EE2703:
APPLIED PROGRAMMING LAB

Assignment 7

Circuit Analysis Using Sympy

Arjun R
EE20B016

April 8, 2022

Contents

1 Introduction

In this assignment, we will be analysing second-order filter circuits using *Laplace transforms*. We will be taking the help of Sympy, Python's powerful library for symbolic mathematics, and scipy's signal processing toolbox.

```
import sympy as sy
import scipy.signal as sp
```

Specifically, we will be working with **3dB Butterworth filters**. Butterworth filter is a type of signal processing filter which gives a flat as possible frequency response in the passband.

2 Low-Pass Filter

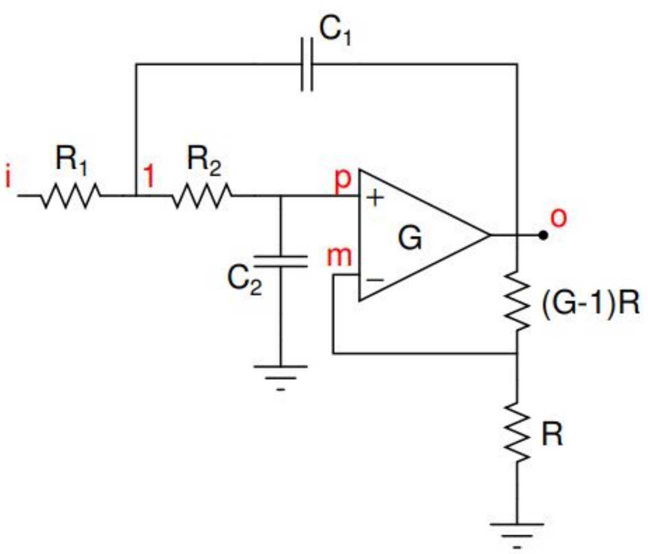

Figure 1: Low-Pass Filter Circuit

The system tranfer function can be obtained by solving the *Modified Nodal Analysis* matrix equation:

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

The code implementation using sympy is

```
def lowpass(R1, R2, C1, C2, G, Vi):
    s = sy.symbols('s')
    A = sy.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 = sy.Matrix([0,0,0,-Vi/R1])
    V = A.inv()*b
    return (A,b,V)

A,b,V = lowpass(10000,10000,1e-9,1e-9,1.586,1)
s = sy.symbols('s')
Vo_LP = V[3]
print(sy.simplify(Vo_LP))
ww = p.logspace(0, 8, 801)
ss = 1j*ww
hf = sy.lambdify(s, Vo_LP, 'numpy')
p.loglog(ww, abs(hf(ss)), lw=2)
p.grid(True)
p.title('Low-Pass Filter Magnitude plot')
p.xlabel(r'$\omega\rightarrow$')
p.show()
```

