Assignment 7: Sympy

Saurav Sachin Kale, EE19B141

April 25, 2021

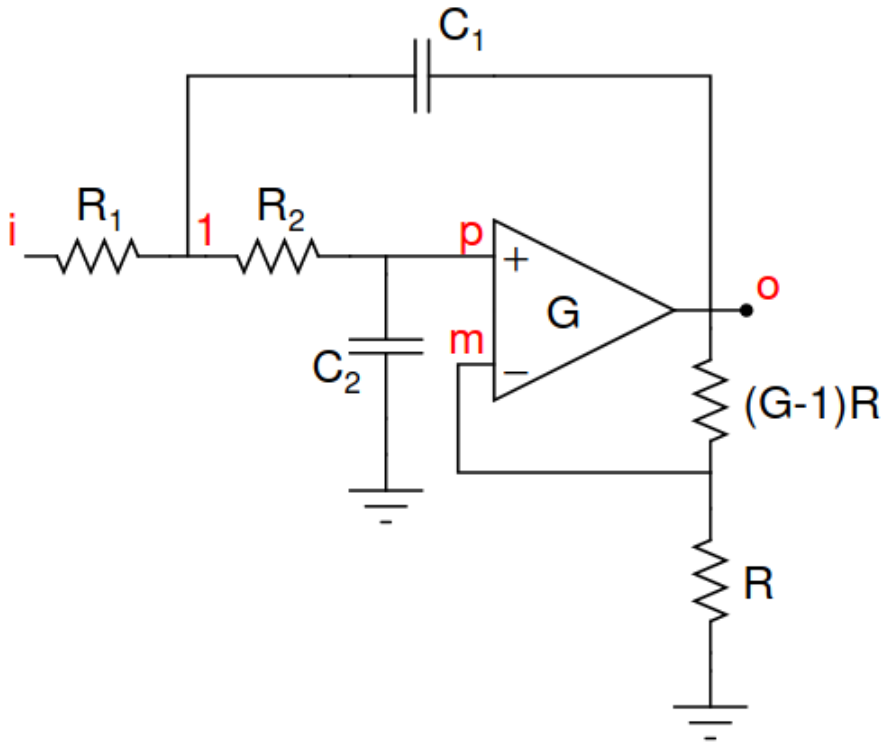
1 Aim

- To solve and analyse circuits using Laplace Transforms
- To explore symbolic algebra in Python using the SymPy module
- Analysis of Active Filters

2 Procedure

2.1 The Active Lowpass filter

Presented below is a circuit for an active lowpass filter. We shall analyse this in Laplace domain and use SymPy to obtain the solutions for various input signals. Here the values of the components are given to be $G = 1.586, R_1 = R_2 = 10k\Omega, C_1 = C_2 = 1nF$



Analysing in laplace domain we obtain the following equations:

$$V_m = \frac{V_o}{G} \tag{1}$$

$$V_1(\frac{1}{R_2} + \frac{1}{R_1} + C_1 s) - \frac{V_i}{R_1} - C_1 s V_o - \frac{V_p}{R_2} = 0 \tag{2}$$

$$V_p = \frac{V_1}{1 + R_2 C_2 s} \tag{3}$$

$$V_o = G(V_p - V_m) \tag{4}$$

Expressing these equations in matrix form, we get:

$$\begin{bmatrix} 0 & 0 & 1 & -\frac{1}{G} \\ -\frac{1}{1+R_2C_2s} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ (\frac{1}{R_2}+\frac{1}{R_1}+C_1s) & -\frac{1}{R_2} & 0 & -C_1s \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{V_i}{R_1} \end{bmatrix}$$

$$Av = b$$

Now we can solve this matrix equation for any given input signal V_i to obtain the matrix v. The last element of v is the output. We shall now solve this using SymPy for a variety of input signals.

