EE2703: Applied Programming Lab

Week 7: Solving Laplace Transforms Using Sympy

Author: Surya Prasad S, EE19B121

25th April, 2021

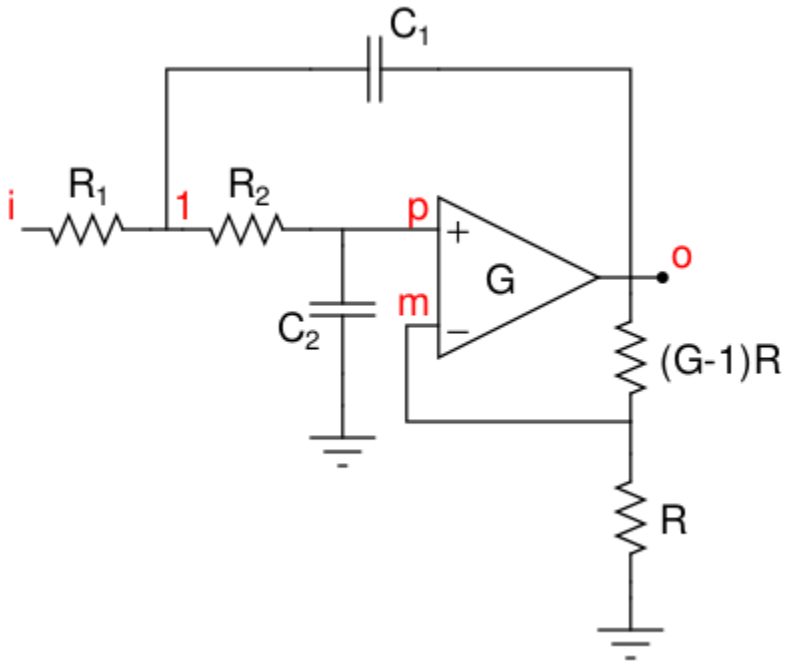
1 Abstract:

In this week's assignment we shall analyse Active Filters using Laplace Transform. We shall explore the algebraic capabilities of Symbolic algebra of Python. For solving the system we shall continue using SciPy's signal library. Here we shall only consider filters made using OpAmps and specifically the Active Lowpass Filter and Active Highpass Filter.

2 Theory:

2.1 Lowpass Filter:

First we shall analyse a Lowpass Filter.



The circuit equations are:

$$V_m = \frac{V_o}{G}$$

$$V_p = \frac{V_1}{1+j\omega R_2 C_2}$$

$$V_0 = G(V_p - V_m)$$

$$\frac{V_i - V_1}{R_1} + \frac{V_p - V_1}{R_2} + j\omega C_1(V_o - V_1) = 0$$

Solving the above equations we get the transfer function $V_o(s)/V_i(s)$

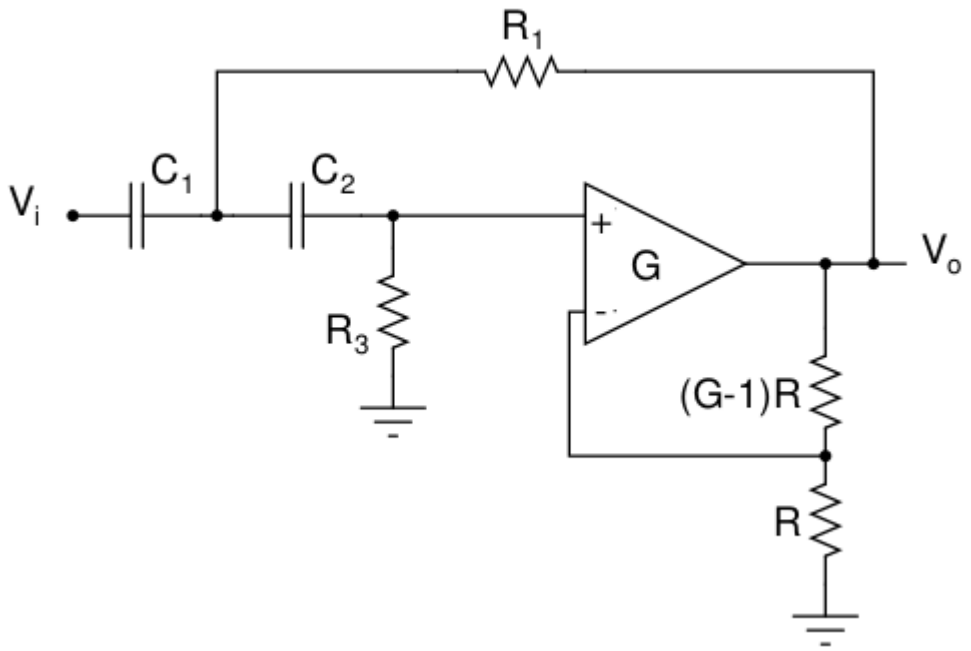
The approximate expression we get is:

$$V_o \approx \frac{V_i}{1+j\omega R_1 C_1}$$

This is assuming $R_1 = R_2$ and $C_1 = C_2$. So for high frequencies, the signal gets attenuated which is expected as the system is a Lowpass Filter.

2.2 Highpass Filter:

For our Highpass Filter we shall consider the following circuit:



The circuit equations are:

$$V_m = \frac{V_o}{G}$$

$$V_p = \frac{j\omega C_2 R_3 V_1}{1+j\omega C_2 R_3}$$

$$V_0 = G(V_p - V_m)$$

$$\frac{V_o - V_1}{R_1} + j\omega C_1(V_i - V_1) + j\omega C_2(V_p - V_1) = 0$$

Solving the above equations we get the transfer function $V_o(s)/V_i(s)$

We can approximately write this as:

$$V_o = \frac{j\omega R_1 C_1}{1+j\omega R_1 C_1} V_i$$

This is assuming $R_1 = R_2$ and $C_1 = C_2$. So for low frequencies, the signal's gain is very small and gets attenuated which is what we expect from a Highpass Filter.

3 Procedure:

3.1 Libraries Imported:

```
import scipy.signal as sp
import pylab as p
import sympy as sy
from sympy.abc import s,t
```

3.2 Common functions defined:

We shall use the following functions for both Lowpass and Highpass Filters:

1. Function to extract coefficients from symbolic representation of a transfer function and returns LTI class. Code snippet for the function:

```
def sym_to_sys(sym):
    ## First we shall split the symbolic representation into its numerator
        and denominator
    n, d = sy.fraction(sym)

    ## Then we shall extract the polynomial terms from it
    n, d = sy.Poly(n, s), sy.Poly(d, s)

    ## Now we can store the coefficients of the terms and then check if
       the given system can be solved.
    ### If valid, then they are used to make the LTI type class
    num, den = n.all_coeffs(), d.all_coeffs()

    if len(num) > len(den):
        print("Invalid system passed.")
        exit()

    H = sp.lti(p.array(num, dtype = float), p.array(den, dtype = float))
    return H
```

2. Function to plot Magnitude Response (Bode plot). Code snippet for the function:

```
def bode_analysis(H, title):
    w = p.logspace(0, 12, 1001)
    ss = 1j * w
    hf = sy.lambdify(s, H, 'numpy')
    h = hf(ss)

    ## Plotting in loglog scale (Bode plot)
    p.figure(0)
    p.loglog(w, abs(h), lw=2)
    p.title("Bode plot of the " + title)
    p.xlabel("Frequency (in rad/s)$\\rightarrow$")
    p.ylabel("Magnitude (in log scale)$\\rightarrow$")
    p.grid(True)
    p.show()
```

3. Function to plot response for a given input signal. Code snippet for the function:

```
def input_sim(H, Vi, t_range, title, type):
    x = sp.lsim(H, Vi, t_range)[1]
    p.plot(t_range, Vi, label = 'Input signal')
    p.plot(t_range, x, label = 'Output response')
    p.title(title + " input signal vs output response of " + type)
    p.xlabel("time (in s)$\\rightarrow$")
    p.ylabel("Voltage (in V)$\\rightarrow$")
    p.legend()
    p.grid(True)
    p.show()
```

