EE2703: Applied Programming Lab

Week 7: Solving Laplace Transforms Using Sympy

Author: Krishna Somasundaram, EE19B147

25th April, 2021

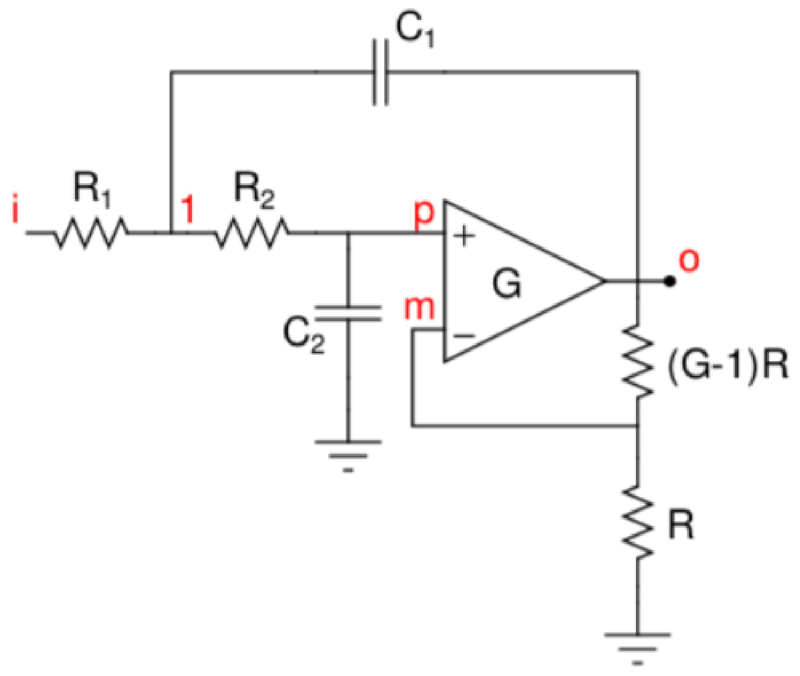
1 Abstract:

We shall analyse Active Filters using Laplace Transform and explore the sympy library in Python. For solving the system we shall continue using SciPy's signal library. Here we shall only consider filters made using OpAmps for Active Lowpass and Active Highpass Filter.

2 Theory:

2.1 Lowpass Filter:

First we shall analyse a Lowpass Filter.



The circuit equations are:

$$V_m = \frac{V_o}{G}$$
$$V_p = \frac{V_1}{1+j\omega R_2C_2}$$
$$V_0 = G(V_p - V_m)$$
$$\frac{V_i-V_1}{R_1} + \frac{V_p-V_1}{R_2} + j\omega C_1(V_o - V_1) = 0$$

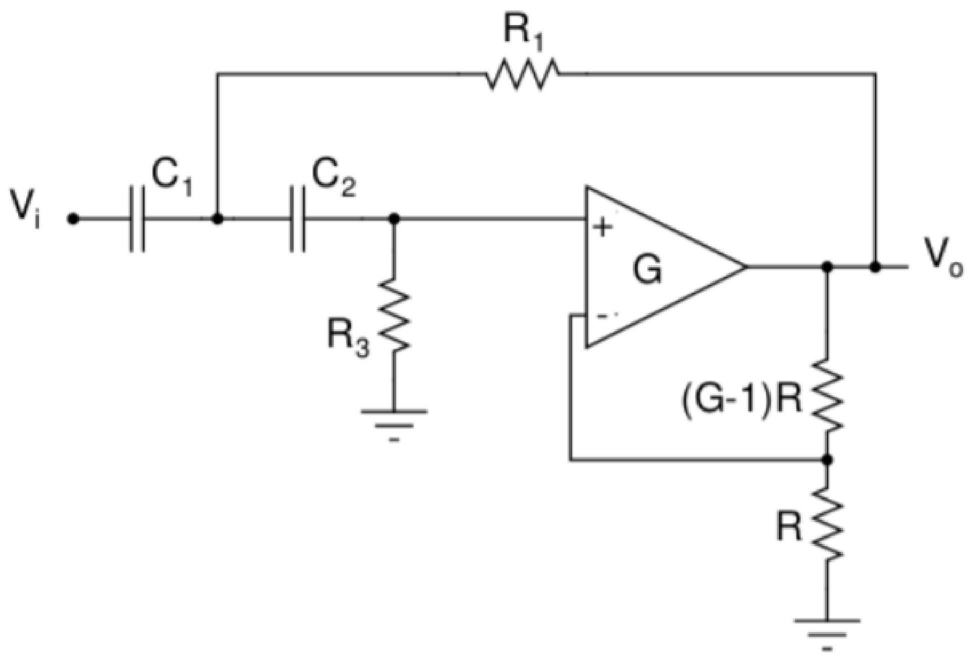
Solving the above equations we get this transfer function $V_o(s)/V_i(s)$
The approx expression we get is:

$$V_o \approx \frac{V_i}{1+j\omega R_1C_1}$$

This is when we assume $R_1 = R_2$ and $C_1 = C_2$. So for higher frequencies, the signal gets attenuated which is expected for 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 this 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 as expected from a Highpass Filter.

3 Code analysis:

These are the libraries that I used

```
import scipy.signal as sp
import sympy as sy
from sympy.abc import s
from pylab import *
```

Next are a description of the functions I've used, with comments.

1. Function to extract coefficients from symbolic representation of a transfer function and returns LTI class.

```
# Here, using sympy we extract numerator and denominator, convert to polynomials
# and store coeffs. Then we apply LTI on it.
def LTI_find(sym):
    num, den = sy.fraction(sym)
    num, den = sy.Poly(num, s), sy.Poly(den, s)
    num, den = num.all_coeffs(), den.all_coeffs()
    H_lti = sp.lti(array(num, dtype = float), array(den, dtype = float))
    return H_lti
```

2. Function to plot Magnitude plot (Bode plot).

```
# Function to plot Magnitude Response (Bode plot)
def bode_plotting(H, heading):
    w = logspace(0, 12, 1001)
    freqs = 1j * w
    hf = sy.lambdify(s, H, 'numpy')
    h = hf(freqs)

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

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

```
# Function to plot response for a given input signal
def input_sim(H, Vi, t_range, heading, type):
    x = sp.lsim(H, Vi, t_range)[1]
    plot(t_range, Vi, label = 'Input signal')
    plot(t_range, x, label = 'Output response')
    title(heading + " input signal vs output response of " + type)
    xlabel("time (in seconds)")
    ylabel("Voltage (in V)")
    legend()
    grid(True)
    show()
```

4. Function to analyse a given Transfer Function through various plots.

```
# Common function to analyse given input signal and transfer function
def plot_analysis(H, Vi, t_range, heading):
    H_lti = LTI_find(H)
    bode_plotting(H, heading)

    ## Computing and plotting step response
    step_response = H * 1/s
    step_response_lti = LTI_find(step_response)
```

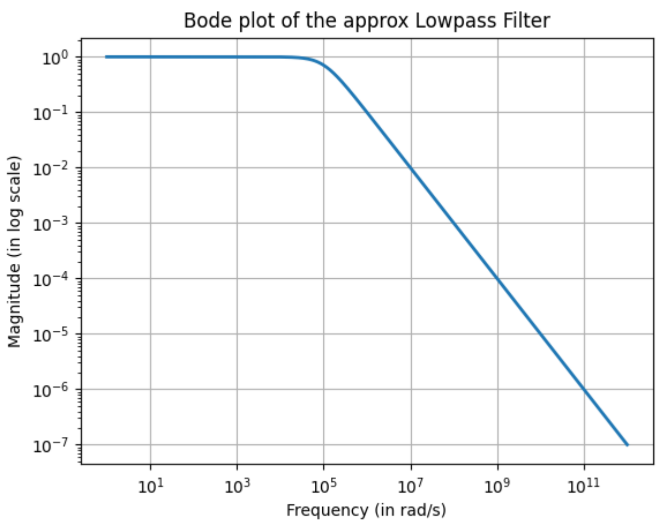
```
x = sp.impulse(step_response_lti, None, t_range)[1] ## Warning??
figure(1)
plot(t_range, x)
title("Step Response of the " + heading)
xlabel("time (in seconds)")
ylabel("Response (in V)")
grid(True)
show()