- We first define the function `lowpass()` which generates the matrices A and b for given system parameters, solves for v and returns all 3 matrices. `s` is defined as a SymPy symbol here.

```
# lowpass filter circuit
def lowpass(R1, R2, C1, C2, G, Vi):
    A = sy.Matrix([[0, 0, 1, -1/G], [-1/(1+R2*C2*s), 1, 0, 0],
                   [0, -G, G, 1], [(1/R1 + 1/R2 + C1*s), -1/R2, 0, -C1*s]])
    b = sy.Matrix([0, 0, 0, Vi/R1])

    V = A.inv() * b

    return A, b, V
```

- Then we initialize this system with the given parameters, and extract the actual output signal in terms of s.

```
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)
Vo = V[3]
```

- We initialize our plotting space in frequency domain to be a logarithmic space from frequency ω of $1rad/s$ to $10^8 rad/s$. We then obtain the values for `s` we need for the frequency respose by populating an array with the imaginary values $j\omega$ that is the variable `ss`.

```
w = p.logspace(0,8,801)
ss = complex(0, 1)*w
```

- We obtain the frequency response by using the function `lambdify()` which converts the SymPy transfer function to a lambda function which can be evaluated for all ω in our logspace.

```
hf = sy.lambdify(s, Vo, 'numpy')
```

- We obtain the magnitude response by evaluating the absolute value of frequency response for our logspace ω. We then generate the magnitude plot of this system.

```
v = hf(ss)
p.loglog(w,abs(v),lw=2)
```

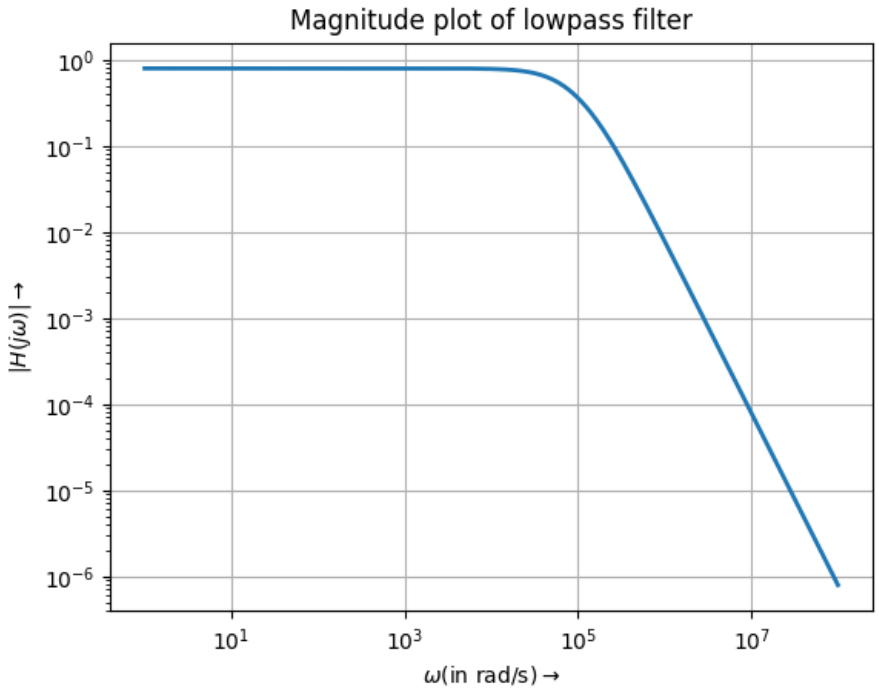


Figure 1: Magnitude of Frequency Response of Active Lowpass filter

- As expected, it is a lowpass filter.
- We shall now evaluate the step response of this lowpass filter.
- We know that the laplace transform of the unit step function is $\frac{1}{s}$. For step response, we will simply multiply the function with $\frac{1}{s}$ to obtain the laplace transform of the output.
- We want to obtain the time domain output. Therefore we must first convert the SymPy transfer function into a `scipy.signal.lti` object. We will then use `scipy.signal.impulse()` to obtain the time domain function for a given time interval.

```
t = p.linspace(0, 0.001, 1000)
StepResponse_SDomain = convertSympyToLTI(Vo * 1/s)
t, StepResponse = sp.impulse(StepResponse_SDomain, None, t)
```

