EE2703 : Applied Programming Lab
Assignment 7
Analysis of circuit using Laplace trasforms and Symbolic Algebra

Adityan C G EE19B003

May 2021

Introduction

The main objective of this assignment is to use the symbolic algebra and scipy tools of python to analyse circuits.

Low-Pass Filter

In this section, we analyse the following circuit:

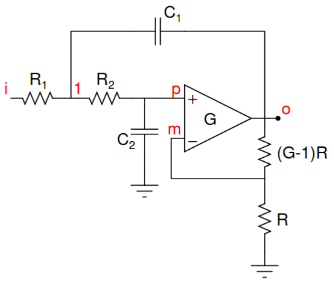


Figure 1: Op-Amp based Low-Pass Filter

Writing KCL equations (in s-domain) of the nodes marked on the figure, we get the following matrix:

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

`Sympy` allows us to create matrices with symbolic entries, and also perform mathematical operations on them, as if they were `numpy` arrays.

Solving for $V_o(s)$, (with the above given values) we get:

$$V_o(s) = \frac{-0.0001586 \cdot V_i(s)}{2 \times 10^{-14}s^2 + 4.414 \times 10^{-9}s + 0.0002} \tag{2}$$

`Sympy` allows us to convert a symbolic expression into functions that are compatible with other packages like `numpy`, `scipy` etc. This can be accomplished by converting the expression into a *lambda*[1] function. For this purpose, `sympy` has a method called *lambdify*, which takes in as arguments, the symbols in the expression (which are to be treated as variables), the expression itself, and also the package with which the resulting function has to be compatible with.

However, since we are required to use the `scipy.signal` toolbox, we have to convert the above the symbolic expression to a format with which we can easily create a `signal.lti` object. For

[1]Lambda functions are, in a nutshell, *one-time use* functions. They are short-lived functions, very useful for quick manipulations, especially inside other functions.

that, we extract the coefficients of the numerator and denominator polynomials of $V_o(s)$ and create a `signal.lti` object using the same. The below piece of code accomplishes this: [breaklines, linenos]pythoncode/sympyToLTI.py

Now, we can easily create a `signal.lti` object with the coefficients we got, and use `signal.bode` to obtain the Bode magnitude and phase plots, which are shown below.

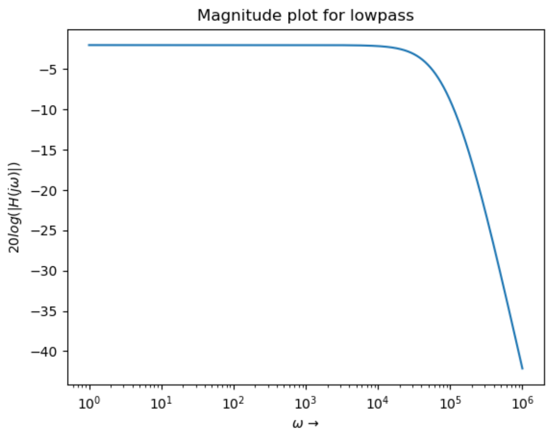


Figure 2: Magnitude plots of LPF

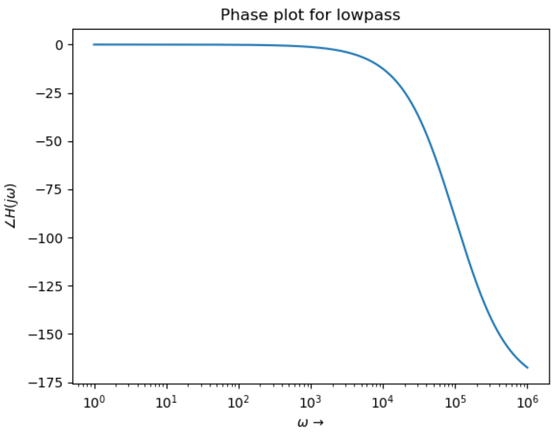


Figure 3: Phase plot of LPF

To see the step response of the system, we can use `signal.step`. The step response of the system is shown below.

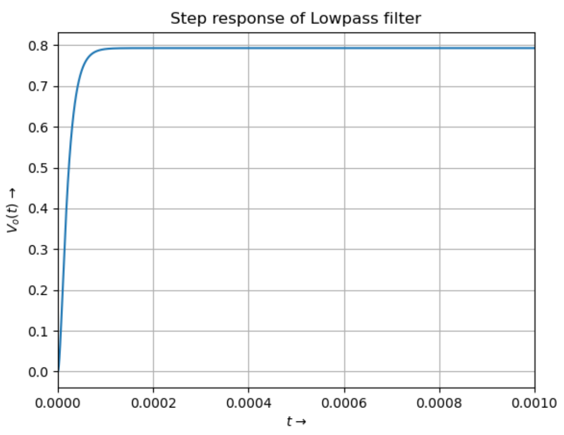


Figure 4: Step response of LPF

Now, we shall see that the circuit is indeed a low-pass filter by plotting the output for a mixed-frequency input, which has both high frequency and low frequency components.