## Response for a damped low frequency signal
V_low_freq = exp(-300 * t_range) * cos(2 * 10**3 * pi * t_range)
input_sim(H_lti, V_low_freq, t_range, "Low frequency (1KHz)", heading)

## Response to a damped high frequency signal
V_high_freq = exp(-300 * t_range) * cos(2 * 10**6 * pi * t_range)
input_sim(H_lti, V_high_freq, t_range, "High frequency (1MHz)", heading)

## Response to the given input signal
input_sim(H_lti, Vi, t_range, "Given", heading)
```

3.1 Lowpass Filter:

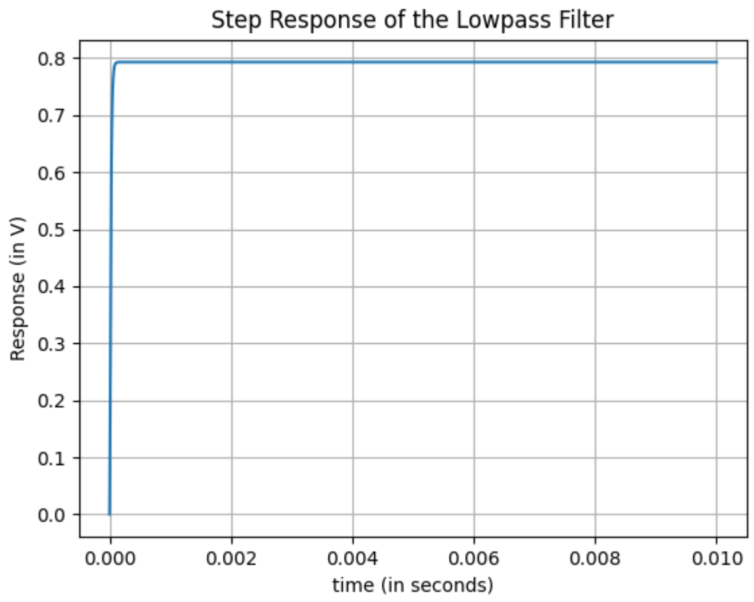
Lowpass Filter's Transfer Function:

```
# Lowpass case
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.

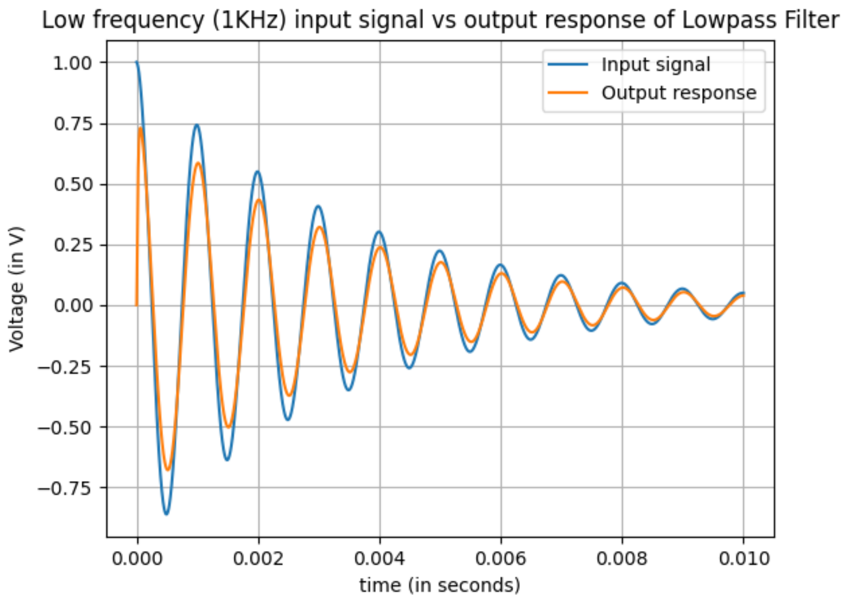


For high Freqs the output has a negligible gain and goes to 0. Now let's look at the step response of the system.