- The conversion from SymPy function to `scipy.signal.lti` object is quite simple. We have made a function `convertSympyToLTI()` for this purpose. We simply obtain the coefficients on both numerator and denominator side by first extracting the polynomials, then obtaining all coefficients.
- We then create our new `scipy.signal.lti` object with these arrays

```
# converts a sympy transfer function to a scipy.signal.lti object
def convertSympyToLTI(H):
    # fraction function: Returns a pair with
        # expressions numerator and denominator.
    n, d = sy.fraction(H)
    # convert those to polynomials
    polynum = sy.poly(n)
```

```
    polyden = sy.poly(d)
    # get arrays for their coefficients
    numCoeff = polynum.all_coeffs()
    denCoeff = polyden.all_coeffs()

    # feed the coefficient arrays into sp.lti to get an
        # lti system object with the transfer function H
    H_lti = sp.lti(p.array(numCoeff, dtype=float),
                                        p.array(denCoeff, dtype=float))

    return H_lti
```

- We have obtain the time domain function for step response. We now plot it.

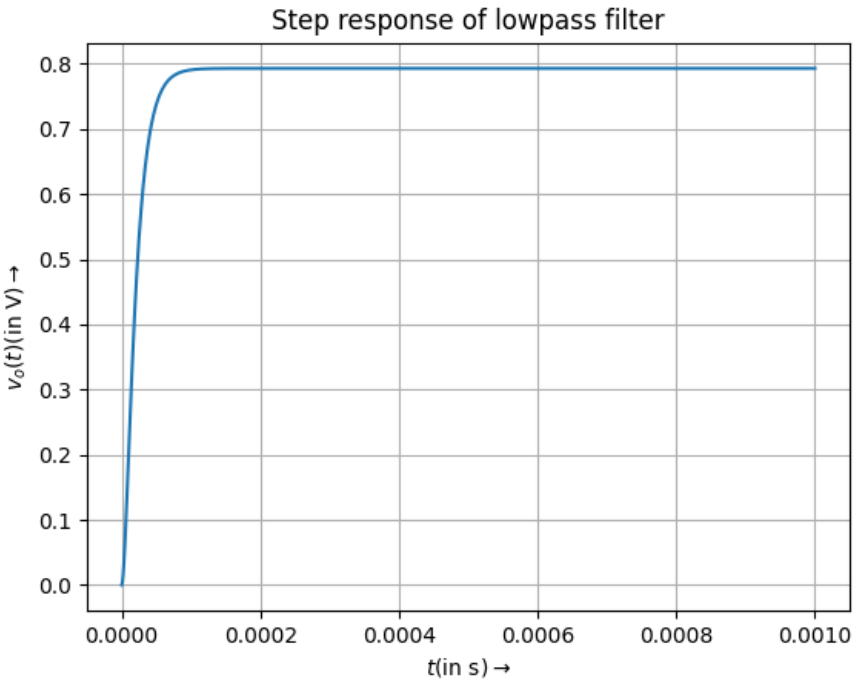


Figure 2: Time domain Step Response for Active Lowpass filter

- Now we analyse the output of the system for

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

- Here we will not be converting V_i to laplace domain. Instead, we will now use `scipy.signal.lsim()` to directly evaluate the convolution of impulse response and the input signal.

```
t = p.linspace(0,0.01,100000)
Vinp = p.sin(2e3*p.pi*t)+p.cos(2e6*p.pi*t)
t, Vout, svec = sp.lsim(convertSympyToLTI(Vo), Vinp, t)
```

- We obtain the following plot:

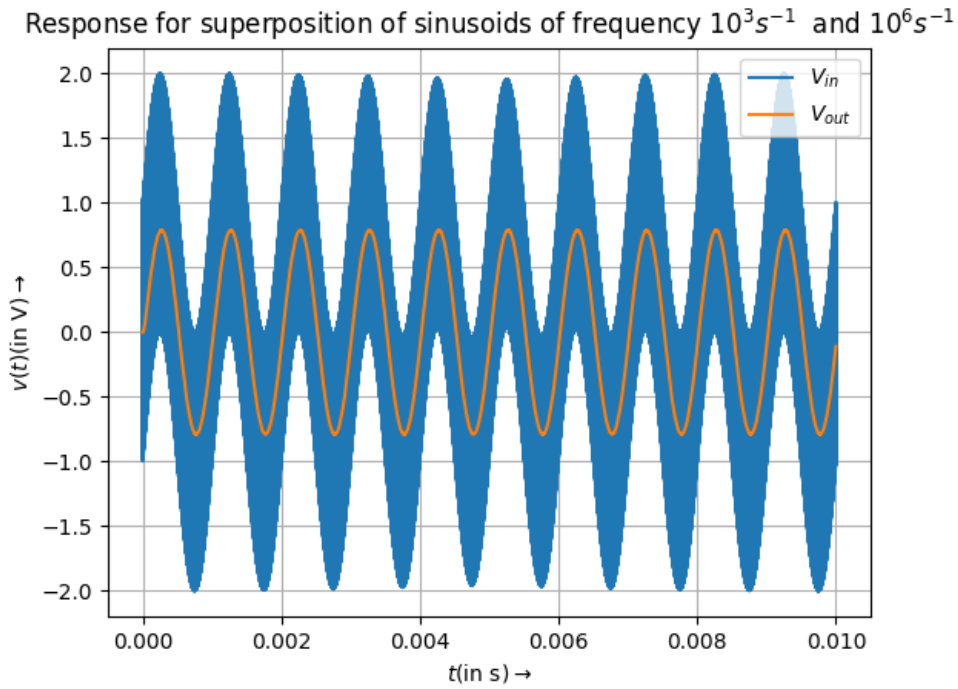
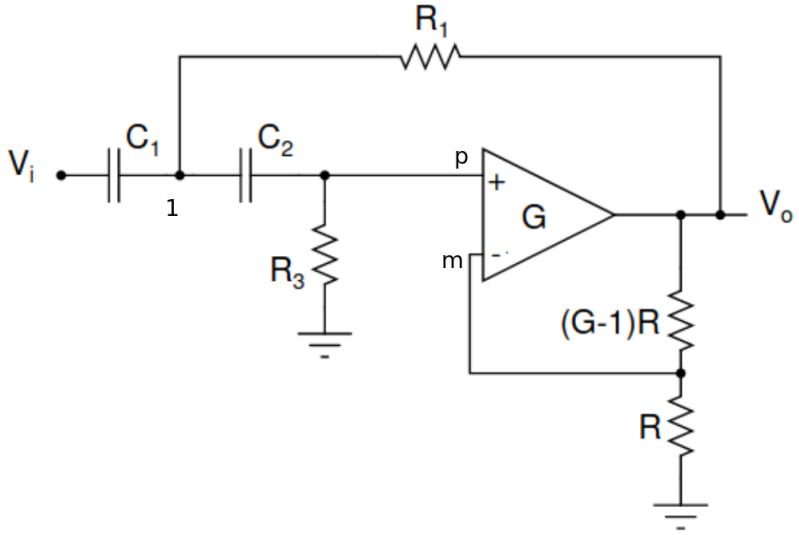


Figure 3: Time domain output $V_o(t)$ for the input $V_i(t) = (sin(2000\pi t) + cos(2 \times 10^6 \pi t))u(t)$

- Again, the plot is as expected, since $10^6 rad/s$ is severely attenuated, we see only the $10^3 rad/s$ component which survives since it is comfortably within the 3dB bandwidth.