4. Function to analyse a given Transfer Function through various plots. Code snippet for the function:

```
def plot_analysis(H, Vi, t_range, title):
    H_lti = sym_to_sys(H)
    bode_analysis(H, title)

    ## Computing and plotting step response
    step_response = H * 1/s
    step_response_lti = sym_to_sys(step_response)
    x = sp.impulse(step_response_lti, None, t_range)[1]
    p.figure(1)
    p.plot(t_range, x)
    p.title("Step Response of the " + title)
    p.xlabel("time (in s)$\\rightarrow$")
    p.ylabel("Response (in V)$\\rightarrow$")
    p.grid(True)
    p.show()

    ## Response to a damped low frequency signal
    p.figure(2)
    V_low_freq = p.exp(-300 * t_range) * p.cos(2 * 10**3 * p.pi * t_range)
    input_sim(H_lti, V_low_freq, t_range, "Low frequency", title)

    ## Response to a damped high frequency signal
    p.figure(3)
    V_high_freq = p.exp(-300 * t_range) * p.cos(2 * 10**6 * p.pi * t_range
        )
    input_sim(H_lti, V_high_freq, t_range, "High frequency", title)

    ## Response to the given input signal
    p.figure(4)
    input_sim(H_lti, Vi, t_range, "Given", title)
```

3.3 Lowpass Filter:

We shall first define a function to compute the Lowpass Filter's Transfer Function:

```
def lowpass_tf(R1, R2, C1, C2, G):
    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, -1/R1])
    V = A.inv() * b
    return A, b, V
```

Let's look at the plot for the Magnitude Response of the Filter.

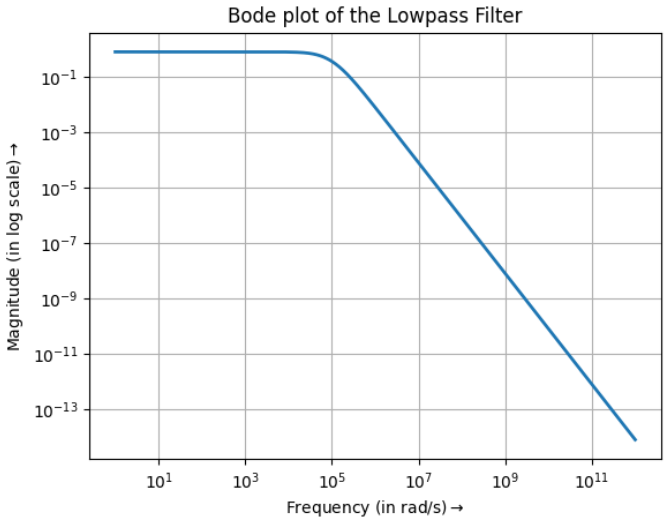


Figure 1: Magnitude Response of Lowpass Filter

So for high frequencies the output has a very low gain and gets attenuated.

Now let's look at the step response of the system.

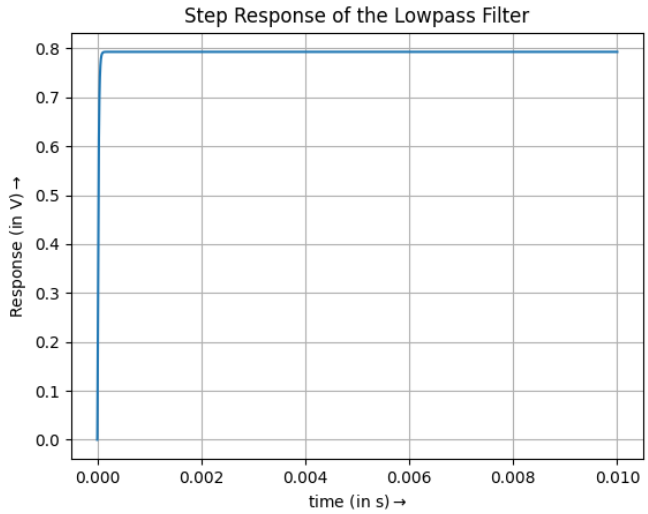


Figure 2: Step Response of Lowpass Filter

The step response of the system steadies with time giving a constant output. This is expected as Lowpass Filters allow DC power to pass through.

Let's give it a signal with different frequencies. When we pass an input with mixed frequencies, we expect the output to only consist of the lower frequencies. Let's test this out. First we shall pass a low frequency signal and see the output.