We shall give the following input to the filter:

$$V_i(t) = (sin(2\pi \times 10^3 t) + cos(2\pi \times 10^6 t))u(t) \tag{3}$$

We shall use `signal.lsim` to calculate the time-domain response of the system.

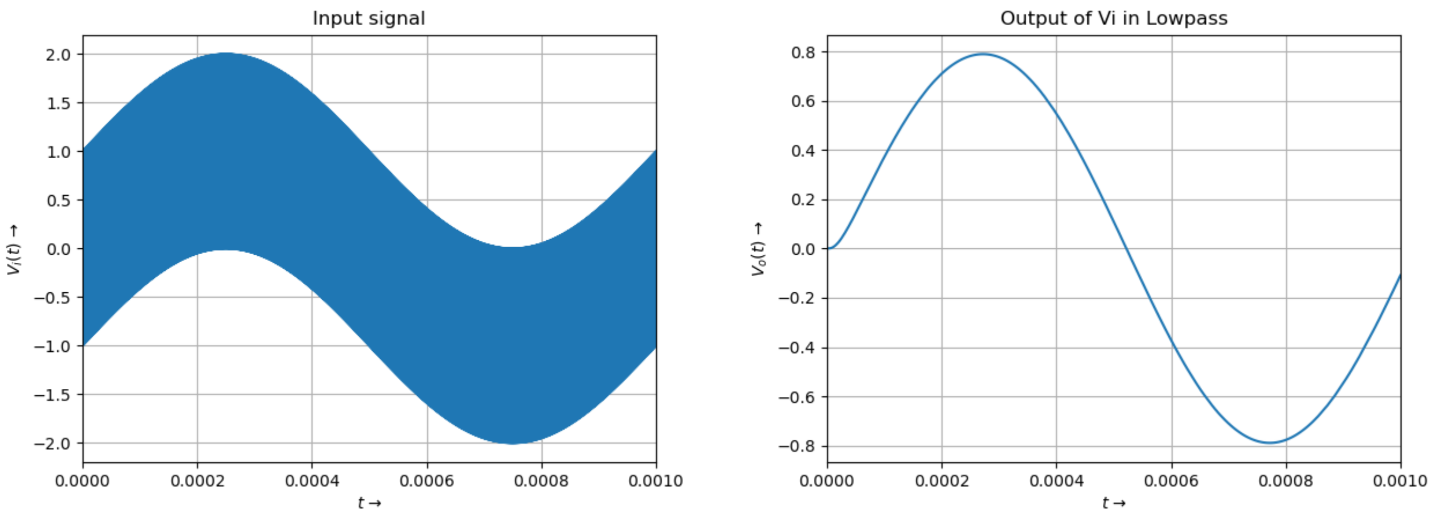


Figure 5: *Left*: Mixed frequency input*Right*: Filtered Output

We can easily see that the output contains only the low frequency component of the input (1 KHz sinusoid). Thus, the circuit is a low-pass filter. It's cut-off frequency for the values of R_1, R_2, C_1, C_2 used is $\frac{1}{2\pi}$ MHz.

High-Pass Filter

We shall now look at a slightly modified version of the above circuit.

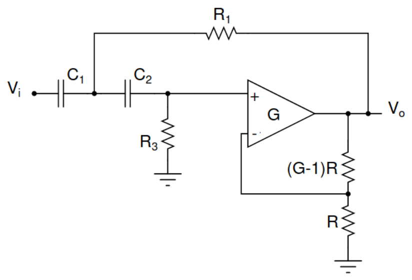


Figure 6: Op-amp based High-Pass Filter

Performing a similar procedure like before, we get the KCL matrix as:

$$\begin{pmatrix} 0 & 0 & 1 & \frac{-1}{G} \\ \frac{sR_3C_2}{1+sR_3C_2} & 0 & -1 & 0 \\ 0 & -G & G & 1 \\ -\frac{1}{R_1} - sC_2 - sC_1 & 0 & sC_2 & \frac{1}{R_1} \end{pmatrix} \begin{pmatrix} V_1(s) \\ V_p(s) \\ V_m(s) \\ V_o(s) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -sC_1V_i(s) \end{pmatrix} \tag{4}$$

Solving it for $V_o(s)$, we get,

$$V_o(s) = \frac{1.586 \times 10^{-14} s^2 \cdot V_i(s)}{2 \times 10^{-14} s^2 + 4.414 \times 10^{-9} s + 0.0002} \tag{5}$$

Now, we can easily create a `signal.lti` object with the coefficients we got, and use `signal.bode` to obtain the Bode magnitude and phase plots, which are shown below.

