# Assignment No. 7: Report

-Akash Reddy A, EE17B001

March 19, 2019

## 1 Overview

Assignment 7 deals with circuit analysis using `sympy`.

This assignment uses `sympy`'s symbolic algebra and Laplace Transforms to analyse two given non-linear circuits: **Butterworth low-pass and high-pass filters**.

`sympy` is a Python library that can be used to exercise a plethora of powerful symbolic algebra operations on Laplace Transform Kirchoff's equations of systems. It enables us to directly write the terms of the equations in matrix form, and very efficiently obtain the solution by solving the matrix equation.

We also use some of the functions belonging to the `signal` toolbox (which were also used in Assignment 6) for Laplace Domain Analysis.

## 2 The Assignment

### Relevant Theory

Capacitors and inductors can be treated as providing a certain impedance to the flow of current in ac circuits. The values of their respective impedances are given as:

$$Z_C = \frac{1}{sC}$$

and

$$Z_L = sL$$

where $s = \sigma + j\omega$ is the *complex frequency* of the input signal. These impedances can be treated as resistors which obey Ohm's Law $I = \frac{V}{Z}$ while penning down the nodal or mesh analysis equations of any circuit.

Butterworth filters are filters that are designed to keep the frequency response as flat as possible in the passband. They are also called maximally flat magnitude filters.

This report tackles the Assignment in a question-by-question approach.

### 2.1 Question 1

The first circuit that we consider is the low-pass filter.

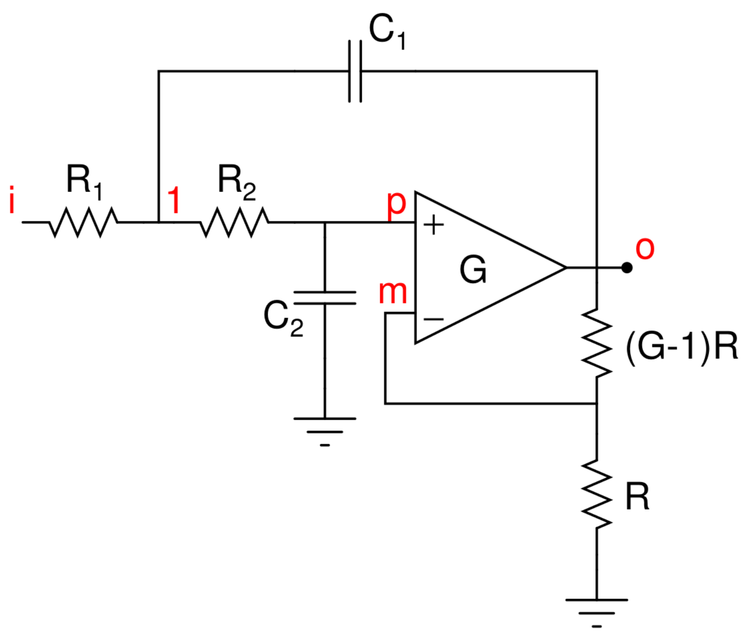

Figure 1: Butterworth low-pass filter

The values taken are $R_1 = R_2 = 10k\Omega, C_1 = C_2 = 1nF$, and $G = 1.586$. The circuit equations are:

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

$$V_p = V_1 \cdot \frac{1}{1 + sR_2C_2} \tag{2}$$

$$V_o = G(V_p - V_m) \tag{3}$$

$$\frac{V_i - V_1}{R_1} + \frac{V_p - V_1}{R_2} + sC_1(V_o - V_1) = 0 \tag{4}$$

Putting the above results in a single matrix equation, we get:

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

$$A \cdot v = b$$

$$v = A^{-1} \cdot b$$

We solve the equation, and take the last element of the solution vector in order to get $V_o$.

When required to find the step response, we put $v_i(t) = u(t)$ or, correspondingly, $V_i(s) = \frac{1}{s}$ in the matrix equation, so that the obtained $V_o$ represents the step response.