2.2 The Active Highpass filter

Presented below is a circuit for an active highpass filter. We shall analyse this in Laplace domain and use SymPy to obtain the solutions for various input signals. Here the values of the components are given to be $G = 1.586, R_1 = R_3 = 10k\Omega, C_1 = C_2 = 1nF$.



Analysing in laplace domain we obtain the following equations:

$$V_m = \frac{V_o}{G} \tag{5}$$

$$V_1(C_2 s + \frac{1}{R_1} + C_1 s) - C_1 s V_i - \frac{V_o}{R_1} - C_2 s V_p = 0 \tag{6}$$

$$V_p = \frac{R_3 C_2 s V_1}{1 + R_3 C_2 s} \tag{7}$$

$$V_o = G(V_p - V_m) \tag{8}$$

Expressing these equations in matrix form, we get:

$$\begin{bmatrix} 0 & 0 & 1 & -\frac{1}{G} \\ -\frac{R_3 C_2 s}{1+R_3 C_2 s} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ (C_2 s + \frac{1}{R_1} + C_1 s) & -C_2 s & 0 & -\frac{1}{R_1} \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ C_1 s V_i \end{bmatrix}$$

$$Av = b$$

Now we can solve this matrix equation for any given input signal V_i to obtain the matrix v. The last element of v is the output. We shall now solve this using SymPy for a variety of input signals.

- We create a `highpass()` function for the transfer function which accepts circuit parameters and solves for the v matrix. It returns A, b as well as v matrices.

```
# highpass filter circuit
def highpass(R1, R3, C1, C2, G, Vi):
    A = sy.Matrix([[0, 0, 1, -1/G], [-R3*C2*s/(1+R3*C2*s), 1, 0, 0],
                   [0, -G, G, 1], [(C1*s + C2*s + 1/R1), -C2*s, 0, -1/R1]])
    b = sy.Matrix([0, 0, 0, Vi*s*C1])

    V = A.inv() * b

    return A, b, V
```

- We use the same method as explained for lowpass to find and generate the magnitude plot of the transfer function using the `lambdify()` function

```
# first of all show the magnitude plot
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)
print('G=1000')
Vo = V[3]
print(Vo)
w = p.logspace(0,8,801)
ss = complex(0, 1)*w
hf = sy.lambdify(s, Vo, 'numpy')
v = hf(ss)
p.loglog(w,abs(v),lw=2)
```

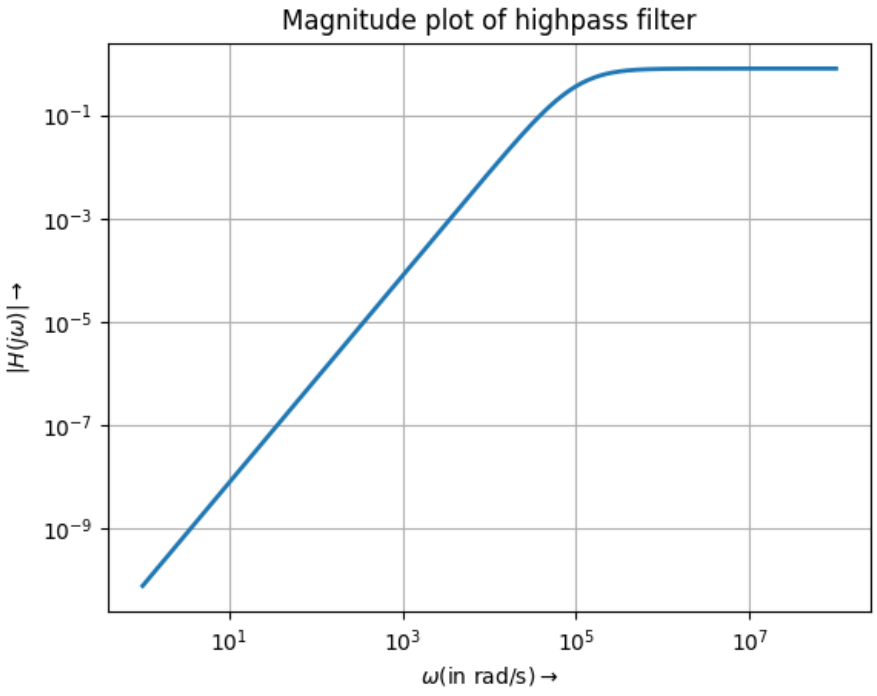


Figure 4: Magnitude of Frequency Response of Active Highpass filter

- As predicted, it is indeed a highpass filter.

