Assignment 7
Circuit analysis using Sympy

V. Ruban Vishnu Pandian
EE19B138

April 25, 2021

Contents

1 Aim:

The aim of this assignment is to:

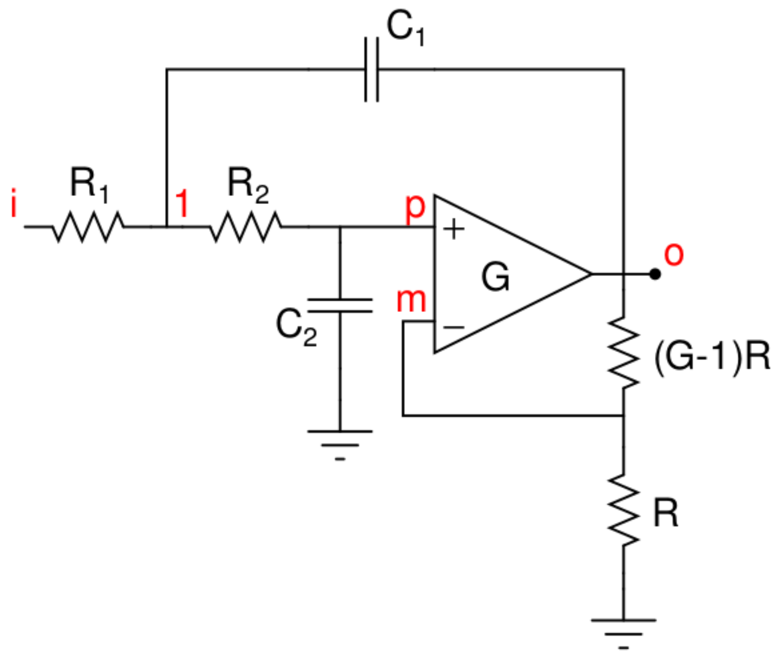
1. Get started with some basic Sympy (Symbolic python) functions.
2. Solve for the output response of given active RC filters using Sympy and Scipy.signal toolbox.
3. Analyse the frequency components in the output responses.

2 Theory: Active RC filters:

Active RC filters are electronic devices which are purely made using active operational amplifiers (OpAmps), resistors and capacitors. These filters help us to filter out certain frequency components present in an electric signal which may not be desirable for a particular application. They are used in various applications such as noise reduction, radio tuning, selective amplification, etc. In this assignment, two types of active RC filters are analysed: Low pass filter and high pass filter.

2.1 Low pass filter:

A low pass filter (LPF) is an electrical filter which allows only the low frequency components of an electric signal and attenuates the high frequency components. The electronic circuit used to realise this filter is shown below:



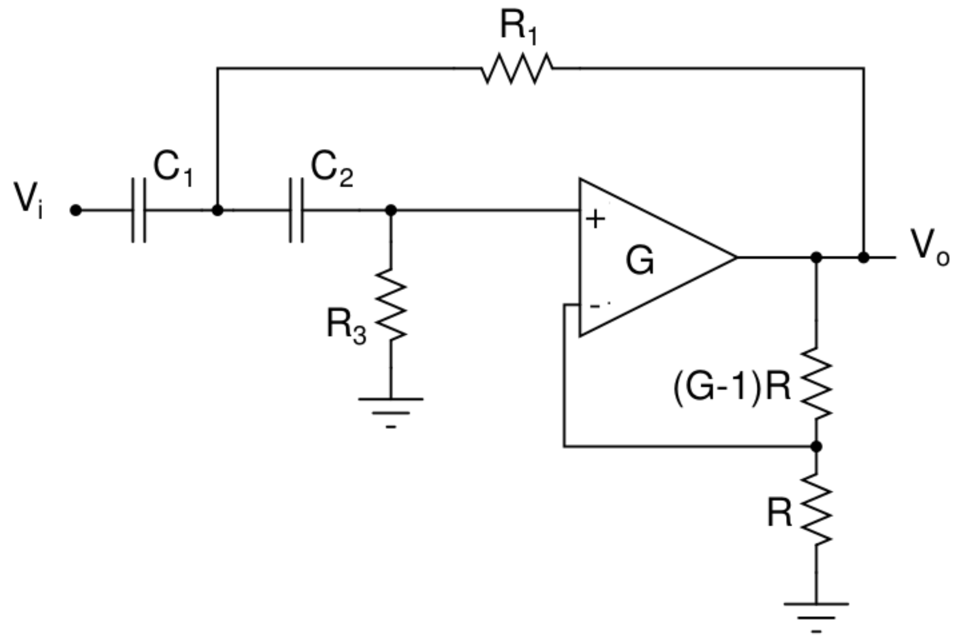
We need to solve for the transfer function $H(s) = \frac{V_o(s)}{V_i(s)}$. To do this, we need to solve for the node voltages using nodal analysis. Also, we know that in the laplace domain, a capacitor acts like a resistor with its resistance $R = \frac{1}{sC}$. With these things in mind, the nodal analysis equation in matrix form will be:

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

Clearly, the source vector on the RHS contains only one non-zero term dependent on V_i. Hence, all the other node voltages would be some multiple of it. We need the multiple relating V_o and that multiple is $H(s)$.

2.2 High pass filter:

A high pass filter (HPF) is an electrical filter which allows only the high frequency components of an electric signal and attenuates the low frequency components. The electronic circuit used to realise this filter is shown below:



The circuit is almost similar to the LPF circuit except that the resistors are replaced by capacitors and vice-versa (Except for the feedback resistors). Also, there isn't much change in the structure of the nodal matrix equation. With the understanding that whatever was initially a resistor is now replaced by a capacitor and vice-versa, we can apply the following transform:

$$R \to \frac{1}{sC}$$

$$C \to \frac{1}{sR}$$

With these changes in mind, the new matrix equation becomes:

$$\begin{bmatrix} 0 & 0 & 1 & \frac{-1}{G} \\ -\frac{sC_2R_3}{1+sC_2R_3} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ -sC_1 - sC_2 - \frac{1}{R_1} & sC_2 & 0 & \frac{1}{R_1} \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -sC_1V_i(s) \end{bmatrix}$$

Again the output response V_o will be some multiple of V_i and that multiple is the transfer function $H(s)$.

2.3 Solving for the transfer function:

As we have seen, the matrix equation is not purely based on numbers. It is also based on mathematical variables. In our case, the variable is 's'. Unlike normal matrix equations, we can't solve this using packages such as numpy. We need to treat the variable 's' as an unknown entity and solve for the transfer function as a function of 's'. To do this, we need the Sympy library of python. Sympy is a special library which allows us to treat certain variables as mathematical/symbolic variables. These variables are different from the usual python variables. They necessarily need not store values but rather store symbols. These symbols don't have any value but can be treated like the variables we use in math (x,y,z) and complex expressions could be formed using these symbols. Finally the value of that expression can be found by providing these symbols some numerical values.

Sympy is very useful when it comes to obtain the expression from primitive inputs. In our filter case too, the primitive inputs are the resistor and capacitor values. However, the final expression is not easily derivable and hence, we can't write a user-defined python function without knowing the final expression. Sympy allows us to find the final expression while still treating the variable 's' as a mathematical variable.

```
import sympy as sp
s = sp.symbols('s')

def LPF(R1,R2,C1,C2,G,Vi):
    A = sp.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]])
    #In the original code, indendation is proper. Here, the line is brought
    #down due to space constraints

    b = sp.Matrix([0,0,0,-Vi/R1])
    V = A.inv()*b; TF = V[3]
    return TF
```

For example, in the python function provided above, the inputs are the resistor, capacitor and Opamp gain values. However, the final expression is not directly used to find the transfer function. Instead, the sympy variable 's' is used to denote the variable and the matrix equation is solved. Finally, the expression is obtained and returned out.

3 Functions defined in the code:

3.1 Function to generate the LPF transfer function:

```
def LPF(R1,R2,C1,C2,G,Vi):
    A = sp.Matrix([[0,0,1,-1/G],[-1/(1+(s*R2*C2)),1,0,0],[0,-G,G,1],[-(1/R1)-(1/
    b = sp.Matrix([0,0,0,-Vi/R1])
    V = A.inv()*b; TF = V[3]
    return TF
```

This function returns the transfer function of an LPF.