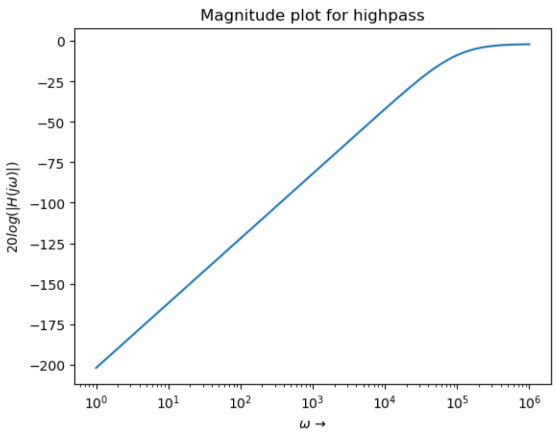


Figure 7: Magnitude plots of HPF

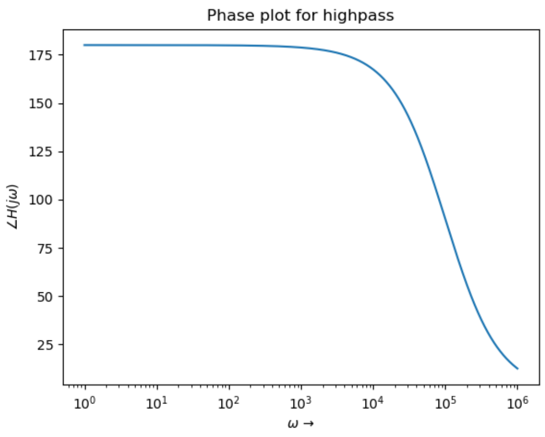


Figure 8: Phase plot of HPF

Now, we shall see that the circuit is indeed a high-pass filter by plotting the output for a mixed-frequency input, which has both high frequency and low frequency components.

We shall use `signal.lsim` to calculate the time-domain response of the system.

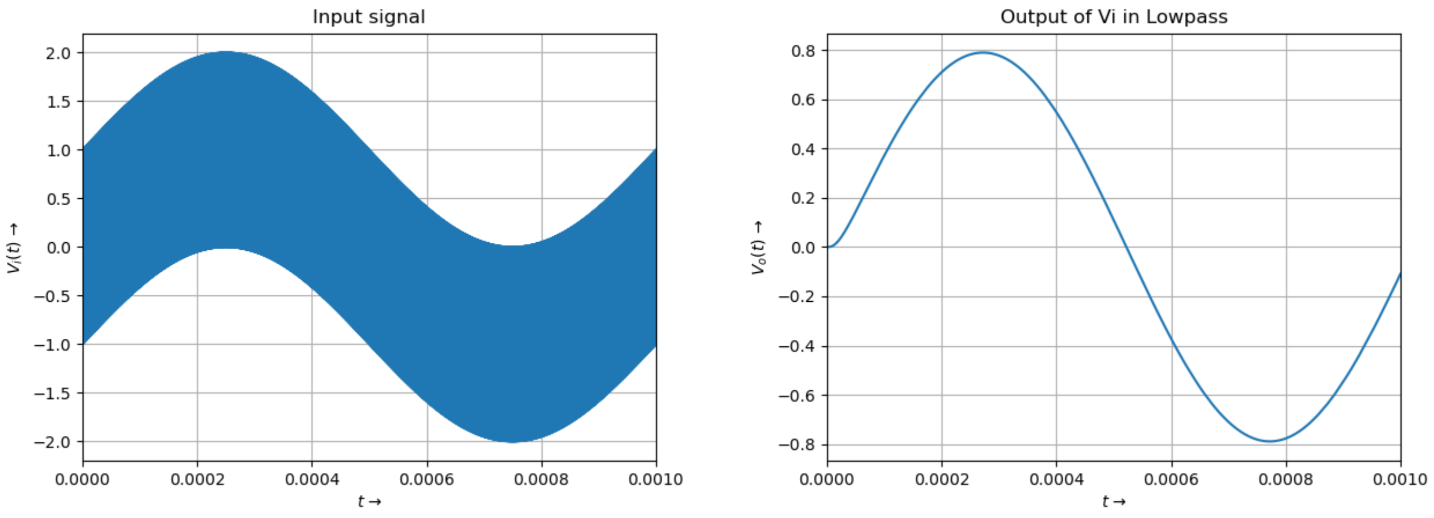
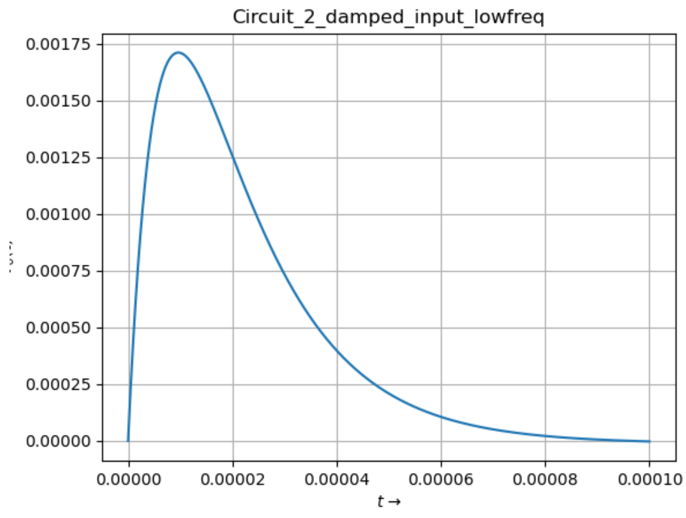