All this is done in the code below. The `Matrix` option that `sympy` offers makes it massively easy to input our equations in a matrix and solve the matrix equation by taking inverse easily with the `inv()` function.

**Relevant Code:**

Firstly, we define a function `transfer` to make the handling of transfer functions (matrix equation solutions in `sympy` form) non-repetitive and modular:

```
def transfer(x):
    s=symbols('s')
    Vo=simplify(x[3])
    vonumden=fraction(Vo)
    vonum=Poly(vonumden[0],s)
    voden=Poly(vonumden[1],s)
    vonumf=[float(i) for i in vonum.all_coeffs()]
    vodenf=[float(i) for i in voden.all_coeffs()]
    v0=sp.lti(vonumf, vodenf)
    return (v0,Vo)
```

We define a function `lowpass` that does all the above equation solving for the low-pass filter.

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

The question asks for the step response of the filter. For the sake of better understanding, we may plot the magnitudes of the *frequency-domain* step response *and* impulse response, after which we plot the required time-domain step response $s(t)$.

```
s=symbols('s')
w=p.logspace(0,8,801)
ss=1j*w
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)
v0,Vo=transfer(V)
hf=lambdify(s,Vo,'numpy')
v=hf(ss)
p.loglog(w,abs(v),lw=2,label='Impulse response')
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1/s)
v0,Vo=transfer(V)
hf=lambdify(s,Vo,'numpy')
v=hf(ss)
p.loglog(w,abs(v),lw=2,label='Step response')
p.xlabel('$\omega$\u279d', size=12)
p.ylabel('$|H(j\omega)|\u279d$', size=12)
p.title('Magnitude response of the step and impulse responses in the
frequency domain for low-pass filter')
p.legend(loc='upper right')
p.grid(True)
p.show()
t,vo=sp.impulse(v0,None,np.linspace(0,50,501))
```

The above code does all of this.

**Plots Generated:**

The magnitude of the frequency-domain impulse response (transfer function) clearly shows that the system is a low-pass filter:

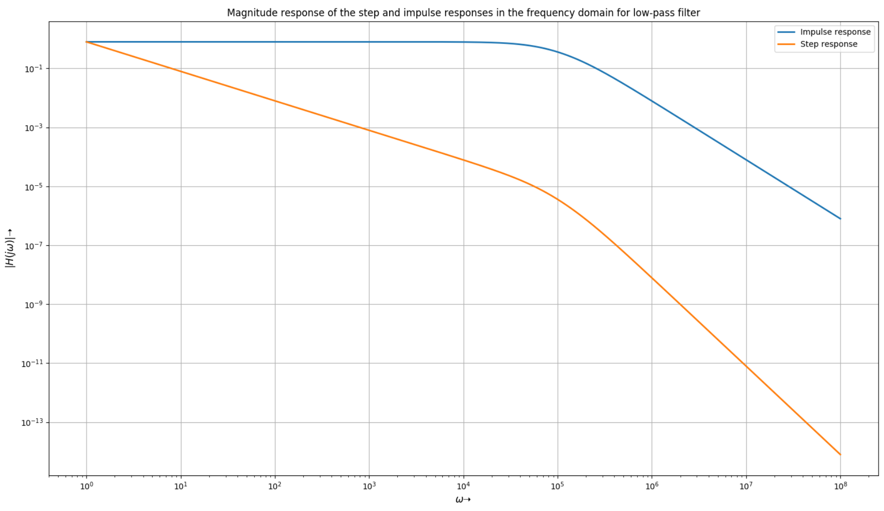

Figure 2: Magnitudes of frequency-domain step and impulse responses vs $\omega$ for low-pass filter