3.2 Function to generate the HPF transfer function:

```
def HPF(R1,R3,C1,C2,G,Vi):
    A = sp.Matrix([[0,0,1,-1/G],[-s*C2*R3/(1+(s*C2*R3)),1,0,0],[0,-G,G,1],[-(s*C
    b = sp.Matrix([0,0,0,-Vi*s*C1])
    V = A.inv()*b; TF = V[3]
    return TF
```

This function returns the transfer function of a HPF.

3.3 Function to convert transfer function expression to signal toolbox LTI form:

```
def SymtoTrans(TF):
    n,d = TF.as_numer_denom()
    num = n.as_poly(s).all_coeffs()
    den = d.as_poly(s).all_coeffs()
    num = [float(i) for i in num]
    den = [float(i) for i in den]
    H = sig.lti(num,den)
    return H
```

This function takes a transfer function expression as input, converts it to the signal toolbox LTI form and returns it.

3.4 Function to generate the magnitude response of a transfer function:

```
def magresponse(TF,w):
    ss = 1j*w
    tf = sp.lambdify(s,TF,"numpy")
    return abs(tf(ss))
```

This function takes a transfer function and the frequencies vector as an input and returns the magnitude response of the transfer function for the provided frequencies.

3.5 Function to generate the response of sum of sinusoids:

```
def sumofsins_response(w1,w2,t,H):
    x = np.sin(w1*t)+np.cos(w2*t)
    vo = sig.lsim(H,x,t)[1]
    return vo
```

This function takes the sinusoid frequencies, time vector and signal toolbox LTI form of a transfer function, generates the output response of a sum of sinusoids input and returns it.

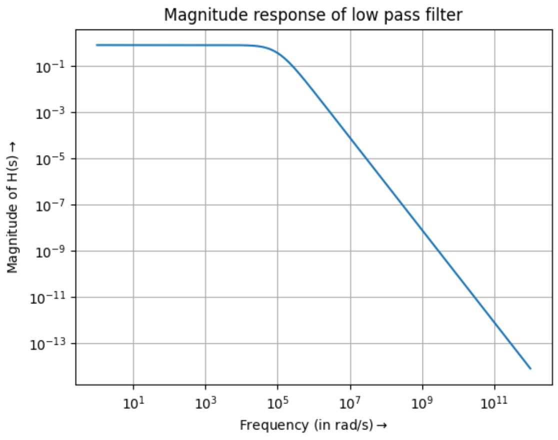
3.6 Function to generate the response of damped sinusoid:

```
def dampedsin_response(wd,a,t,H):
    x = np.cos(wd*t)*np.exp(-a*t)
    vo = sig.lsim(H,x,t)[1]
    return vo
```

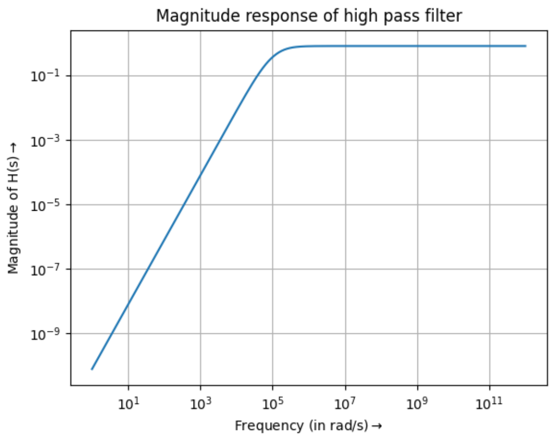
This function takes the sinusoid frequency, decay constant, time vector and signal toolbox LTI form of a transfer function, generates the output response of a damped sinusoid input and returns it.

