EE2703 : Applied Programming Lab
Assignment 7
Analysis of circuits using Laplace transforms

Debojyoti Mazumdar
EE20B030

April 7 , 2022

0.1 Abstract

Aim of this assignment is to understand how to use simpy library to analyze filters.

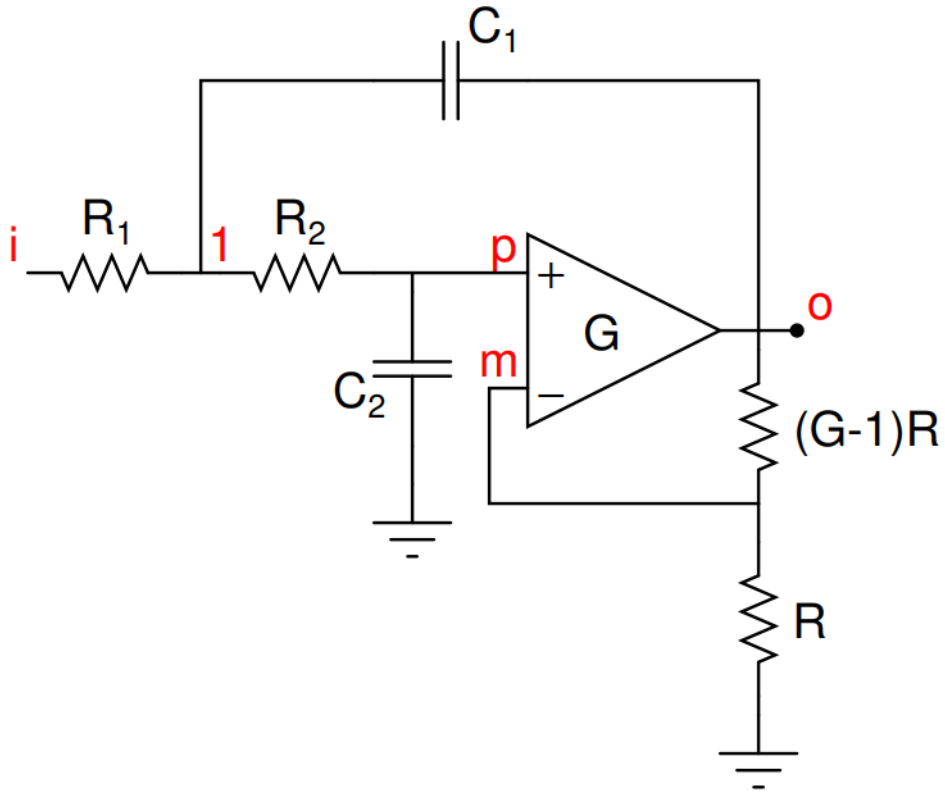
0.2 Introduction

In this assignment, we will analyze the second order high-pass and the low-pass filters using simpy library coupled with scipy library. Simpy library is a powerful python library that helps us to plot accurate graphs using modified nodal analysis equations.

0.3 Assignment

0.3.1 Question 1

We will be using the following second order Low-pass filter.



The matrices of the modified nodal analysis of the above circuit in the Laplace domain will be:

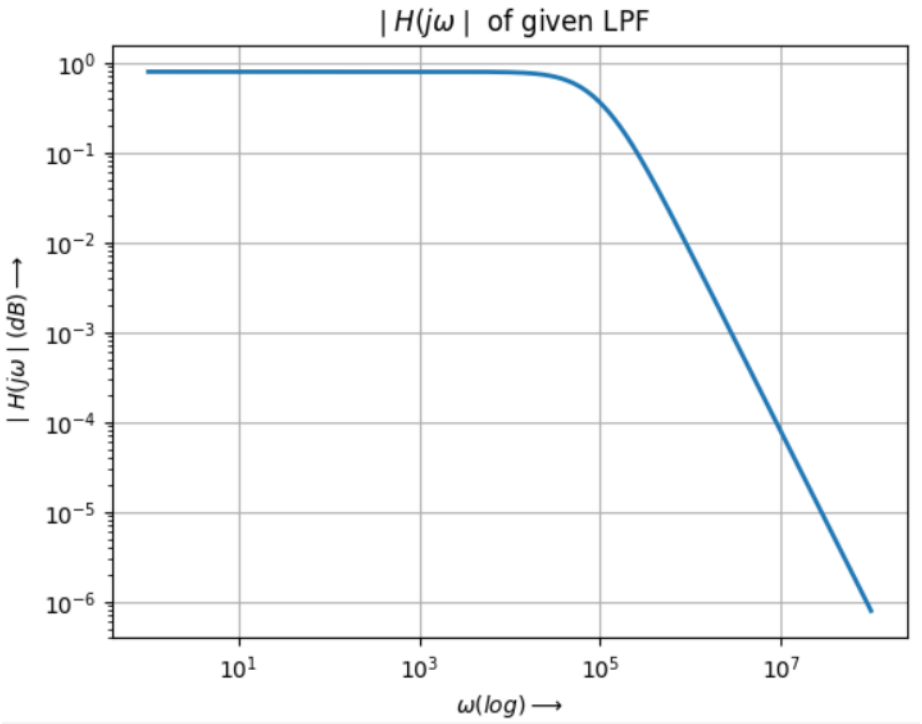
$$\begin{pmatrix} 0 & 0 & 1 & -\frac{1}{G} \\ -\frac{1}{1+sR_2C_2} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ -\frac{1}{R_1}-\frac{1}{R_2}-sC_1 & \frac{1}{R_2} & 0 & sC_1 \end{pmatrix} \begin{pmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ V_i(s)/R_1 \end{pmatrix}$$

We will be using the above matrices to find the transfer function using the following code.

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

def transfer_function_of_LPF(Vi=1, R1: float = 10e3, R2: float = 10e3, C1: float = 1e-9,
    A, b, V = lowpass(R1, R2, C1, C2, G, Vi)
    V = A.inv()*b
    Vo = V[3]
    print(Vo)
    ww = p.logspace(0, 8, 801)
    ss = 1j*ww
    s = sm.symbols("s")
    hf = sm.lambdify(s, Vo, "numpy")
    v = hf(ss)
    p.loglog(ww, abs(v), lw=2)
    p.grid(True)
    p.show()
    return Vo
```

The transfer function plot of the low pass filter is the following.



The above transfer function generated will now be converted to scipy.signal type by using the following function.

```
def symExpToLTI(exp, s=sm.symbols('s')):
    exp_num, exp_den = exp.as_numer_denom()
    num_coeffs = sm.Poly(exp_num, s).all_coeffs()
    den_coeffs = sm.Poly(exp_den, s).all_coeffs()
    num = p.poly1d([float(n) for n in num_coeffs])
    den = p.poly1d([float(n) for n in den_coeffs])
    return sp.lti(num, den)
```

Now we will define the unit step function.

```
def u(t):
    return np.where(t > 0, 1, 0)
```

using the above functions we will now plot the step response of the transfer function of the Low-pass filter using the following code.