Figure 9: *Left*: Mixed frequency input*Right*: Filtered Output

We can easily see that the output contains only the high frequency component of the input (1 KHz sinusoid). Thus, the circuit is a high-pass filter.

Now for Q.No.4 , we will see the decay output of the highpass filter that we built, separately for **LOW** and **HIGH** frequency inputs. Those are as shown below for frequencies 2*π and 2e5*π



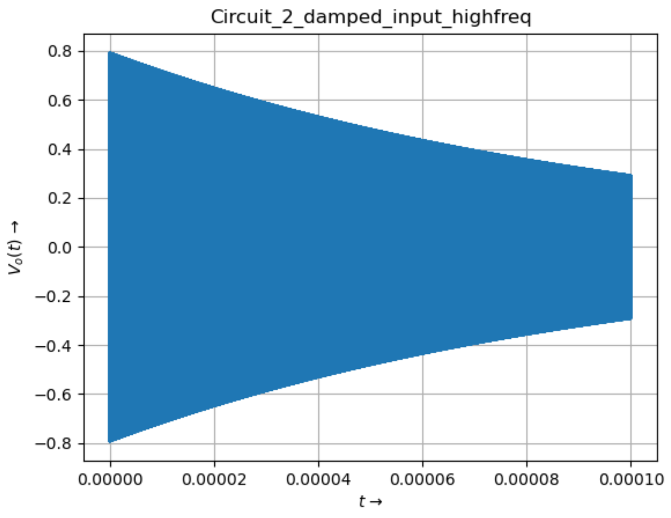


Figure 10: *Left*: Mixed frequency input*Right*: Filtered Output

For Q.No.5 , To see the step response of the system, i.e. $V_i = 1/s$, we can use `signal.step`. The step response of the system is shown below.

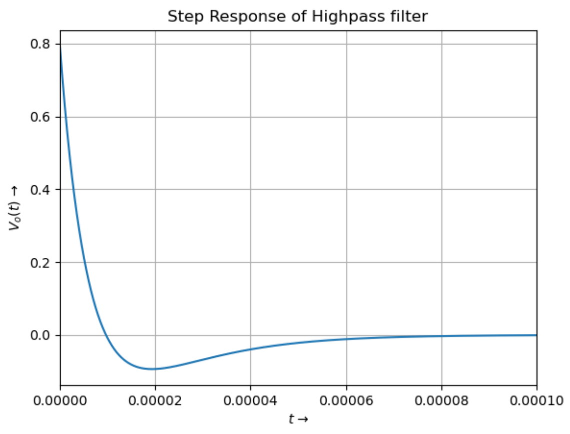


Figure 11: Step response of HPF

Conclusion

`Sympy` provides a convenient way to analyse LTI systems using their Laplace transforms. The toolbox was used to study the behaviour of a low pass filter,implemented using an op-amp of gain G. For a mixed frequency sinusoid as input, it was found that the filter suppressed the high frequencies while allowing the low frequency components. Similarly, a high pass filter was implemented using an op-amp with the same gain. The magnitude response of the filter was plotted and its output was analysed for damped sinusoids.The step response of the filter was found to have a non-zero peak at $t = 0$,due to the sudden change in the input voltage.