4 Observations and plots:

4.1 Magnitude response plots:

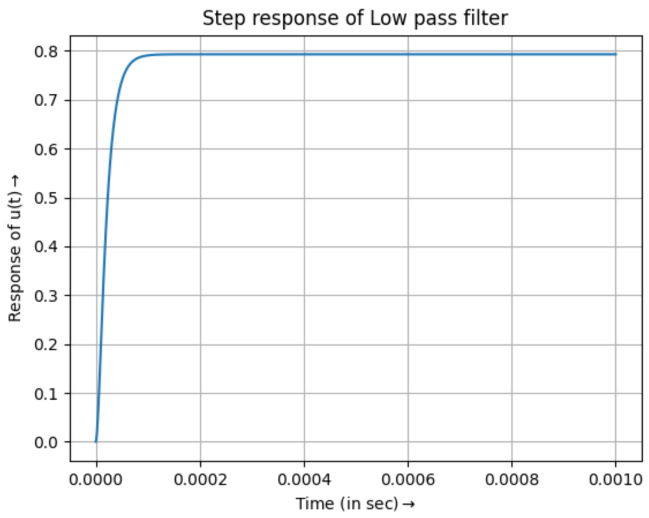


The LPF we have seems to be a single pole LPF with pole frequency=10^5rad/s

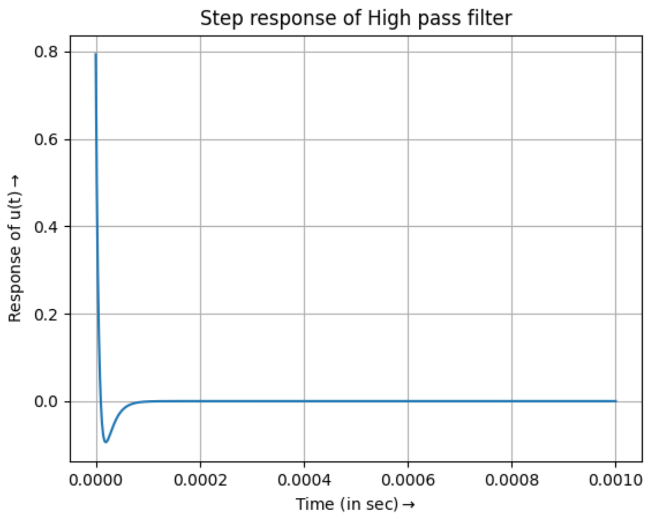


The HPF we have seems to be a DC zero-Single pole HPF with pole frequency=10^5rad/s

4.2 Step response plots:

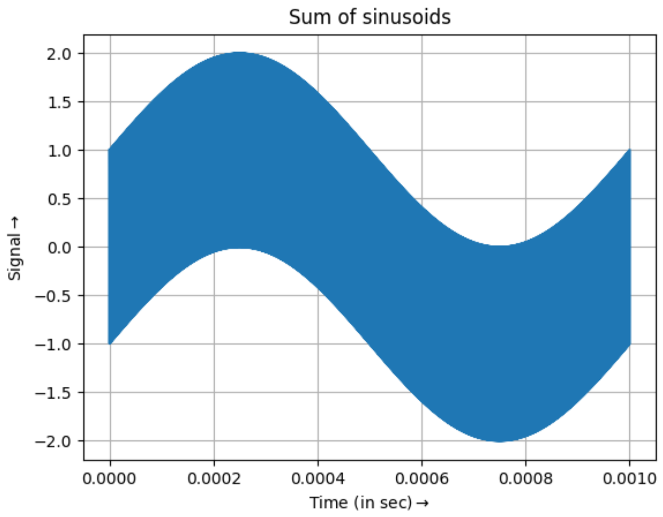


The DC gain of the LPF seems to be 0.8. Hence, the step response is looking like a DC signal with value 0.8



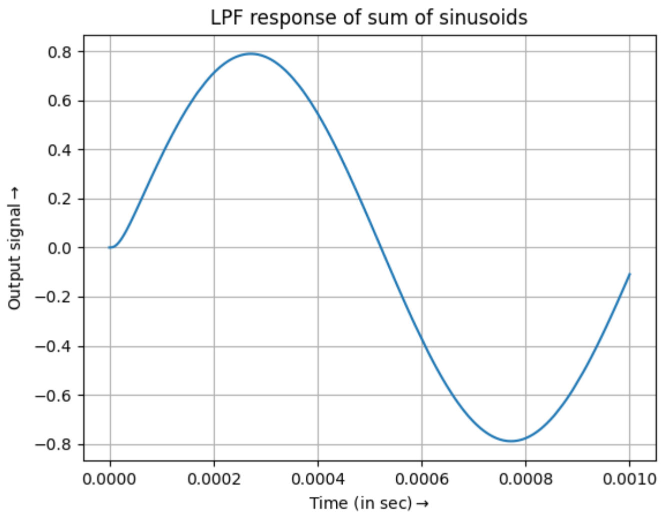
As expected, the step response of the HPF is a DC signal but with value very close to zero