- We now analyse the output of the system for the input signal of a decaying sinusoid. First case we will consider a sinusoid of low frequency, here $1Hz$.

$$V_i(t) = e^{-0.5t}sin(2\pi t)$$

using the `scipy.signal.lsim()` function as follows:

```
# output for a decaying sinusoid with
# freq = 1
# decay factor = 0.5
t = p.linspace(0, 5, 1000)
Vinp = p.exp(-0.5*t) * p.sin(2*p.pi*t)
t, Vout, svec = sp.lsim(convertSympyToLTI(Vo), Vinp, t)
```

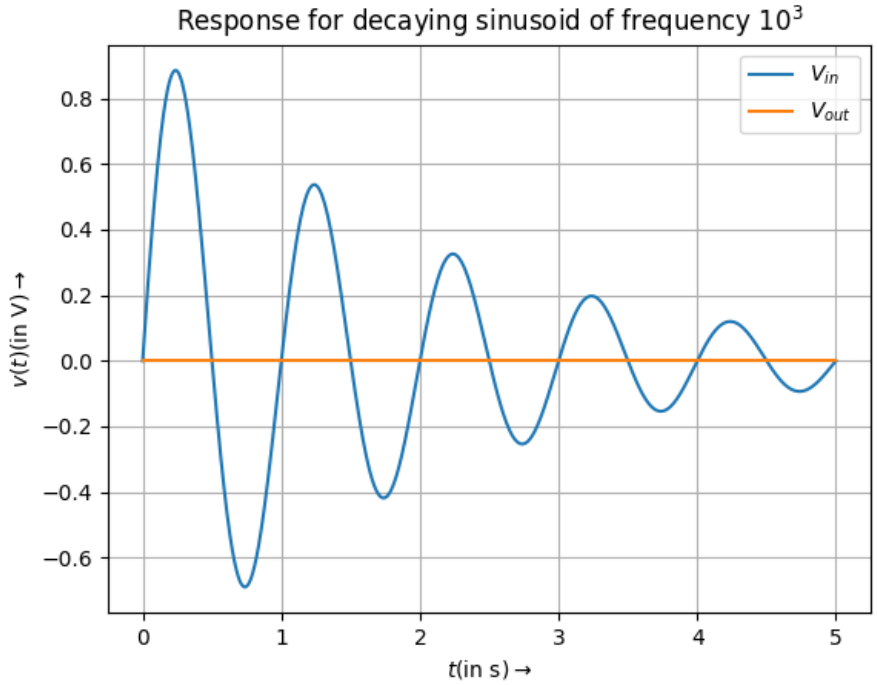


Figure 5: Time domain output $V_o(t)$ for the input $V_i(t) = e^{-0.5t}sin(2\pi t)$

- The system almost completely cuts out this frequency because $2\pi rad/s$ is very low frequency which is attenuated heavily, hence the highpass filter shows no output.
- Let us now test this for a high frequency (within the passband). Let us consider 10^6Hz.

$$V_i(t) = e^{-0.5t}sin(2 \times 10^6\pi t)$$

```
# output for a decaying sinusoid with
# freq = 1e6
# decay factor = 0.5
t = p.linspace(0, 0.0001, 10000)
Vinp = p.exp(-0.5*t) * p.sin(2e6*p.pi*t)
t, Vout, svec = sp.lsim(convertSympyToLTI(Vo), Vinp, t)
```