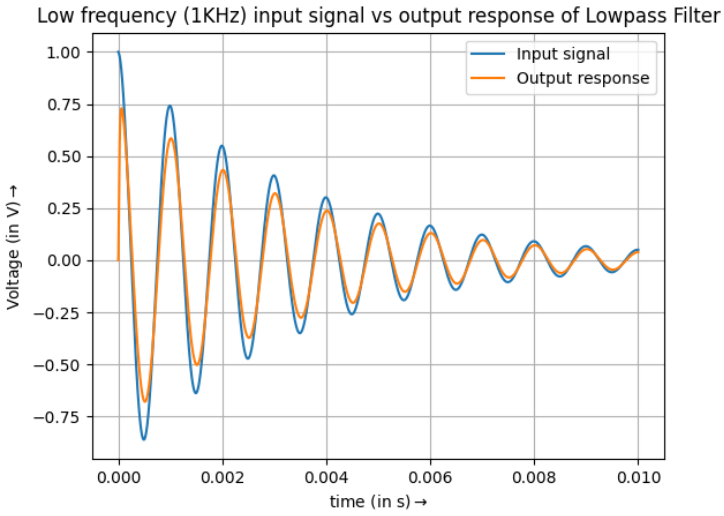


Figure 3: Output Response of Lowpass Filter for low frequency input

The frequency of the signal is $1KHz$. The output is starting from zero and has almost quickly stablised to a sinusoidal waveform. Now, let's pass a high frequency signal and see the output.

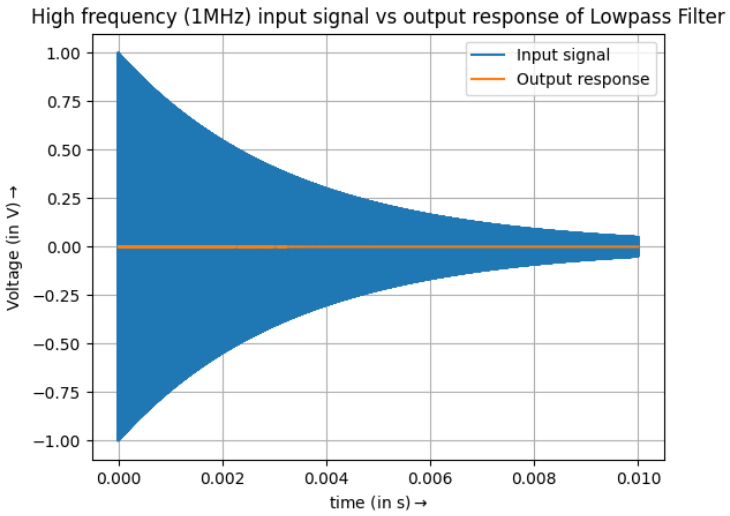


Figure 4: Output Response of Lowpass Filter for high frequency input

Here the frequency is so high ($1MHz$) that it appears to be a continous block. The signal got attenuated as expected and we get a 0 output response.

Let's pass the following given input.

$$v_i = (sin(2000\pi t) + cos(2 * 10^6 \pi t))u(t)$$

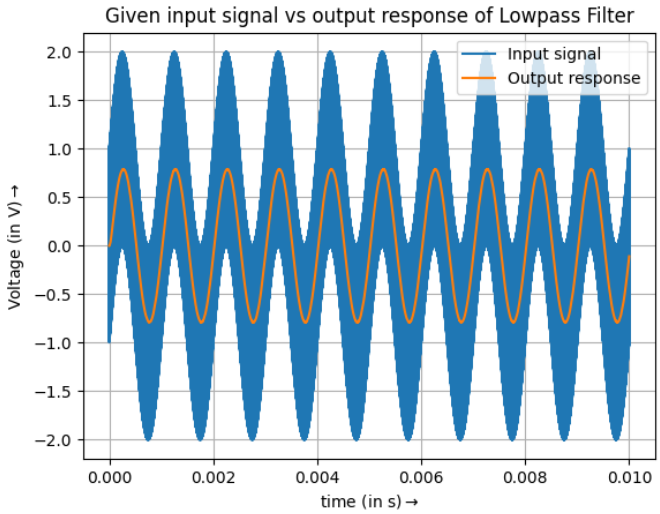


Figure 5: Output Response of Lowpass Filter for mixed frequency input

Here the input signal is a combination of two sinusoids, one with low frequency and other with high frequency. The Lowpass Filter will remove the high frequency component and the output will only contain the low frequency component. We should also note that the amplitude of the signal affects the output wave in an almost linear manner and doesn't influence our filter design.