4.3 Sum of sinusoids signal:

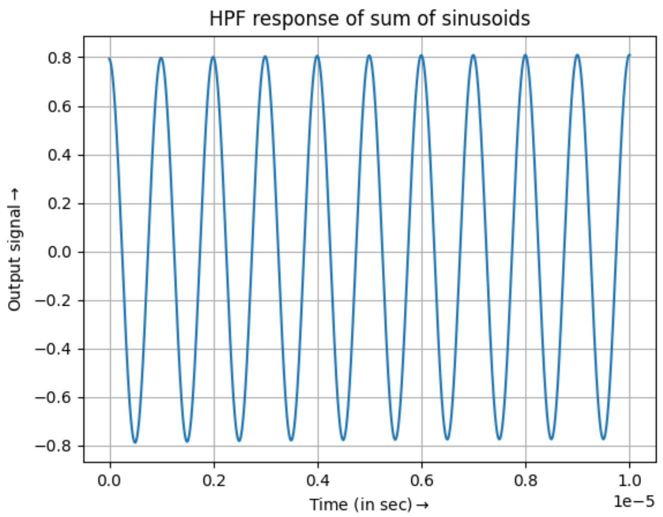


This is a signal which is made up of two sinusoids of frequencies,
$f_1 = 1kHz$,$f_2 = 1MHz$

4.4 Response of sum of sinusoids:

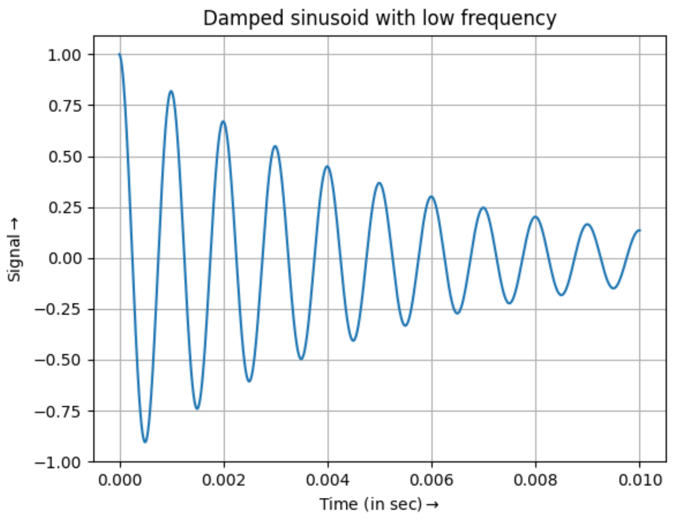


The output response of the sum of sinusoids, when passed through an LPF purely contains only the f=1kHz component

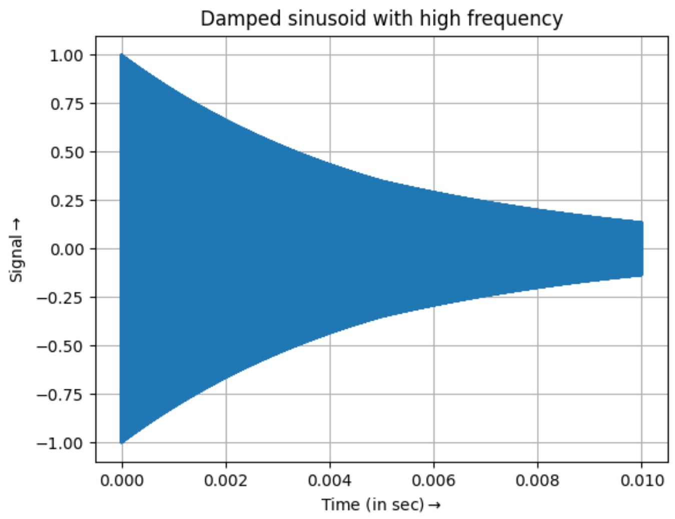


The output response of the sum of sinusoids, when passed through a HPF purely contains only the f=1MHz component