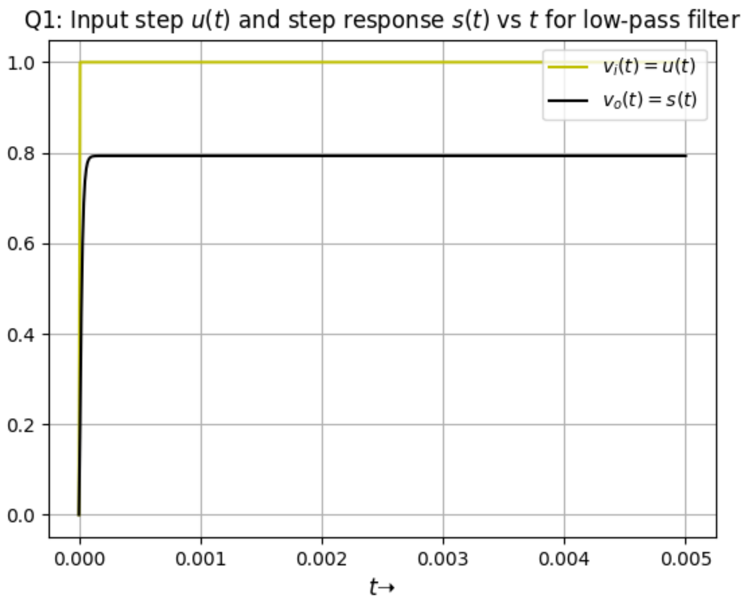

Figure 3: Input step $u(t)$ and step response $s(t)$ vs $t$ for low-pass filter

For an input $u(t)$, we get a step response equal to $0.793u(t)$ as seen above, due to the following:

The right half of the circuit is a non-inverting amplifier. The left half of the circuit performs the functions of low-pass and Butterworth flattening. Assuming we neglect those for analysing the step response, we can treat the circuit to have the functionality of a non-inverting amplifier.

The gain of a non-inverting amplifier using an ideal opamp of infinite gain would be $G$. However, the opamp is given to have a finite gain of $G$ itself. Therefore, the gain of a non-inverting amplifier using an opamp of finite gain would be:

$$Gain_{CL} = \frac{G}{1 + \frac{G}{A}}$$

where $A$ is the opamp gain.
We have $A = G$, therefore:

$$\begin{aligned} Gain_{CL} &= \frac{G}{1 + \frac{G}{G}} \\ &= \frac{G}{2} \end{aligned}$$

We have $G = 1.586$, therefore:

$$\begin{aligned} Gain_{CL} &= \frac{1.586}{2} \\ &= 0.793 \end{aligned}$$

**This explains the magnitude of the step response.**

We also notice that the step response takes some finite time to rise, unlike the instantaneous rise of the step input. This is because this point is a discontinuity. A discontinuity is a sharp jump, and therefore ideally requires infinite sinusoidal components in order to be represented accurately. However, **since our system is a low-pass filter, it does not allow higher frequencies which are needed to accurately account for the jump.** This is why the response takes some finite time to achieve the jump.

## 2.2 Question 2

We are required to calculate the output $v_o(t)$ of the above low-pass filter for a given input:

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

What we send as the input $v_i(t)$ is essentially a $\cos(2 \times 10^6\pi t)$ function superimposed on top of a $\sin(2000\pi t)$ wave. Therefore, we get a **sinusoidal band** as the input.

We observe from the magnitude plot of the impulse response $H(s) = \frac{V_o(s)}{V_i(s)}$, that the the ratio $\frac{V_o(s)}{V_i(s)}$ is very close to 1 at a frequency of $\omega = 10^3$, but drops to nearly $10^{-2}$ or $\frac{1}{100}$ at $\omega = 10^6$. This means that the low-pass filter allows the $\sin(2000\pi t)$ component to pass while attenuates/blocks the $\cos(2 \times 10^6\pi t)$ component.

The input *sinusoidal band* gets converted to a *sinusoidal wave*, which is attenuated due to the closed-loop gain of the amplifier circuit.

The solution is achieved by taking the impulse response by passing $V_i(s) = 1$ in the `lowpass` function, and then using this impulse response along with the input $v_i(t)$ in the function `scipy.signal.lsim` in order to obtain the **convolution** or the output $v_o(t)$.