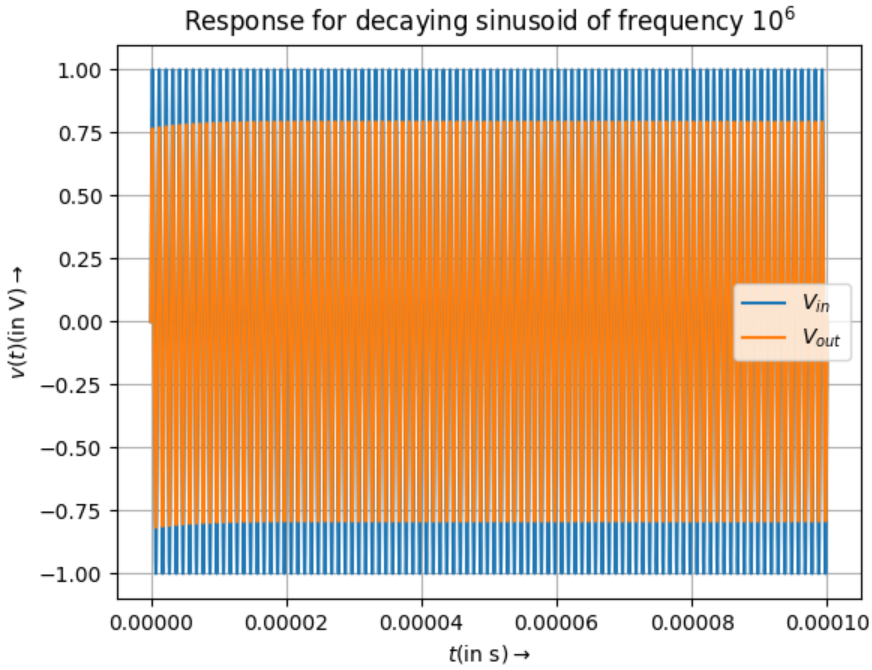


Figure 6: Time domain output $V_o(t)$ for the input $V_i(t) = e^{-0.5t}sin(2 \times 10^6 \pi t)$