3.4 Highpass Filter:

Here also we shall first define a function to compute the Transfer Function:

```
def highpass_tf(R1, R3, C1, C2, G):
    A = sy.Matrix([[0, -1, 0, 1/G],
        [s * C2 * R3/(s * C2 * R3 + 1), 0, -1, 0],
        [0, G, -G, 1],
        [-s * C2 -1/R1 - s * C1, 0, s * C2, 1/R1]])
    b = sy.Matrix([0, 0, 0, -s * C1])
    V = A.inv() * b
    return A, b, V
```

Let's look at the plot for the Magnitude Response of the Filter.

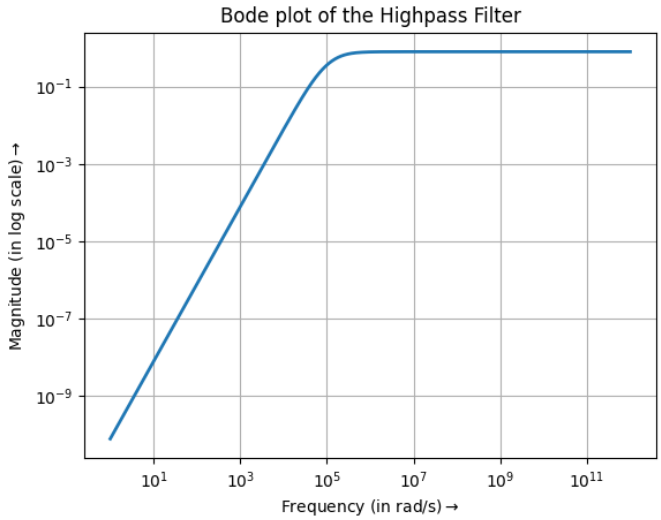


Figure 6: Magnitude Response of Highpass Filter

So here for low frequencies the output gets attenuated.

Now let's look at the step response of the system.

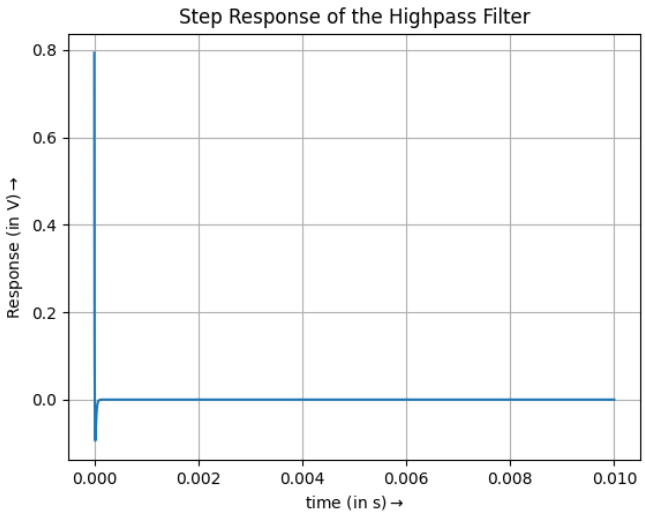


Figure 7: Step Response of Highpass Filter

There is an initial impulse response from the system but at steady state the output becomes 0.

Let's now give the Highpass Filter a signal with different frequencies. When we pass an input with mixed frequencies, only those components which have frequencies can give an output response and the lower frequency components get attenuated. We shall test this out by passing a low frequency signal and a high frequency signal and see the output response.