This is done in the code below.

**Relevant Code:**

```
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)
v0,Vo=transfer(V)
t=np.linspace(0,1e-2,1e+5)
vi=(np.sin(2000*m.pi*t)+np.cos(2*10**6*m.pi*t))*np.heaviside(t,0)
t,vo,svec=sp.lsim(v0,vi,t)
```

We then plot this output $v_o(t)$.

**Plots Generated:**

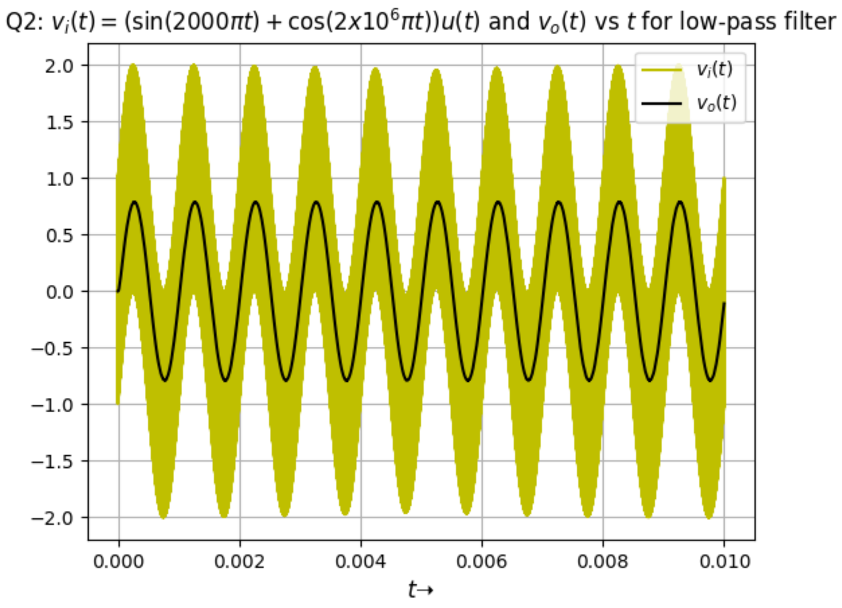

Figure 4: $v_i(t) = (\sin(2000\pi t) + \cos(2 \times 10^6 \pi t))u(t)$ and $v_o(t)$ vs $t$ for low-pass filter

## 2.3 Question 3

The next circuit that we consider is the high-pass filter.

The values taken are $R_1 = R_2 = 10k\Omega, C_1 = C_2 = 1nF$, and $G = 1.586$ as before. The circuit equations are:

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

$$V_p = V_1 \cdot \frac{sC_2R_3}{1 + sC_2R_3} \tag{6}$$

$$V_o = G(V_p - V_m) \tag{7}$$

$$sC_1(V_i - V_1) + sC_2(V_p - V_1) + \frac{V_o - V_1}{R_1} = 0 \tag{8}$$

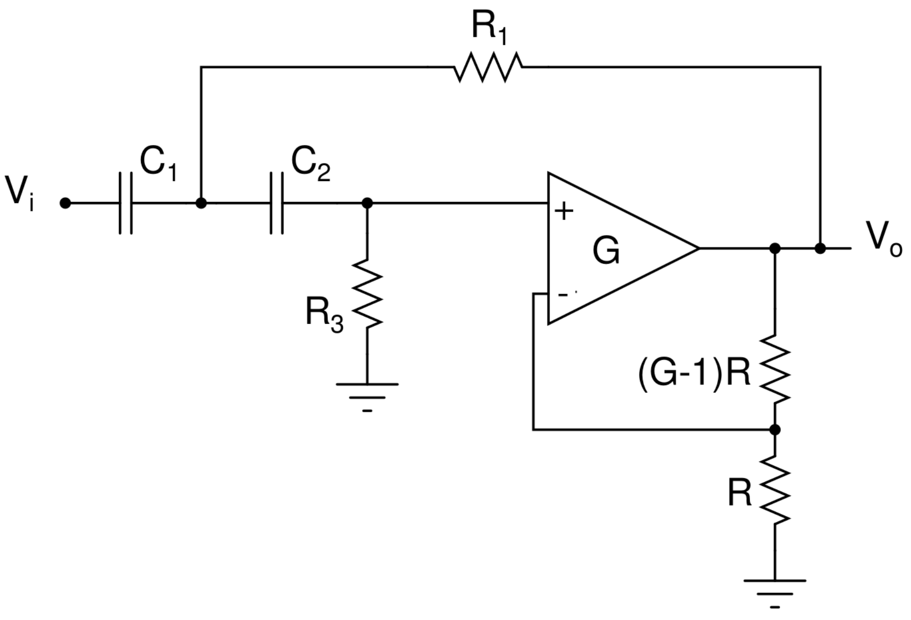

Figure 5: Butterworth high-pass filter

Putting the above results in a single matrix equation, we get:

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

$$A \cdot v = b$$

$$v = A^{-1} \cdot b$$

We solve the equation, and take the last element of the solution vector in order to get $V_o$.

**Relevant Code:**

We define a function `highpass` that does all the above for the high-pass filter.

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

We may plot the magnitudes of the *frequency-domain* step response *and* impulse response. This is implemented below:

```
A,b,V=highpass(10000,10000,1e-9,1e-9,1.586,1)
v0,Vo=transfer(V)
hf=lambdify(s,Vo,'numpy')
```

```
v=hf(ss)
p.loglog(w,abs(v),lw=2,label='Impulse response')
A,b,V=highpass(10000,10000,1e-9,1e-9,1.586,1/s)
v0,Vo=transfer(V)
hf=lambdify(s,Vo,'numpy')
v=hf(ss)
p.loglog(w,abs(v),lw=2,label='Step response')
p.xlabel('$\omega$\u279d', size=12)
p.ylabel('$|H(j\omega)|\u279d$', size=12)
p.title('Magnitude response of the step and impulse responses in
the frequency domain for high-pass filter')
p.legend(loc='upper right')
p.grid(True)
p.show()
```

**Plots Generated:**

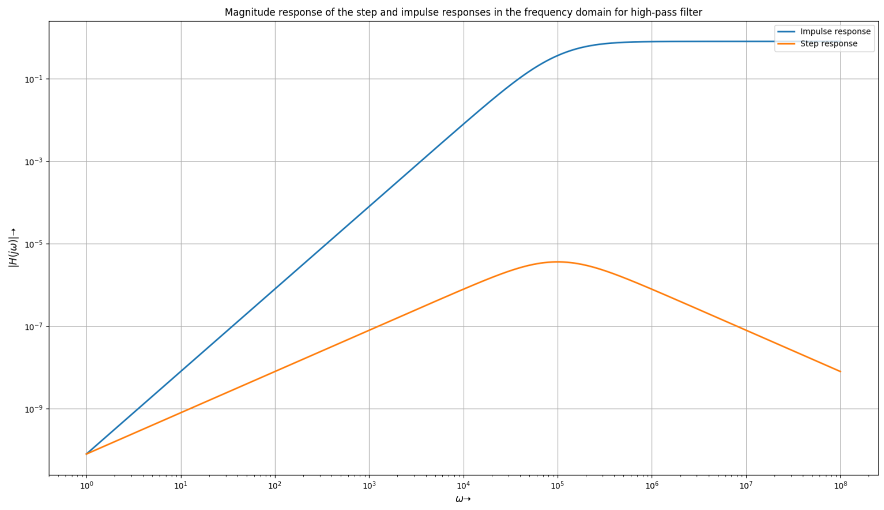

Figure 6: Magnitudes of frequency-domain step and impulse responses vs $\omega$ for high-pass filter

From the above transfer function, it is clear that the system is a high-pass filter.