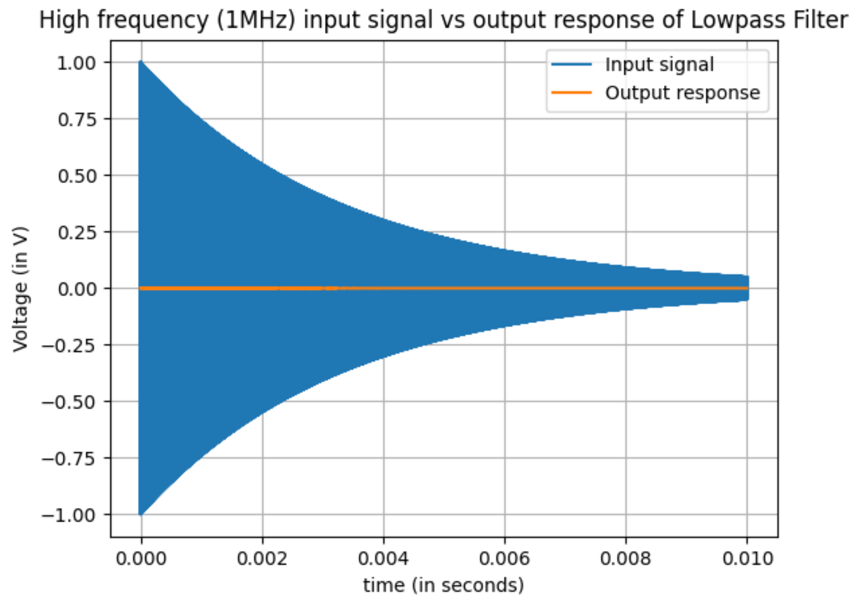
The step response of the system steadies with time giving a constant output. It is expected as Lowpass Filters allow DC power to pass through.

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. The 3 cases are shown next.



The frequency of the signal is $1KHz$. The output is starting from zero and has almost quickly stablised to a sinusoidal waveform.

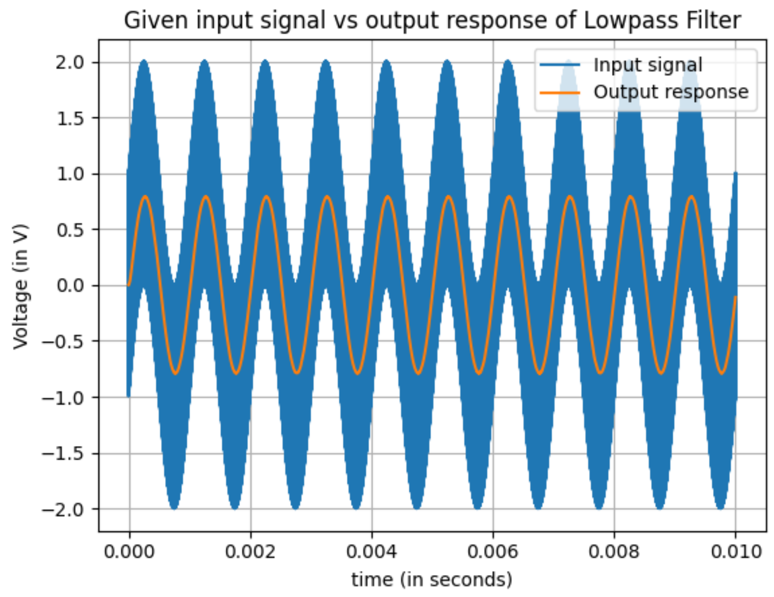
Now, let's pass a high frequency signal and see the output.



Here the frequency is so high ($1MHz$) and so it appears to be a continous block. The signal goes to 0, as expected

Let's pass the following given input.