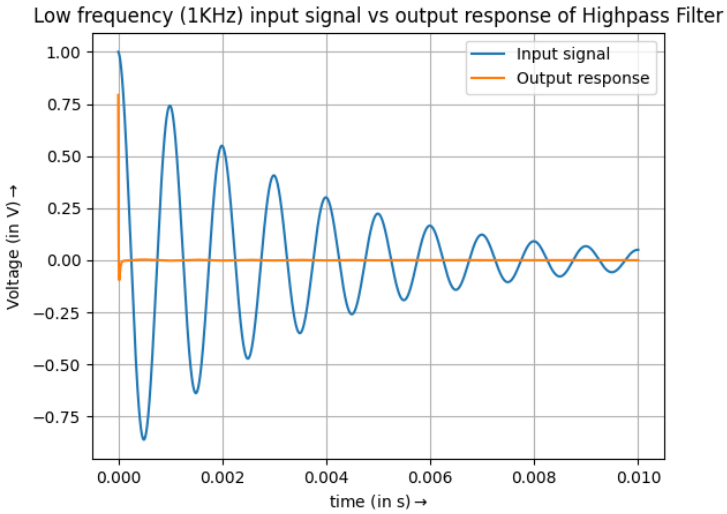


Figure 8: Output Response of Highpass Filter for low frequency input

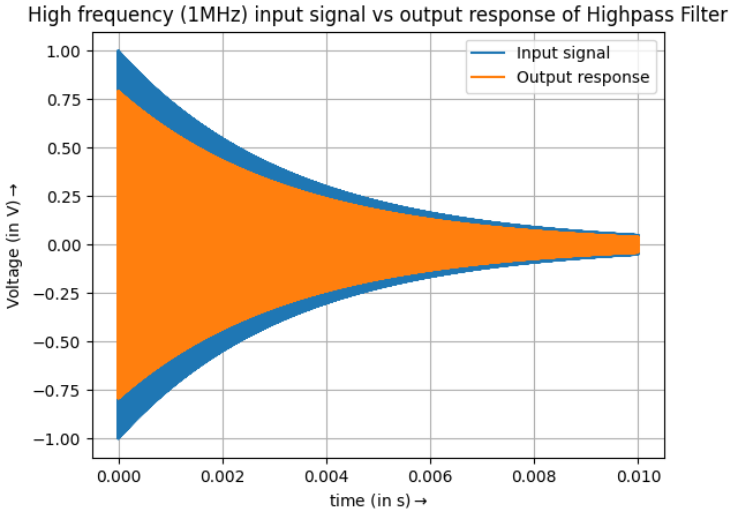


Figure 9: Output Response of Highpass Filter for high frequency input

The low frequency input got attenuated very quickly and the high frequency input had a fixed sinusoidal output which decayed with time. Here also the amplitude of the signal affects the output wave in an almost linear manner. Now let's pass the following given input.

$$v_i = exp(-300 * t) * (cos(2000\pi t) + sin(2 * 10^6 \pi t))u(t)$$

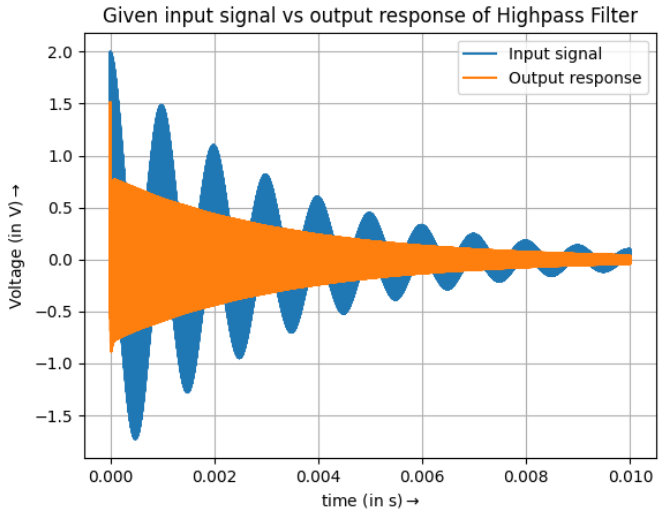


Figure 10: Output Response of Highpass Filter for the mixed frequency input

Here the input signal is again a combination of two sinusoids, one with a low frequency and other with high frequency. The low frequency component got attenuated as we can see from the graph that the signal is more like a continous block like in Fig.9.

3.5 Approximating the Transfer Functions:

The exact transfer functions of our Active filters are fairly complicated second order functions. As we can see from the Bode plots we can say that the system behaves almost like a first order given that the resistors and the capacitors are of equal value. Let's compare the Magnitude Response of the approximate Transfer Functions with the Actual Transfer Function.