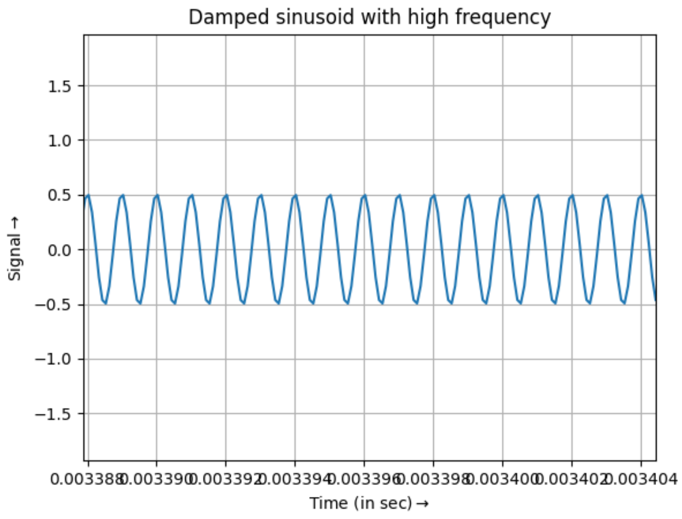
4.5 Damped sinusoid signals:



This is a damped sinusoid signal with f=1kHz and decay constant=$200s^{-1}$

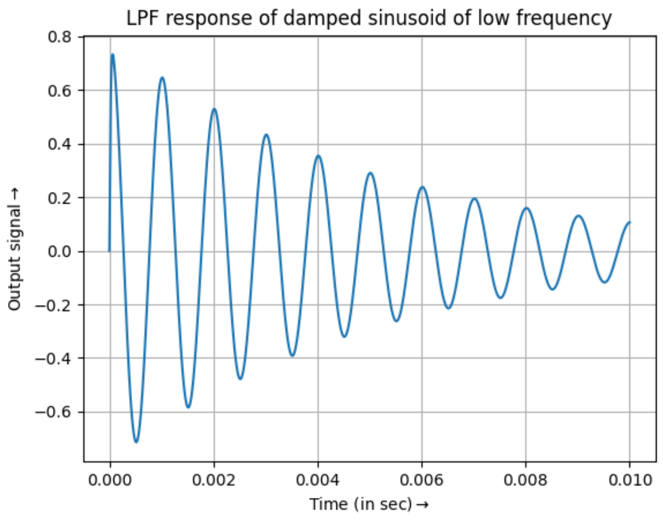


This is a damped sinusoid with f=1MHz and decay constant=$200s^{-1}$

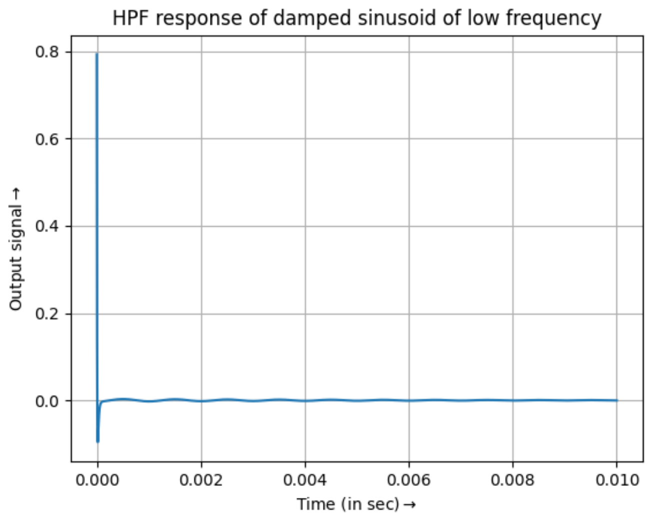


On zooming in, we can see the sinusoidal variation of the high frequency sinusoid. There isn't much variation in the dampening component in this time scale