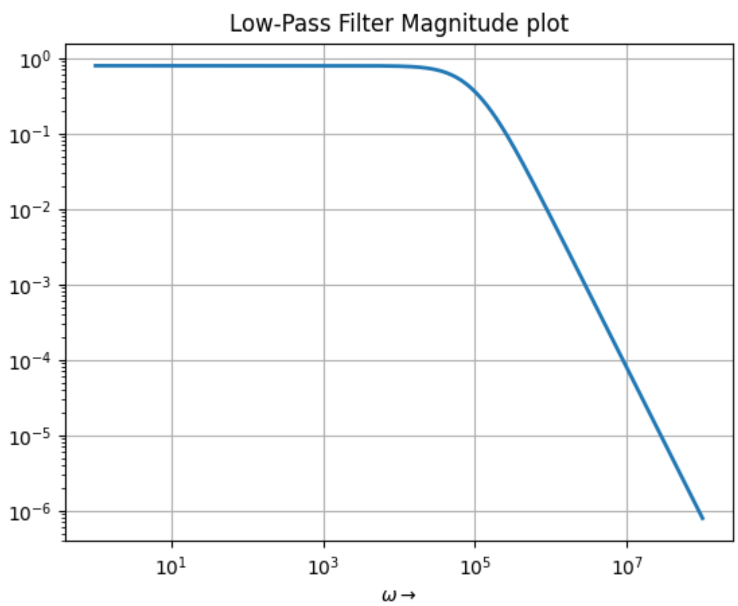

Figure 2: Low-Pass Filter Magnitude Plot

3 High-Pass Filter

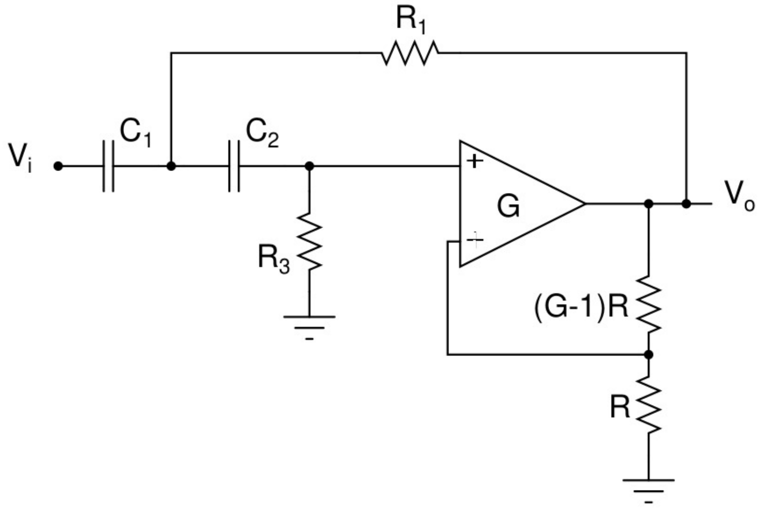

Figure 3: High-Pass Filter Circuit

The system tranfer function can be obtained by solving the *Modified Nodal Analysis* matrix equation:

$$\begin{pmatrix} 0 & 1 & 0 & -\frac{1}{G} \\ -\frac{-sR_3C_2}{1+sR_3C_2} & 0 & 1 & 0 \\ 0 & G & -G & 1 \\ sC_1 + sC_2 + \frac{1}{R_1} & 0 & -sC_2 & -\frac{1}{R_1} \end{pmatrix} \begin{pmatrix} V_1 \\ V_m \\ V_p \\ V_o \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ sC_1V_i(s) \end{pmatrix}$$

