Assignment 8

Lakshya J ee19b035

April 22nd, 2021

Introduction

In this code we try to find the transfer function of the output for various inputs for a high pass and low pass filter. I start by importing these libraries.

```python
from sympy import *
import scipy.signal as sp
import matplotlib.pyplot as plt
from sympy.utilities.lambdify import lambdify
import numpy as np
import warnings
```

Q1

I begin by simulating the Low Pass Filter circuit as given by sir. First I plot the transfer function of the low pass filter with the code as given provided by sir.
Now I attempt to find the time varying response using *scipy.signal.lsim*. I begin by generating a time vector t and simplify the response V_o.
Using the following code I extract the numerator and denominator which I then feed into the *scipy.signal.lti* command. This generated a transfer function which can be fed into *scipy.signal.lsim* to generate the time response.

```python
Vo = Vo.simplify()
# Using Poly from the sympy libraray to extract the coefficients of
     numerator and denominator
num = Poly(Vo.as_numer_denom()[0], s).all_coeffs()
den = Poly(Vo.as_numer_denom()[1], s).all_coeffs()
# Converting the extracted coefficients to float type
num = np.array(num, dtype = float)
den = np.array(den, dtype = float)
# using lti to generate the transfer function
H = sp.lti(num, den)
```

Here is the plot obtained:

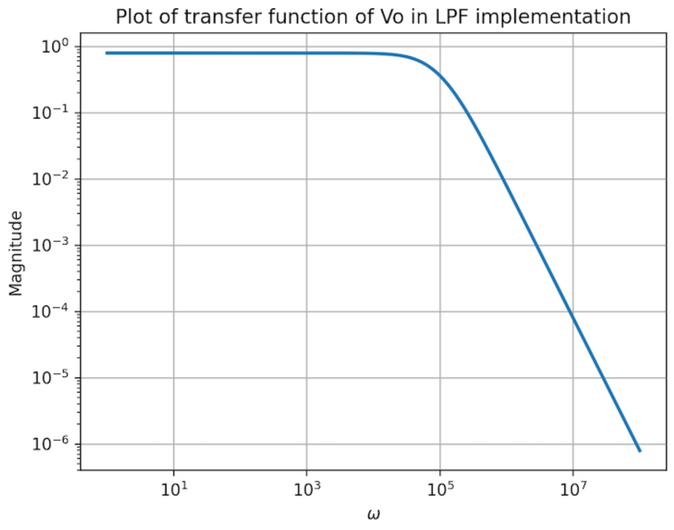

Figure 1: Transfer function of low pass filter

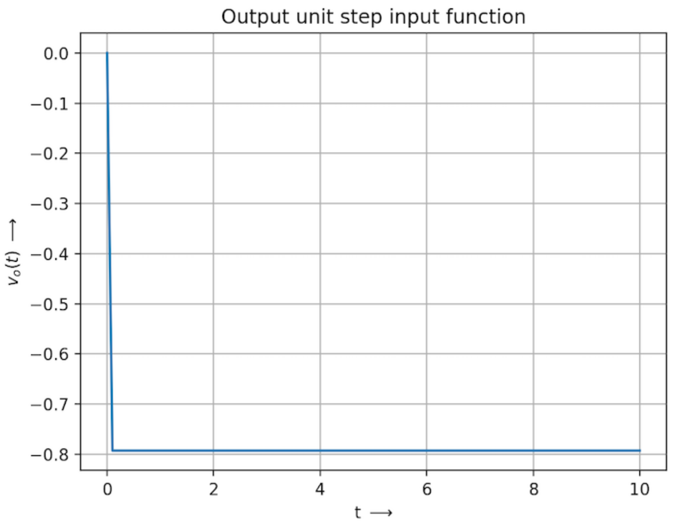

Figure 2: Time varying responce to step input

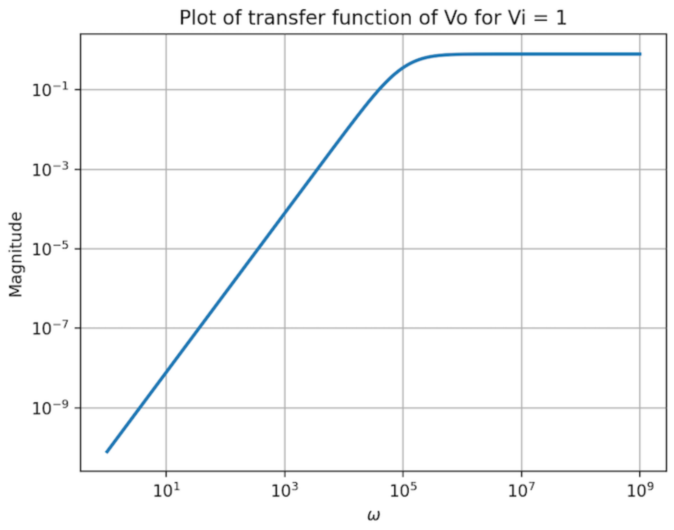

Figure 3: Transfer Function of High Pass Filter

Q2, Q3, Q4, Q5

Now I will solve the circuit for the high pass filter. On analyzing it can be noticed that we can use the same matrix by just replacing all the resistor components with capacitors and vice versa. Here is function implemented to create the high pass filter

```python
def highpass(R1, R3, C1, C2, G, Vi):
  s = symbols('s')
  A = Matrix([[0,0,1,-1/G],[-1/(1+(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)
```

I will now plot the transfer functions for the various inputs. I do so by multiplying the laplace transforms of the various inputs with the obtained transfer functions.
First I plot the transfer function for a constant input of 1. Now I am plotting I obtain the transfer function for the input $V_i = cos(2 * 10^6 \pi t) + sin(2000\pi t)$.
Here is the code for the same:

```python
f = cos(2e6*np.pi*t) + sin(2000*np.pi*t)
vi_L = laplace_transform(f, t, s, noconds=True)
A,b,V_ = highpass(10000, 10000, 1e-9, 1e-9, 1.586, vi_L)
```

I pass the laplace transform as the input function for the high pass filter method. This is the plot I obtained:

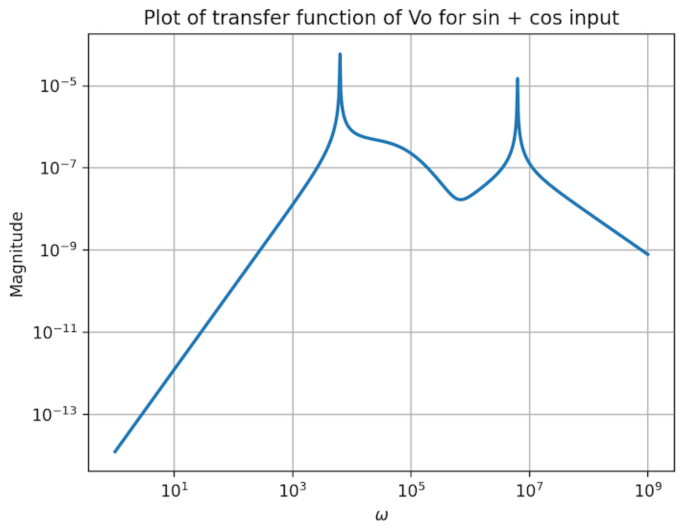

Figure 4: Transfer Function of High Pass Filter for $V_i = cos(2 * 10^6 \pi t) + sin(2000\pi t)$

Similarly I plot the transfer function for a decaying sinusoid. The decaying sinusoid input I took is $V_i = cos(2000\pi t) * exp(-0.5 * t)$.