## 2.4 Question 4

This question requires us to find the output of the high-pass filter to a damped sinusoid. The suitable $v_i(t)$ chosen to show the trend of the output is:

$$v_i(t) = e^{-5t}\sin(1000t)u(t)$$

This input is chosen as it reflects well the trends of interest to us.

The above input function has $\omega = 1000$. This is a low frequency according to the transfer function of the high-pass filter, therefore it is highly attenuated/blocked before passing. As a result, we get a nearly DC output.

**Relevant Code:**

```
A,b,V=highpass(10000,10000,1e-9,1e-9,1.586,1)
v0,Vo=transfer(V)
t=np.linspace(0,0.5,1e+5)
vi=np.sin(1000*t)*np.exp(-5*t)*np.heaviside(t,0)
t,vo,svec=sp.lsim(v0,vi,t)
```

We once again use `scipy.signal.lsim` to use the transfer function and the input signal to convolve for the output signal.

**Plots Generated:**

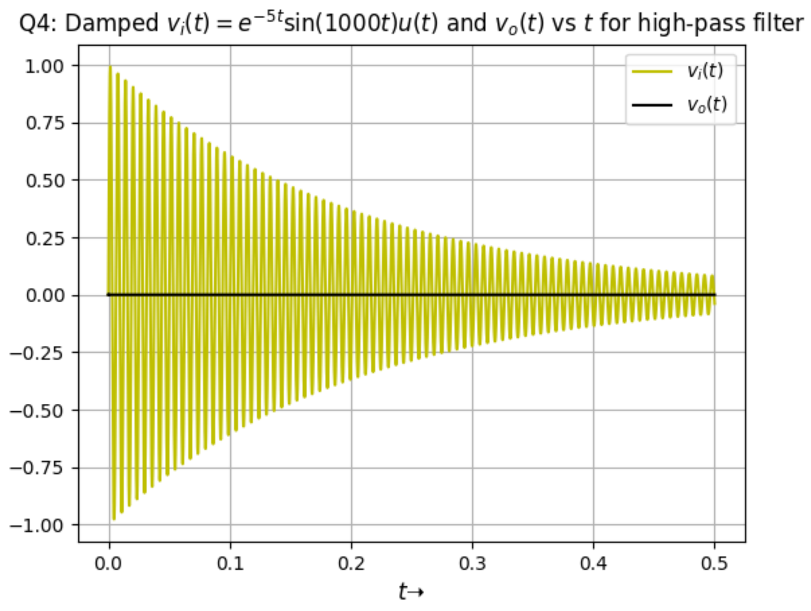

Figure 7: Damped $v_i(t) = e^{-5t}\sin(1000t)u(t)$ and $v_o(t)$ vs $t$ for high-pass filter

We can see above that the output is nearly a DC signal, because the input signal is of a small frequency. However, we get a tiny kink at $t = 0$ because the discontinuity is of a high frequency, and therefore we obtain it in the output:

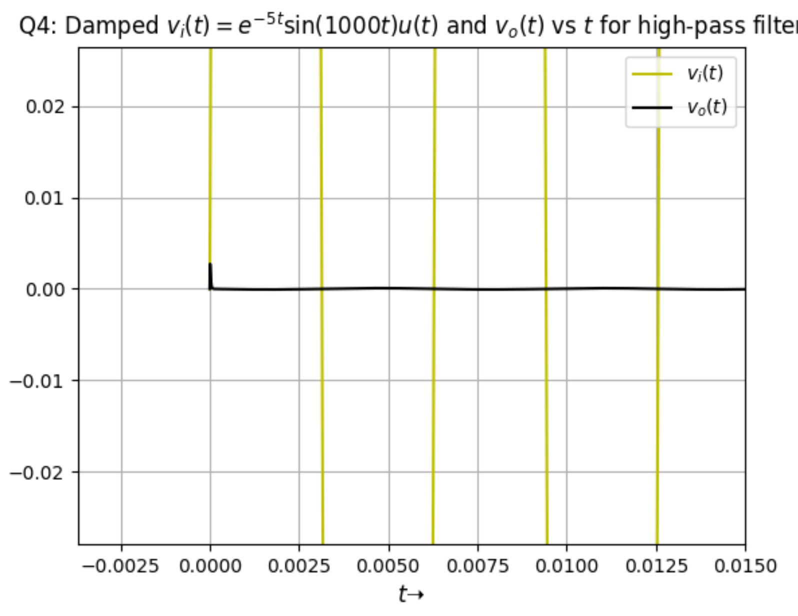

Figure 8: Damped $v_i(t) = e^{-5t}\sin(1000t)u(t)$ and $v_o(t)$ vs $t$ for high-pass filter (ZOOMED IN)

We now try increasing the frequency $\omega$ of the input damped sinusoid. Hence, we take:

$$v_i(t) = e^{-5t}\sin(10^5 t)u(t)$$

and get the following plot:

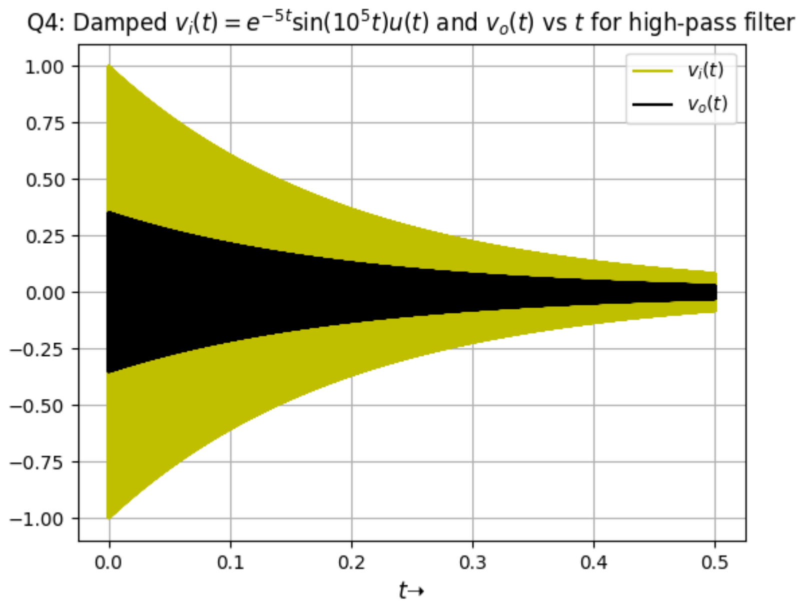

Figure 9: Damped $v_i(t) = e^{-5t}\sin(10^5 t)u(t)$ and $v_o(t)$ vs $t$ for high-pass filter

Next, we take:

$$v_i(t) = e^{-5t}\sin(10^6 t)u(t)$$

and get:

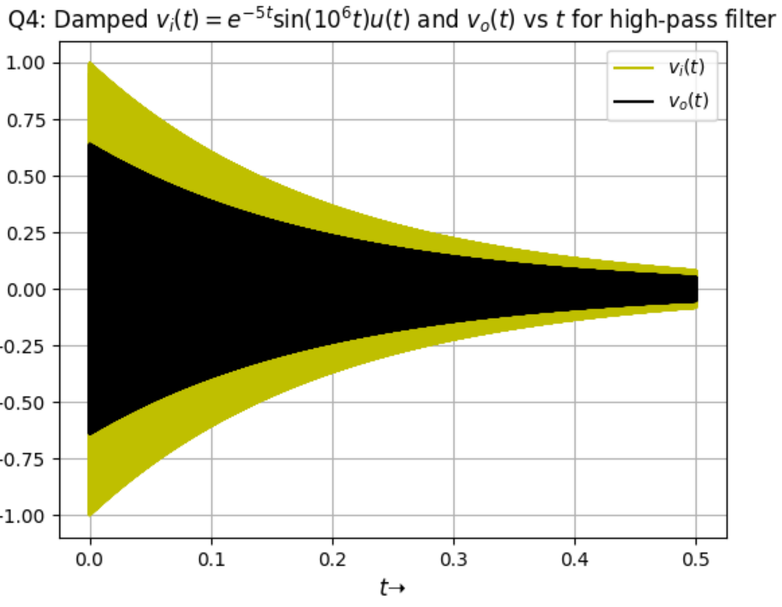

Figure 10: Damped $v_i(t) = e^{-5t}\sin(10^6 t)u(t)$ and $v_o(t)$ vs $t$ for high-pass filter