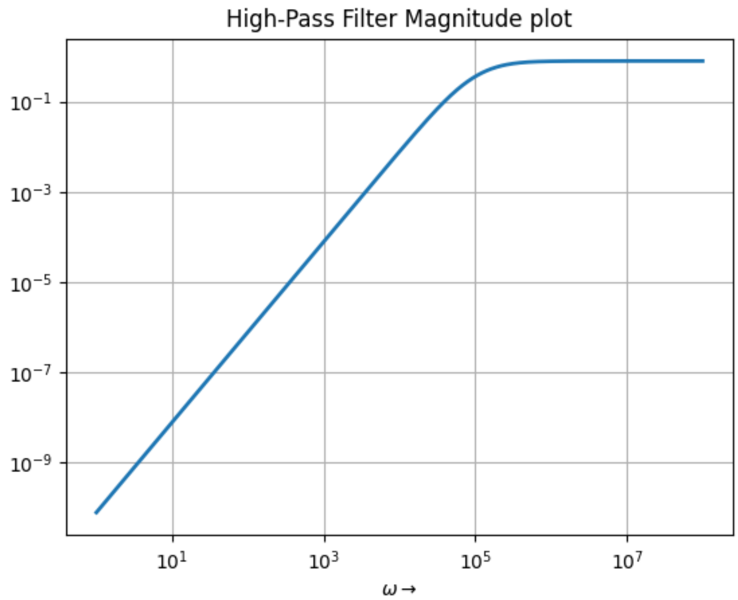

Figure 4: High-Pass Filter Magnitude Plot

4 User-Defined Python Functions

To properly extract the functionality of `sympy`, I have used two functions to facilitate conversion of sympy expressions to to transfer functions of Linear Time-Invariant (LTI) systems.

```
def sympy_to_transferfn(xpr, s=sy.symbols('s')):
""" Takes in Sympy transfer function polynomial
    Returns Scipy LTI system expression's Nr and Dr """
num, den = sy.simplify(xpr).as_numer_denom()
# simplify() reduces the expression, as_numer_denom() returns a/b as a, b
p_num_den = sy.Poly(num, s), sy.Poly(den, s)  # numer and denom polynomials
c_num_den = [p.all_coeffs() for p in p_num_den]  # coefficients of the s-polynomial
l_num, l_den = [sy.lambdify((), c)() for c in c_num_den]  # convert to floats
return l_num, l_den

def sympy_to_lti(xpr, s=sy.symbols('s')):
N, D = sympy_to_transferfn(xpr)
return sp.lti(N, D)
```

5 Problems

5.1 P1. Step Response

The step response can be generated by multiplying $1/s$ to the system tranfer function since:

$$\mathcal{L}^{-1}\{u(t)\} = \frac{1}{s}$$

I have used a Python function `stepresponse` to do this:

```
def stepresponse(H, t):
N, D = sympy_to_transferfn(H)
D.append(0) # Multiplying H with 1/s
H_u = sp.lti(N, D)
t, y = sp.impulse(H_u, None, t)
return y
```

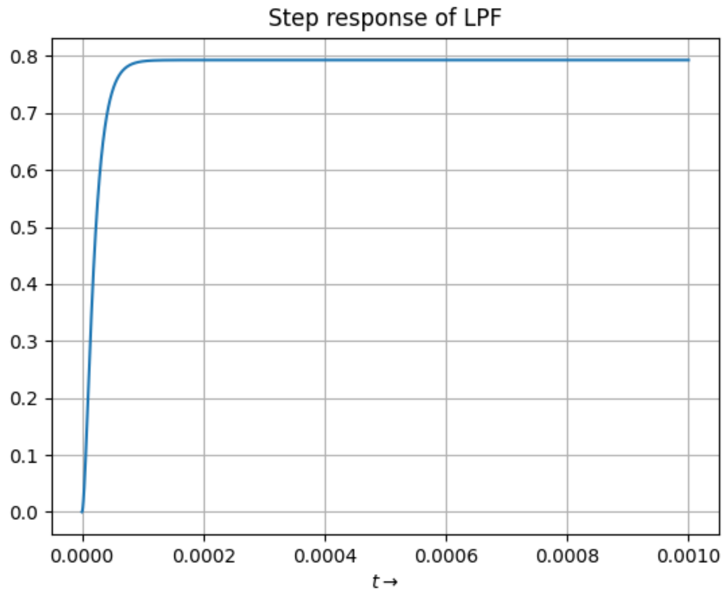

Figure 5: LPF Step Response

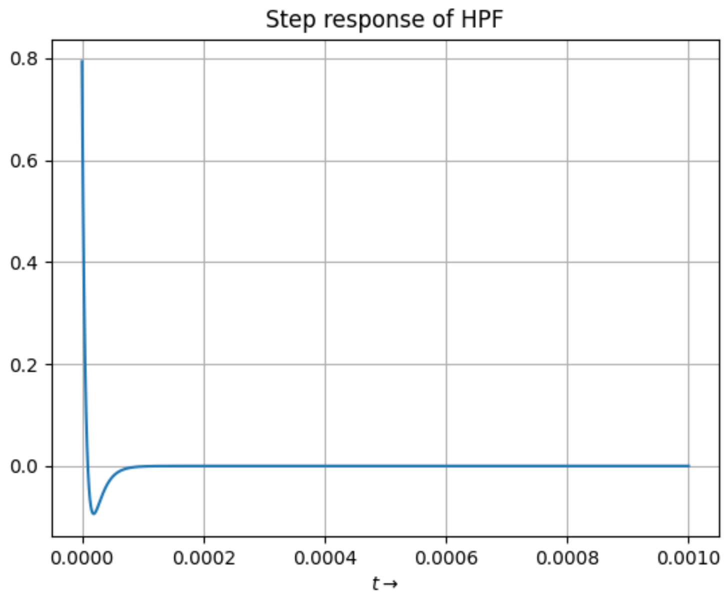

Figure 6: HPF Step Response

As expected, the unit step response has a large value at $t = 0s$ due to the abrupt change in input. At steady state, the high-pass filter filters out the lower frequency input (unit step input has frequency = 0 Hz) and thus the voltage output goes to zero.