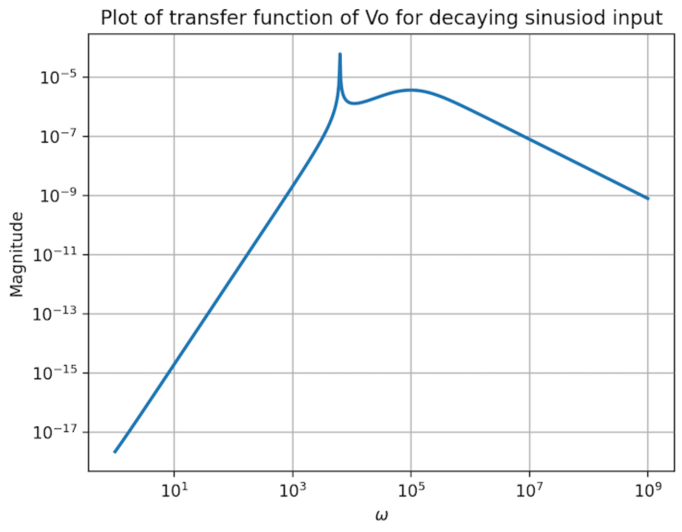

Figure 5: Transfer Function of High Pass Filter for $V_i = cos(2000\pi t) * exp(-0.5 * t)$

Lastly I plot the transfer function for a unit step input. The laplace transform for the step function is $\frac{1}{s}$. Here is the plot of the same.

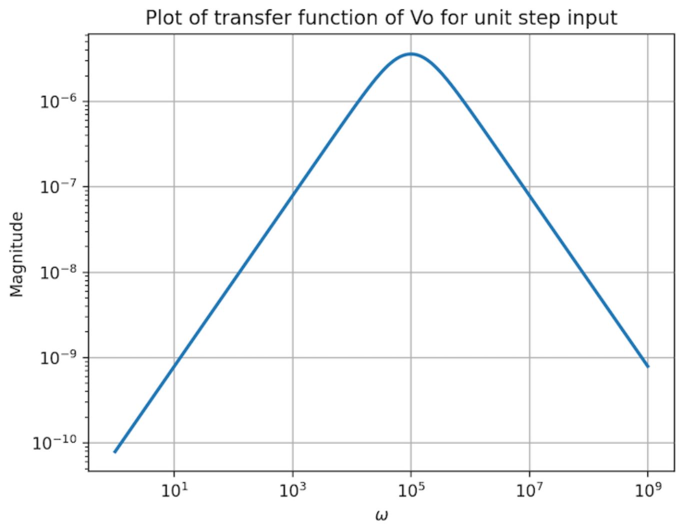

Figure 6: Transfer Function of High Pass Filter for $V_i = u(t)$

After plotting the transfer functions I plot the time varying responses using *scipy.signal.lsim*.

```python
# Using simplify to make the function easier to read
Vo = Vo.simplify()
# Using Poly from the sympy libraray to extract the coefficients of
    numerator and denominator
num = Poly(Vo.as_numer_denom()[0], s).all_coeffs()
den = Poly(Vo.as_numer_denom()[1], s).all_coeffs()
# Converting the extracted coefficients to float type
num = np.array(num, dtype = float)
den = np.array(den, dtype = float)
```

I use V_o transfer function obtained from the constant input of 1. Then I find the laplace transformation of each of the inputs and mutiply it with the transfer function V_o, numerator and denominator separately.

```python
# Finding the laplace transforms of the input function
f = cos(2e6*np.pi*t) + sin(2000*np.pi*t)
vi_L = laplace_transform(f, t, s, noconds=True)
vi_L = vi_L.simplify()
# Writing the input as a polynomial
num1 = Poly(vi_L.as_numer_denom()[0], s).all_coeffs()
den1 = Poly(vi_L.as_numer_denom()[1], s).all_coeffs()
# Converting coefficients to float data type
num1 = np.poly1d(np.array(num1, dtype = float))
den1 = np.poly1d(np.array(den1, dtype = float))
# Multiplying the numerators and denominators
numf = np.polymul(num, num1)
denf = np.polymul(den, den1)
# using lti to generate the transfer function
```