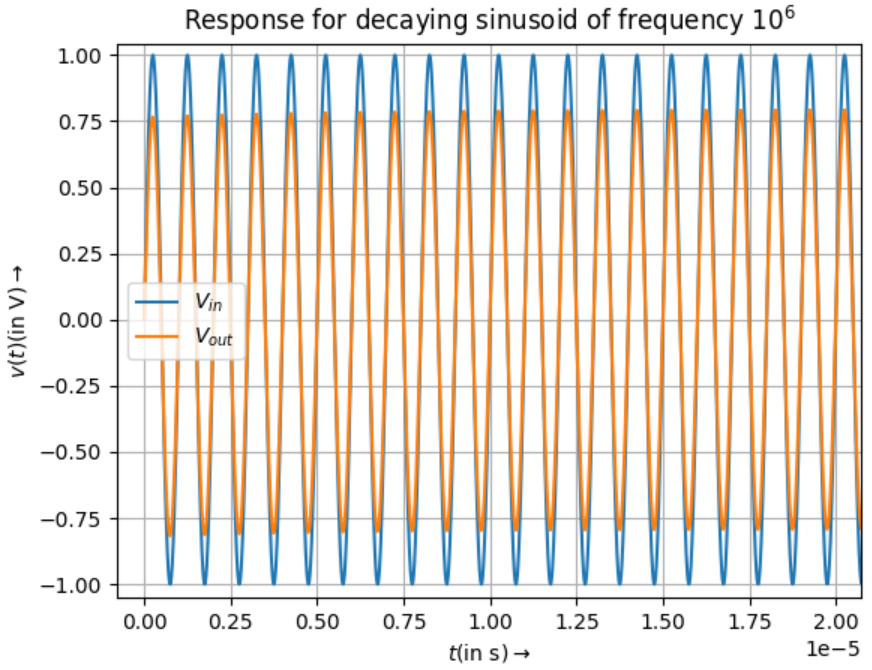


Figure 7: A zoomed in look at Figure 6

- We can see that the signal survives, albeit it is slightly attenuated because at $2 \times 10^6 \pi rad/s$ the gain is just smaller than 0dB.
- This is expected, because the frequency is within the passband of the highpass filter.
- We now analyse the step response of this system in the same way we did for lowpass filter, using `scipy.signal.impulse()`

```
# now we obtain the step response
t = p.linspace(0, 0.001, 1000)
StepResponse_SDomain = convertSympyToLTI(Vo * 1/s)
t, StepResponse = sp.impulse(StepResponse_SDomain, None, t)
```

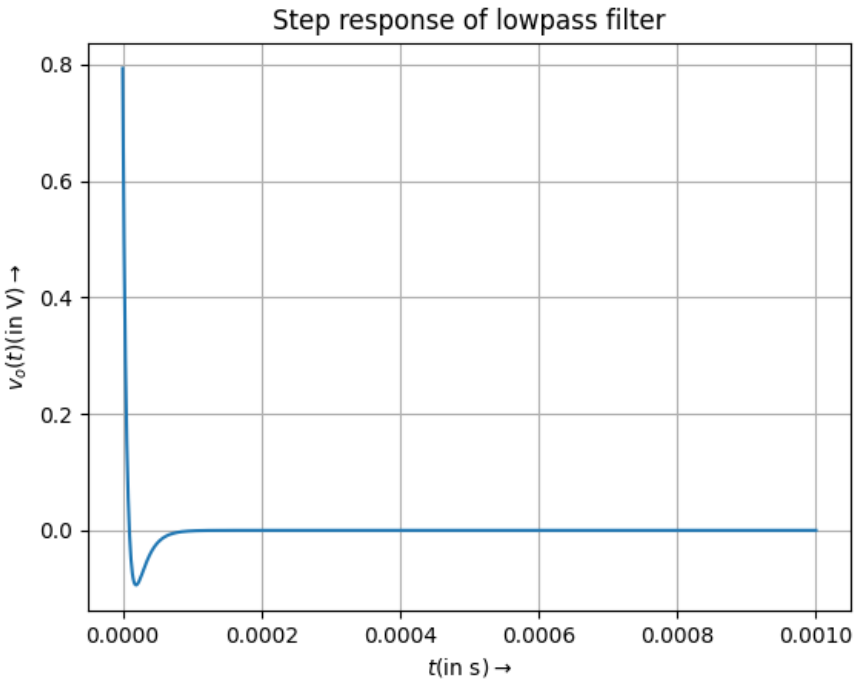


Figure 8: Time domain Step Response for Active Highpass filter

- The reason for the initial peak is the initial conditions. The capacitor does not allow instantaneous changes in the voltage across it. Before $t = 0$, both capacitors have no charge, and the voltage across both is 0, and this will not change instantaneously.
- Refering to the circuit diagram, the instant the 1V of the unit step is applied ($t = 0^+$), the drop across the capacitors C_1 and C_2 will still be zero. Thereform $V_m = 1V$.
- $V_m = \frac{V_o}{G}$ and $V_o = G(V_p - V_m)$, therefore we get

$$V_0 = G(1 - \frac{V_o}{G})$$

$$2V_o = G$$

$$V_o = \frac{G}{2} = \frac{1.586}{2} = 0.793 \approx 0.8V$$

- This is what explains the initial peaking of 0.8V. It then quickly decays and settles down to the expected 0V because it is a highpass filter, and DC will give no output on such a filter

3 Conclusions

- Two electrical systems, the Active Lowpass and Active Highpass filters were analysed using the Laplace Transform.
- The analysis was performed using symbolic algebra in the SymPy module for Python, and Magnitude plots of the frequency response were plotted for both filters.
- SciPy's Signals module was used to obtain time domain solutions for given inputs, and their cutoff frequencies were found.
- The step response of both filters was plotted and analysed.