5.2 P2. Sinusoidal Response

We are given the following input signal:

$$V_i(t) = (sin(2000\pi t) + cos(2x10^6\pi t))u_o(t)V$$

We are asked to calculate the output voltage $V_o(t)$. The function `sp.lsim` simulates the output of a continuous-time linear system and we can use it for our purpose.

```
LP_Vout_sum = sp.lsim(sympy_to_lti(Vo_LP), Vi, t)[1]
p.plot(t, LP_Vout_sum)
p.grid(True)
p.title('LPF O/P for sum of sinusoids')
p.xlabel(r'$t\rightarrow$')
p.show()
HP_Vout_sum = sp.lsim(sympy_to_lti(Vo_HP), Vi, t)[1]
p.plot(t, HP_Vout_sum)
p.grid(True)
p.title('HPF O/P for sum of sinusoids')
p.xlabel(r'$t\rightarrow$')
p.show()
```

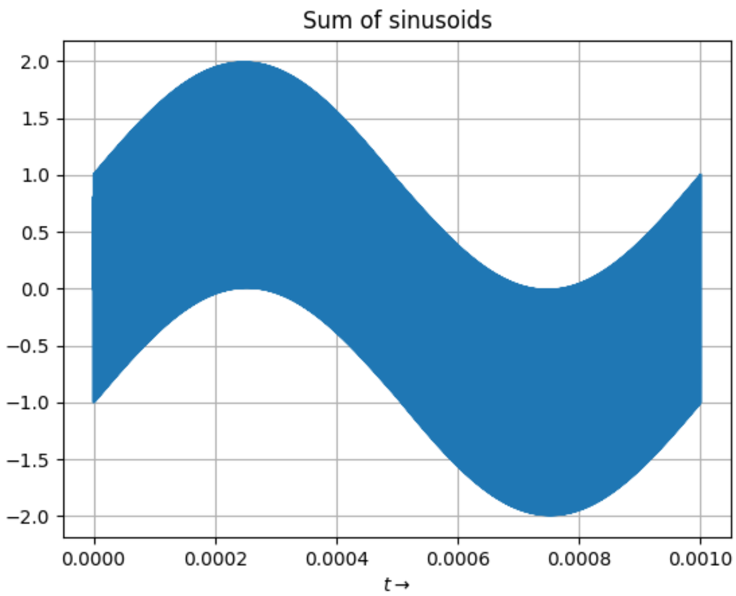

Figure 7: Input- Summation of Sinusoids

5.3 Response for Damped Sinsoids

We will take two damped sinusoids for proper analysis of each filter's response:

$$x_1(t) = e^{-50000}cos(10000000t)$$

$$x_2(t) = e^{-10}cos(1000t)$$

As we can see, one is a high frequency damped sinusoid and the other is a low frequency damped sinusoid. Similar to the last case, we can use the function `sp.lsim` to get the output. Sample code to get output for the high frequency damped sinusoid is as follows:

```
def damped_input(t, w0, decay):
    return p.cos(w0*t)*p.exp(-1*decay*t)
```