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



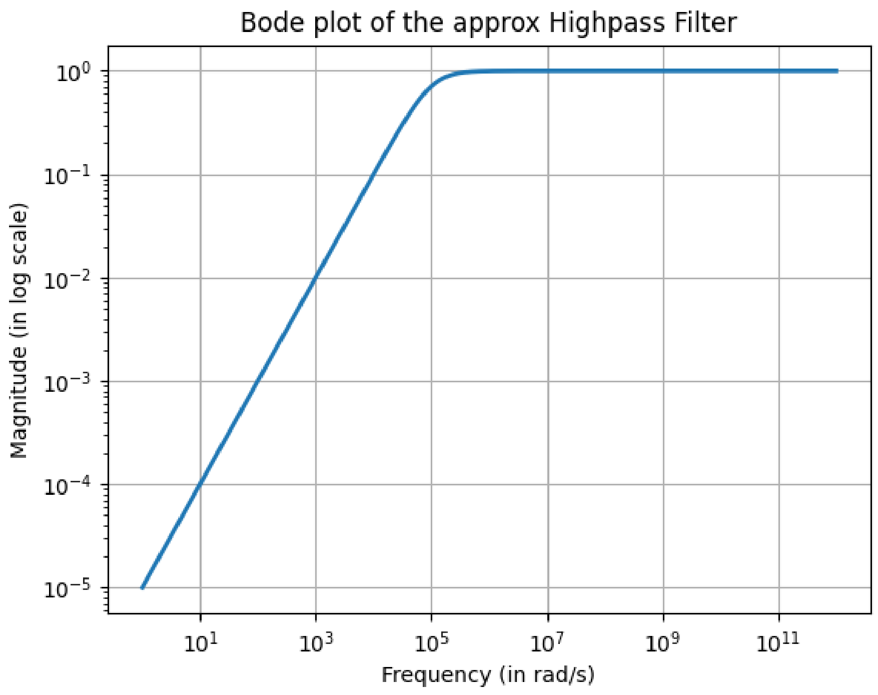
The Lowpass Filter will remove the high frequency component and the output will only contain the low frequency component. We should also note that the amplitude of the signal affects the output wave in an almost linear manner and doesn't influence our filter design.

3.2 Highpass Filter

Highpass Transfer Function:

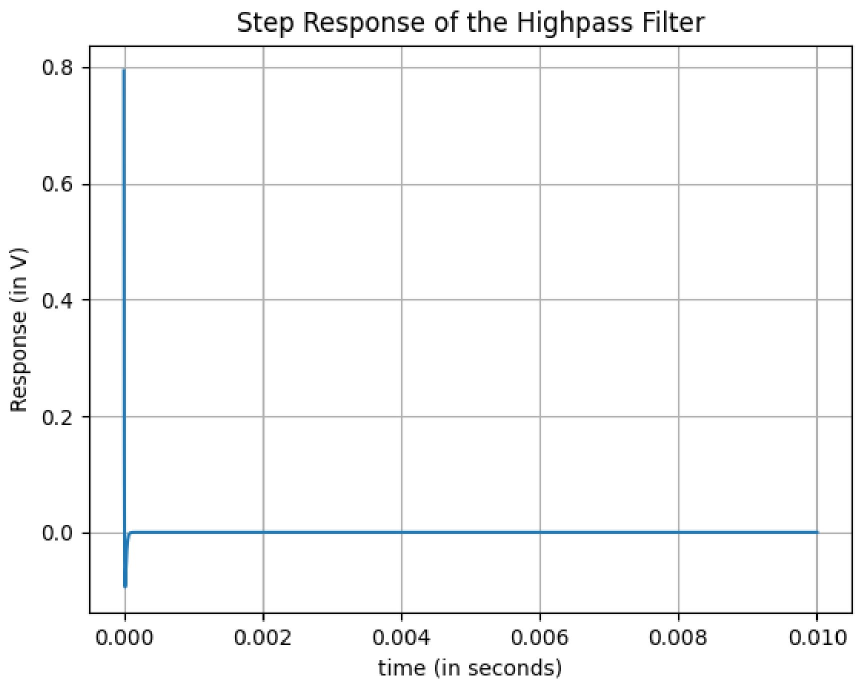
```
def highpass_tf(R1, R3, C1, C2, G):
    A = sy.Matrix([[0, -1, 0, 1/G],[s * C2 * R3/(s * C2 * R3 + 1), 0, -1, 0],
         [0, G, -G, 1],[-s * C2 -1/R1 - s * C1, 0, s * C2, 1/R1]])
    b = sy.Matrix([0, 0, 0, -s * C1])
    V = A.inv() * b
    return A, b, V
```

Magnitude Response of the Filter:



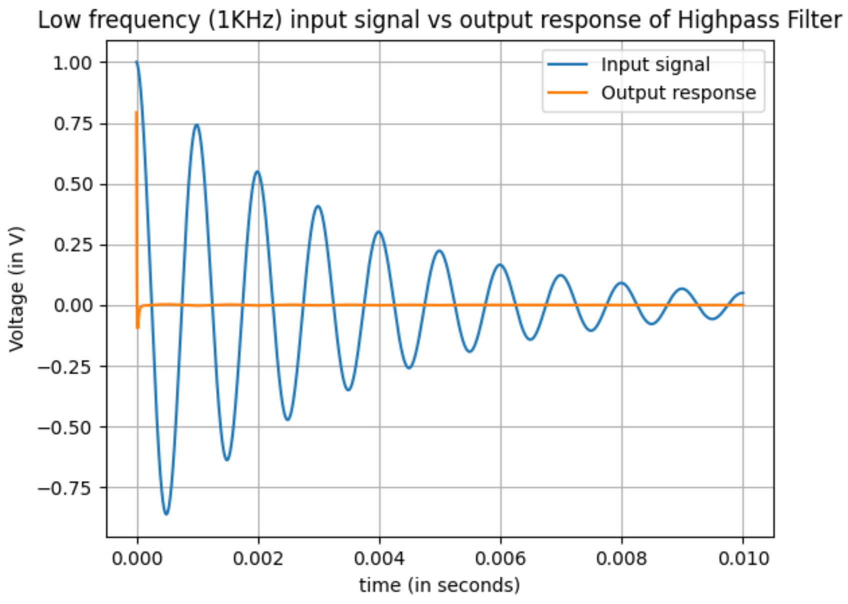
So here for low frequencies the output gets attenuated.

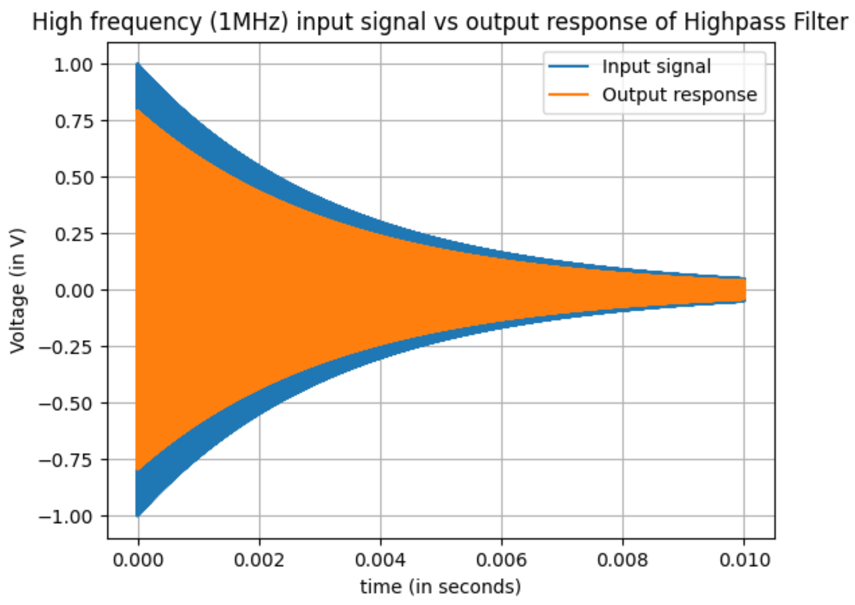
Step response of the system:



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

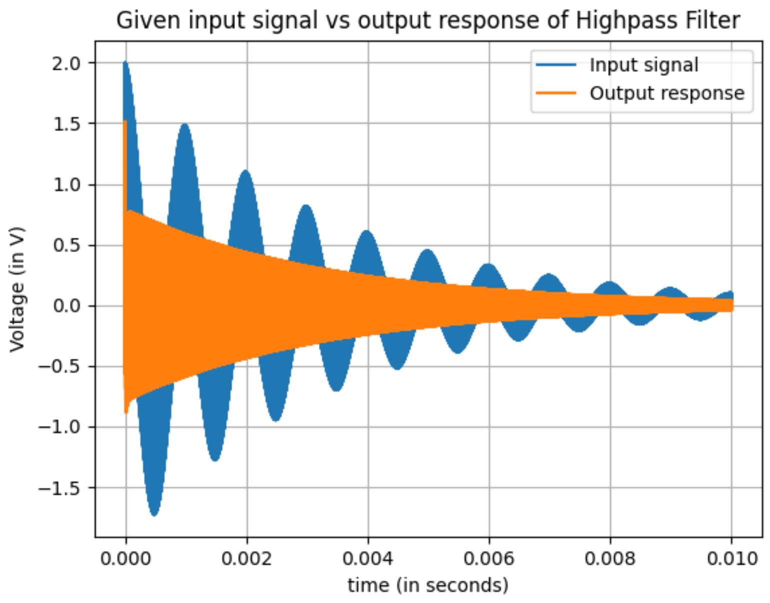
Let's now give the Highpass Filter a signal with a few different frequencies. When we pass an input with mixed frequencies, only those components which have frequencies can give an output response and the lower frequency components get attenuated. The 3 cases are shown next.





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.

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

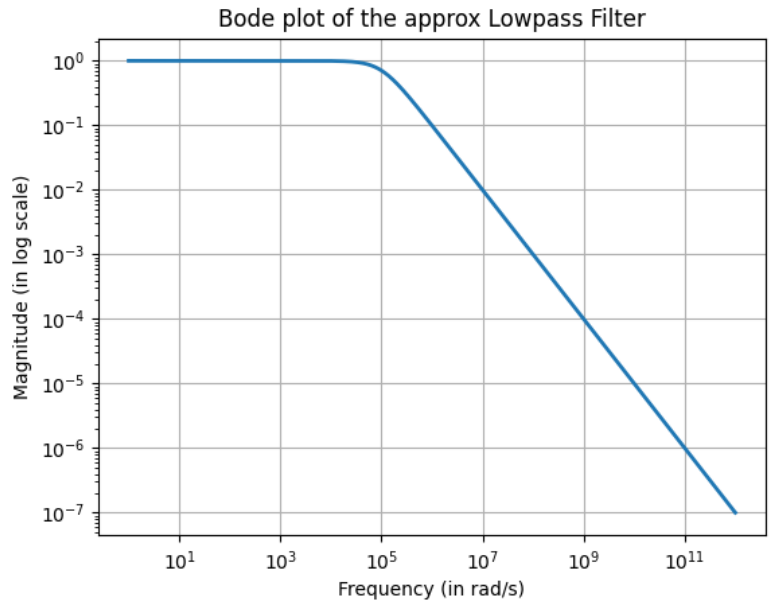


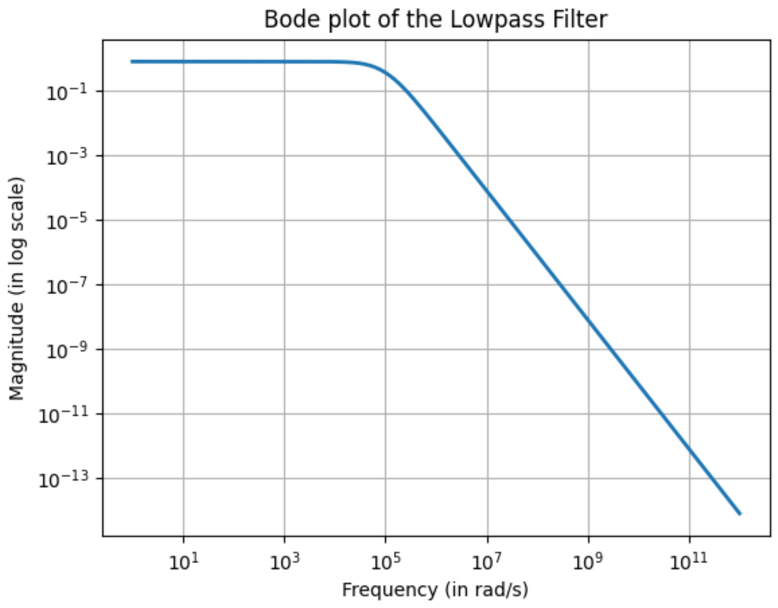
Here the input signal is again a combination of two sinusoids, one with a low frequency and other with high frequency. The low frequency component goes to zero as we can see from the graph.