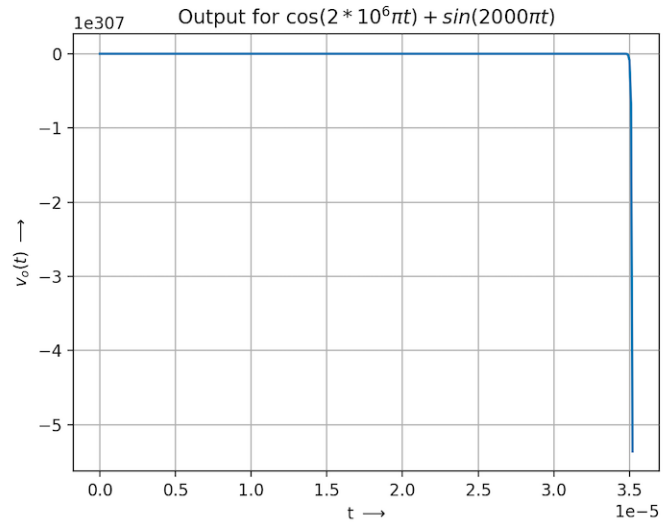

Figure 7: Response for $V_i = cos(2*10^6\pi t) + sin(2000\pi t)$

```
H1 = sp.lti(numf, denf)
```

Hence I obtain the net transfer function after passing it to the *scipy.signal.lti* command. I then use *scipy.signal.lsim* to obtain the time varying response as follows:

```
time, y, svec = sp.lsim(H1, np.ones(time.shape), time)
```

I do these for each of the functions and plot the resulting time varying inputs. Here are the plots I obtained.
In this output we see a sudden dip in the output voltage and then it settles to 0. From the transfer function it can be inferred that maximum gain be observed only around $\omega = 10^5$. Since the input does not have any such frequency component, the output is observed to be zero.

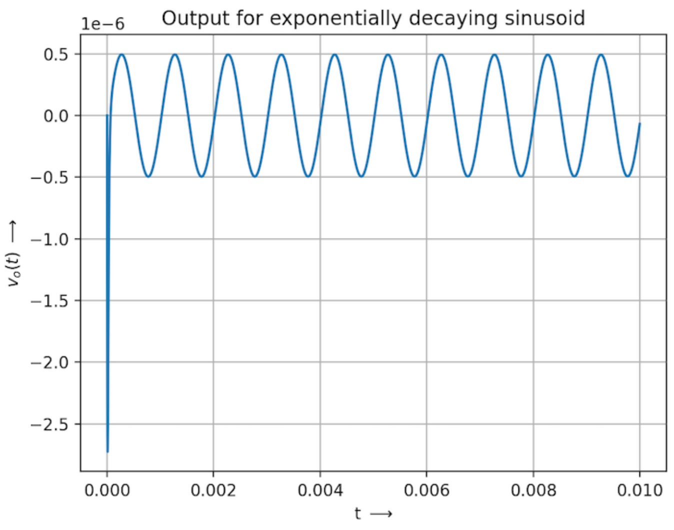

Figure 8: Response for $V_i = cos(2000\pi t) * exp(-0.5 * t)$

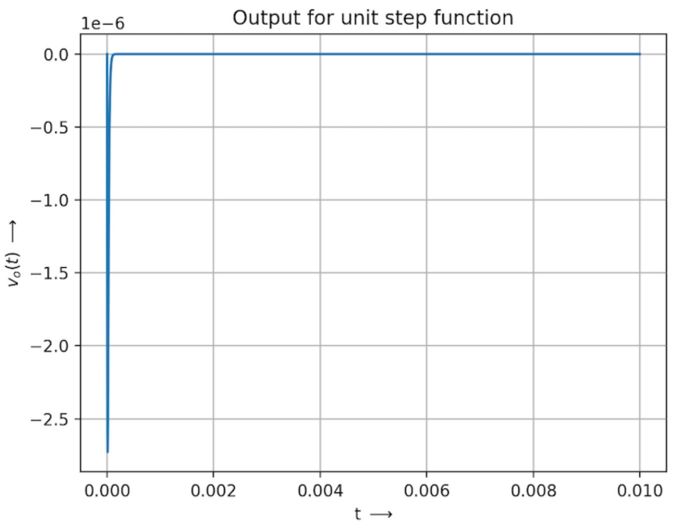

Figure 9: Response for $V_i = u(t)$