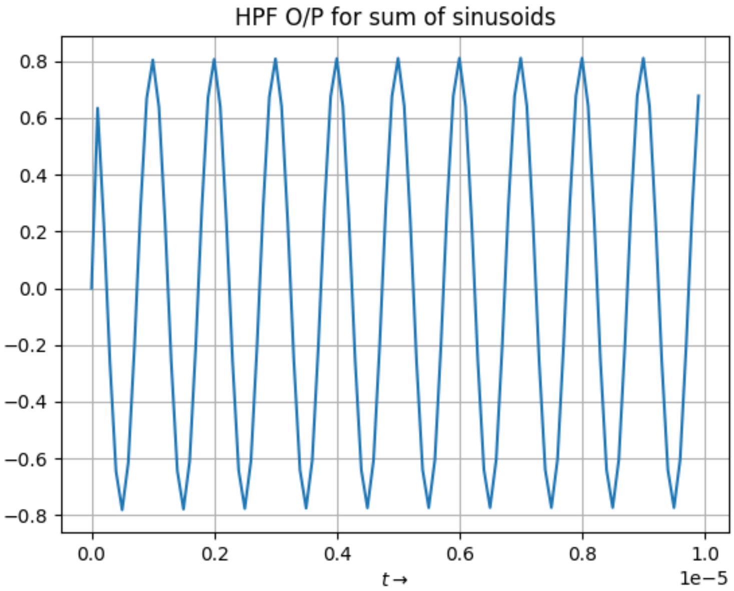

Figure 8: High-Pass Filter Response

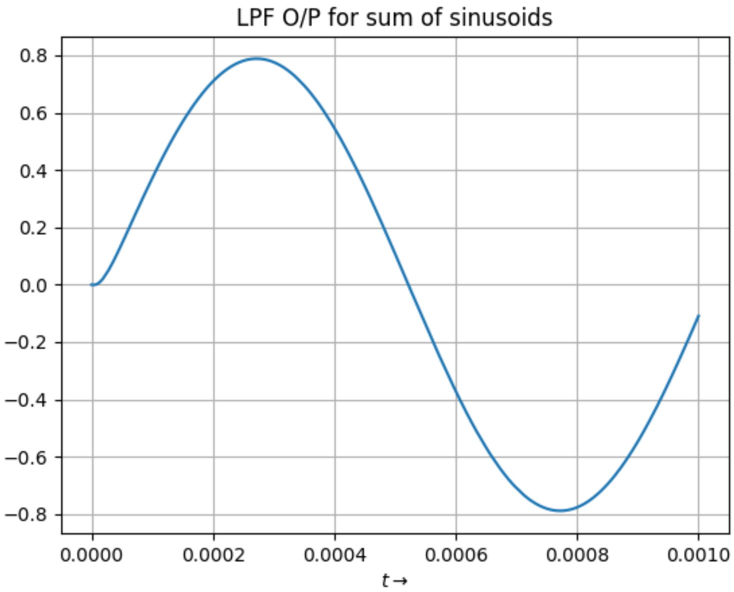

Figure 9: Low-Pass Filter Response

```
# Damped HF sinusoid parameters
t = p.linspace(0, 1e-3, 10000)
w0 = 1e7
decay = 5e4
V_damped_HF = damped_input(t, w0, decay)
p.plot(t[:1000], V_damped_HF[:1000])
p.grid(True)
p.title('Damped high frequency sinusoidal input')
p.xlabel(r'$t\rightarrow$')
p.show()
```

For the high frequency case:

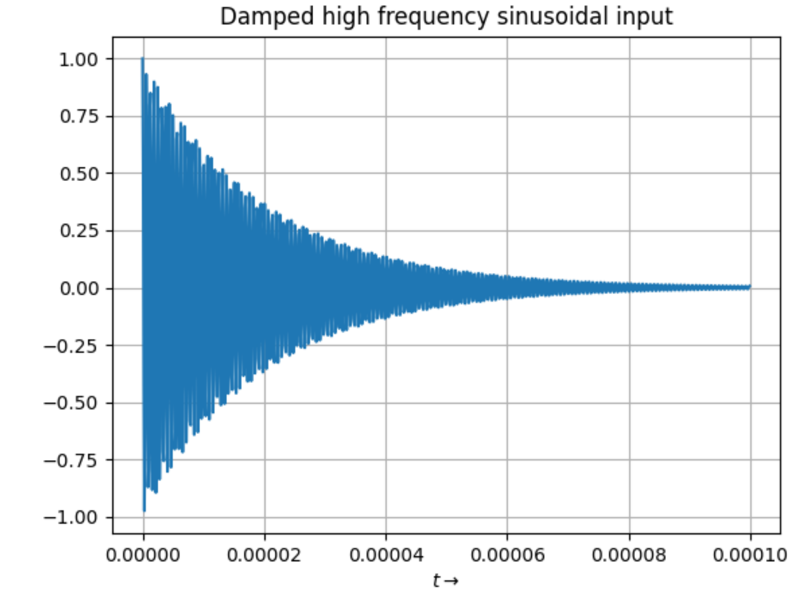

Figure 10: High Frequency(10000000 rad/s) Damped Sinusoid

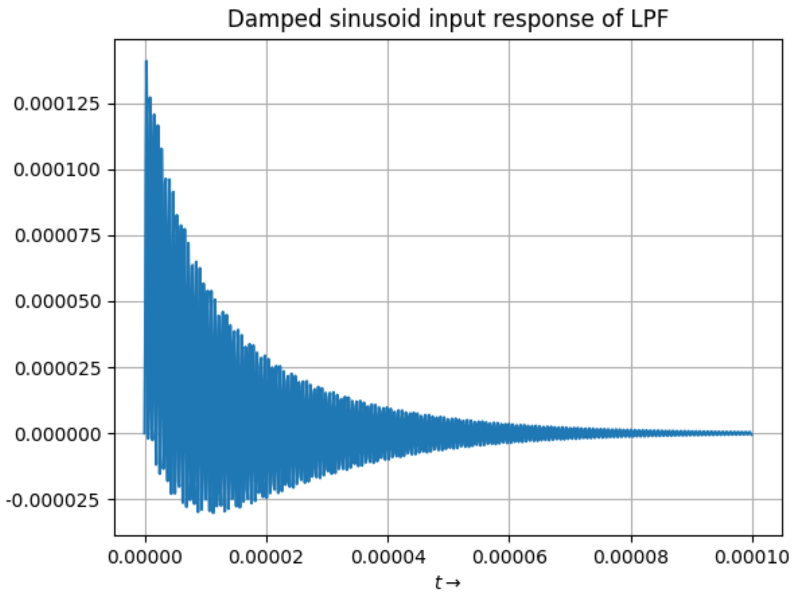

Figure 11: LPF Response to High Freq. Input

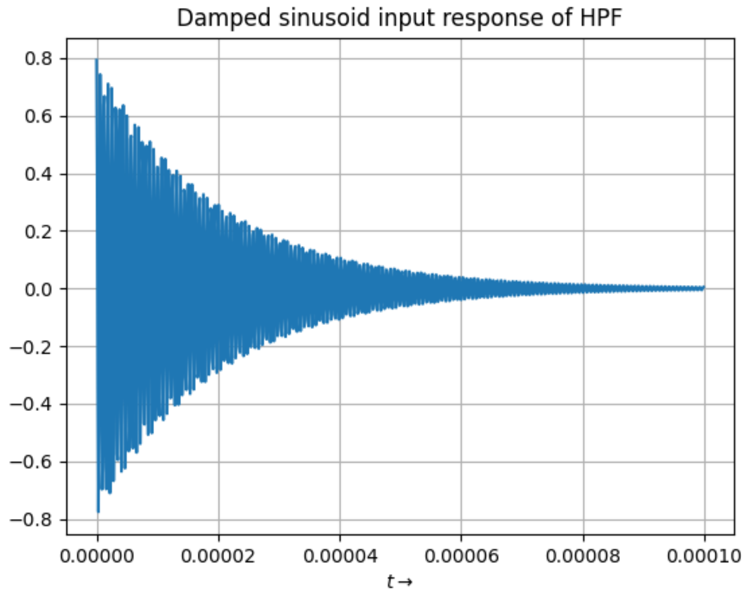

Figure 12: HPF Response to High Freq. Input

For the low frequency case:

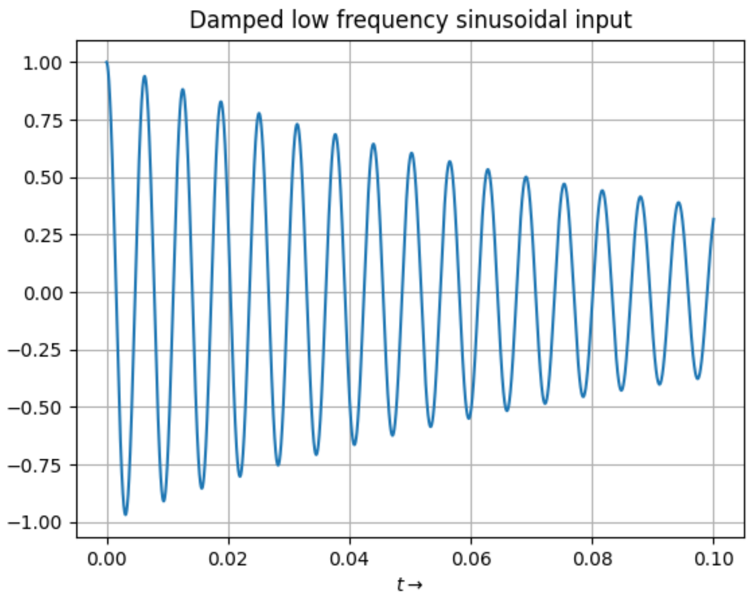

Figure 13: Low Frequency(1000 rad/s) Damped Sinusoid

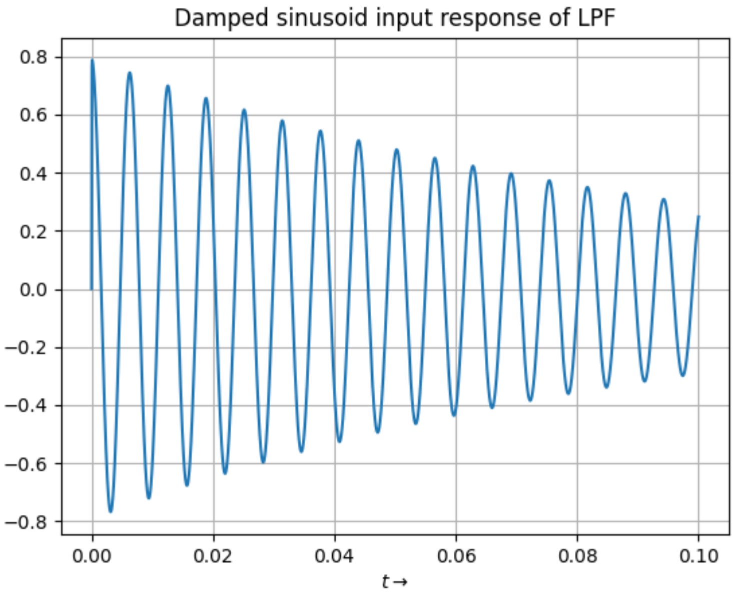

Figure 14: LPF Response to Low Freq. Input

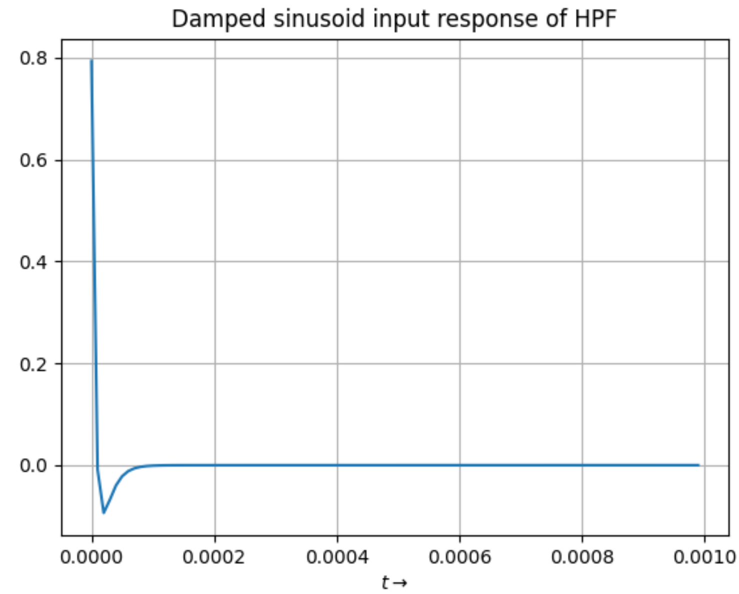

Figure 15: HPF Response to Low Freq. Input

6 Conclusion

- Through this assignment, we have established that sympy is a very powerful tool for symbolic algebra. It has helped us to easily analyze and solve complicated transfer function.
- We have seen the action of low-pass and high-pass Butterworth filters on various sinusoidal signals.