3.3 Real vs Approx Transfer Functions

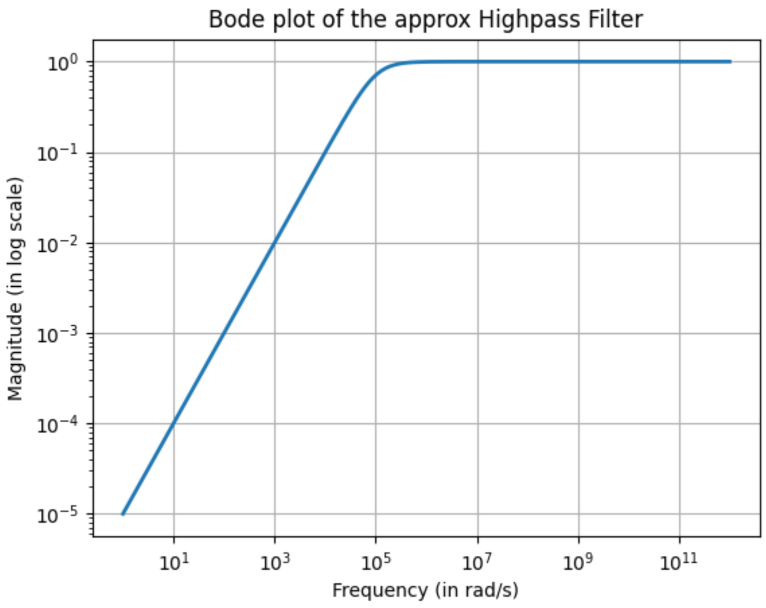
Letâs compare the Magnitude Response of the approximate Transfer Functions with the Actual Transfer Function. Even though our transfer functions are supposed to be second order, as R and C values are equal, it behaves like a first order system.

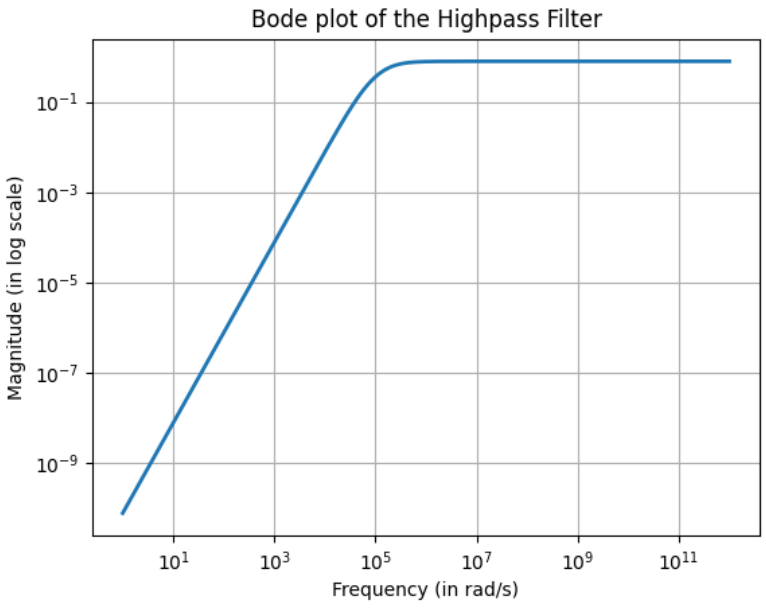
Lowpass case:





Highpass case:





As we can see, our result is very close to the real solution. The point of inflection is around 100 kHz. As these are the ranges where significant operating frequencies exist for the respective Filters these approximations work.

4 Conclusion

We have used the Sympy library in python to analyse LTI systems and their laplace transforms. We used it to analyse the transfer functions of an Active Lowpass Filter and an Active Highpass Filter. We then converted the symbolic representation of the transfer functions to a LTI class which is then used with SciPyâs signal toolbox. Active Filters are very energy efficient and their magnitude response is frequency dependent which is used in a variety of fields, such as audio signal processing. We have used approximations to simplify complex transfer functions and shown that the approximations are good enough to use.