It is now an obvious observation that **higher frequencies of sinusoid pass through the high-pass filter without getting blocked out**. However they are attenuated because of the amplifier gain's effect (just like the low-pass filter case).

## 2.5 Question 5

The step response to the system is calculated by putting $v_i(t) = u(t)$ or, correspondingly, $V_i(s) = \frac{1}{s}$ in the matrix equation, so that the obtained $V_o$ represents the step response.

**Relevant Code:**

```
A,b,V=highpass(10000,10000,1e-9,1e-9,1.586,1/s)
v0,Vo=transfer(V)
t,vo=sp.impulse(v0,None,np.linspace(0,0.005,501))
```

**Plots Generated:**

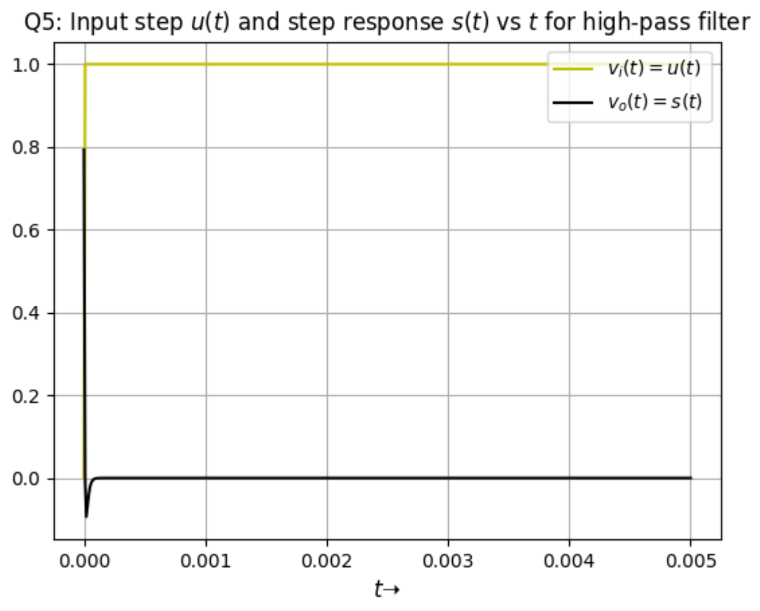

Figure 11: Input step $u(t)$ and step response $s(t)$ vs $t$ for high-pass filter

We notice that the step response forms a kink, unlike the neat rise of the step input. This is because this point is a discontinuity. Ideally, it requires infinite sinusoidal components of all frequencies in order to be represented accurately. However, **since our system is a high-pass filter, it does not allow lower frequencies which are needed to accurately account for the jump.** This is why the response cannot be softened by a low-frequency component, and is a victim of a high-frequency jump, and as a result it overshoots.

The value of the output is 0 beyond this point, as the step input is now effectively a DC input (the frequency is 0). Since the system is a high-pass filter, it completely blocks out this DC and gives a 0 output.

## 3 Conclusion

From this Assignment, one learns that:

- The Laplace Domain is a powerful tool to solve complex circuits that have capacitive and inductive impedances.
- `sympy` has an extremely powerful symbolic algebra toolkit, which can easily create, manipulate and solve for polynomials, matrices, etc. in their written form as they are.
- Butterworth filters are filters that are designed to keep the frequency response as flat as possible in the passband. They are also called maximally flat magnitude filters.
- Kirchoff's equations in the Laplace Domain, in the form of a matrix equation can be easily solved by a program to obtain the voltages in a circuit.

******************************