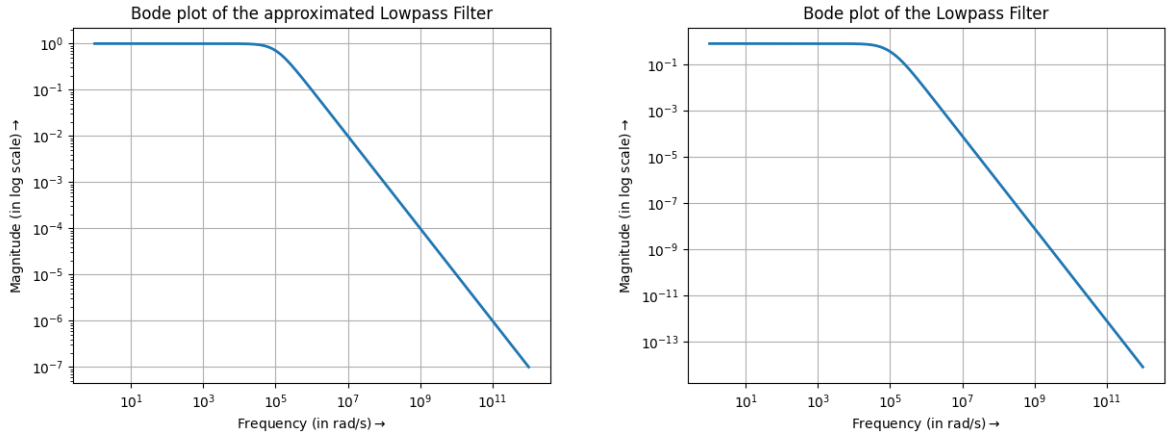


Figure 11: Magnitude Response of the two Transfer Functions for Lowpass Filter

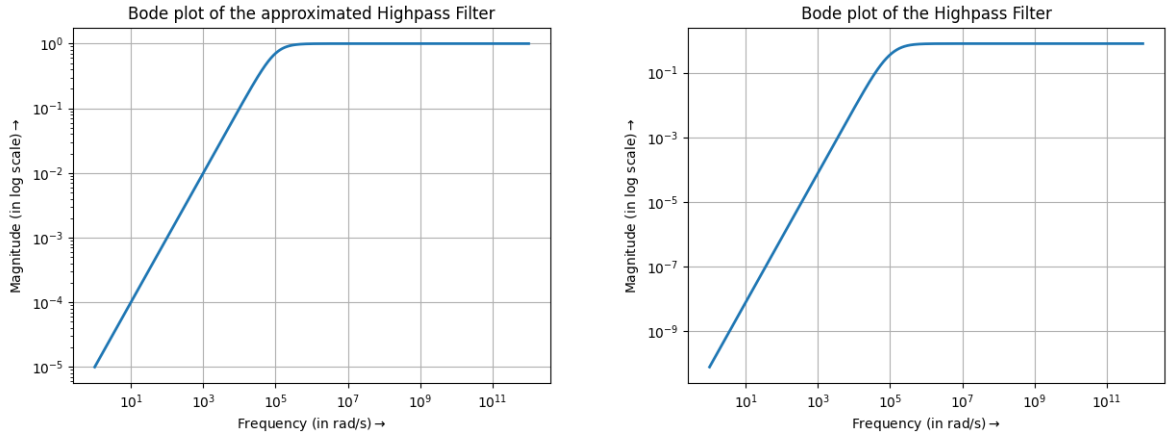


Figure 12: Magnitude Response of the two Transfer Functions for Highpass Filter

From the graphs, we can see how close the approximations are. The graph changes its slope at around $0.1MHz$ and the magnitude response at low frequencies in the case of Lowpass Filter and high frequencies in the case of Highpass Filter are almost the same. Since these ranges are the significant operating frequencies for the respective Filters we can say that this is a very good approximation.

4 Conclusion

Thus we have succesfully analysed the transfer functions of an Active Lowpass Filter and an Active Highpass Filter. Sympy provides a convenient way to analyse LTI systems using their Laplace Transforms. We converted the symbolic representation of the transfer functions to a LTI class which can be used with SciPy's signal toolbox. Active Filters are very power efficient and their magnitude response is frequency dependent which opens up a whole new world of audio signal processing.

The exact transfer function of the systems are fairly complicated and we have also simplified them to approximate expressions to ease our analysis in large systems.