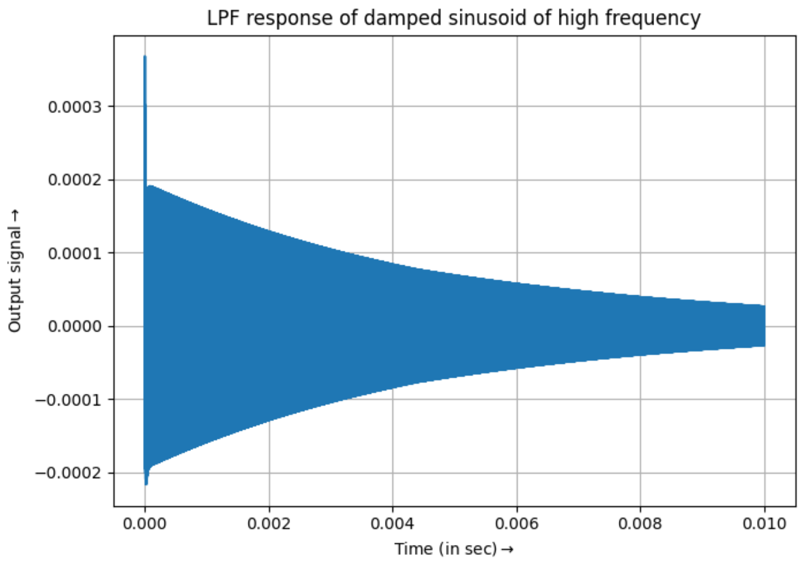
4.6 Response of damped sinusoid:



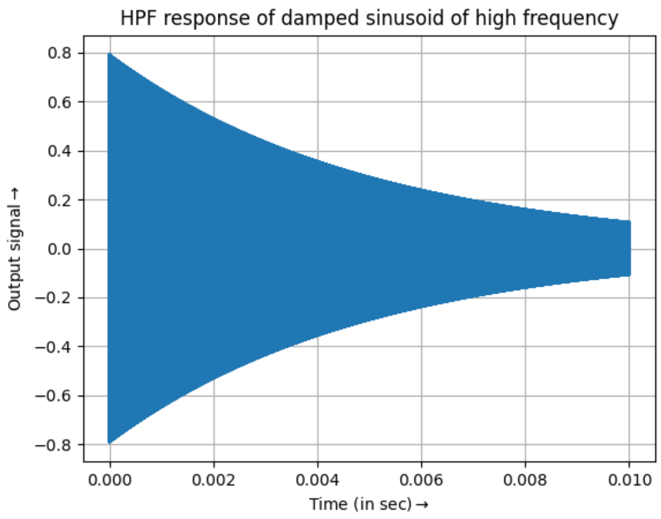
When the low frequency damped sinusoid is passed through an LPF, the output isn't much different from the input. The amplitude has reduced to 0.8



When the low frequency damped sinusoid is passed through a HPF, the output is highly attenuated as seen from the image. The line at t=0 is due to initial conditions



When the high frequency damped sinusoid is passed through an LPF, the output is highly attenuated as seen from the image (y-axis scale is very low denoting the attenuation)



When high frequency damped sinusoid is passed through a HPF, the output isn't much different from the input. The amplitude has reduced to 0.8

5 Conclusions:

1. Both the filters have a pole frequency of 10^5 rad/s and a constant gain of 0.8 in their operating region. This is evident from their magnitude response plots.
2. The step response of the low pass filter starts from zero (due to initial conditions) and stabilises at a DC value of 0.8.
3. The step response of the high pass filter starts from 0.8 (due to initial conditions) and stabilises at a very low value close to zero.
4. When the sum of sinusoids signal is passed through the LPF, the 1kHz component gets filtered out with a gain of 0.8.
5. When the sum of sinusoids signal is passed through the HPF, the 1MHz component gets filtered out with a gain of 0.8.
6. When the low frequency damped sinusoid is passed through the LPF, the output is not so much different from the input.
7. When the low frequency damped sinusoid is passed through the HPF, the output is highly attenuated. It starts from 0.8 due to initial conditions.
8. When the high frequency damped sinusoid is passed through the LPF, the output is highly attenuated (Evident from the y-axis scale)
9. When the high frequency damped sinusoid is passed through the HPF, the output is not so much different from the input.