```
s = sm.symbols("s")
Vo = transfer_function_of_LPF()
H = symExpToLTI(Vo)
u_ = u(t)
_, u_resp, _ = sp.lsim(H, u_, t)
p.plot(t, u_resp)
p.title('Step response of given LPF')
p.xlabel(r'time $\longrightarrow$')
p.ylabel("output")
p.show()
```

running the above code we get the following as the step response.

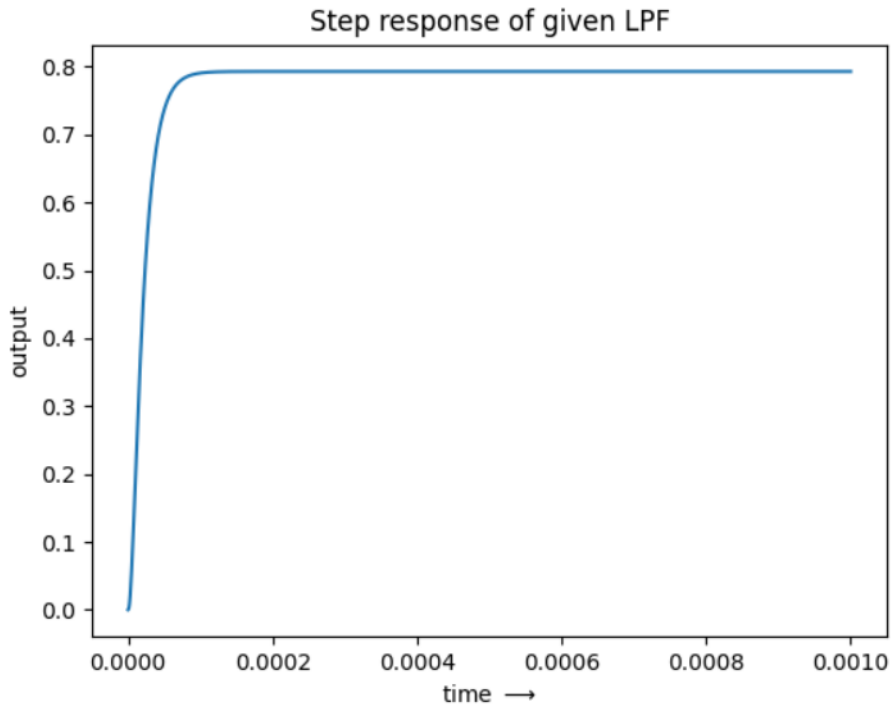


Figure 1: plot of output vs time for Low-pass filter for unit step input

0.3.2 Question 2

Now we define the given sinusoidal input.

```
def vi(t):
    return (np.sin(2e3*np.pi*t) + np.cos(2e6*np.pi*t))*(t > 0)
```

then we again plot the output but with vi as input.

```
s = sm.symbols("s")
Vo = transfer_function_of_LPF()
H = symExpToLTI(Vo)
_, vi_resp, _ = sp.lsim(H, vi(t), t)
p.plot(t, vi_resp)
p.title('Input response of the given LPF')
p.xlabel(r'time $\longrightarrow$')
p.ylabel("output")
p.show()
```

And we get the following as the output.

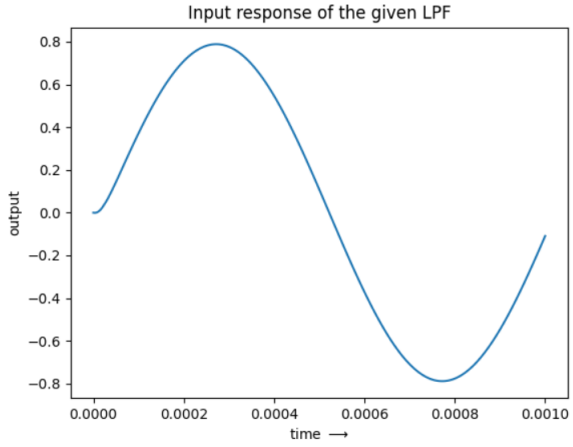
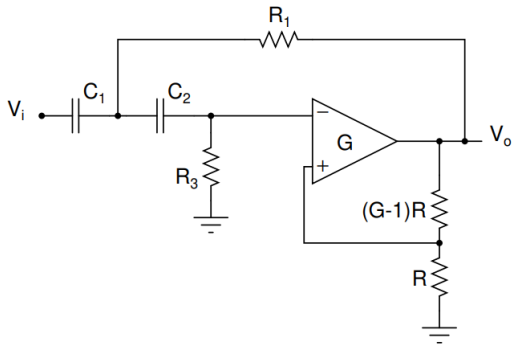


Figure 2: plot of output vs time for Low-pass filter

0.3.3 Question 3

Now we find out the nodal equations of the following second order High-pass filter circuit.



After finding the nodal equations of the above circuit we find out the transfer function of the above circuit using the following code.

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

def transfer_function_of_HPF(Vi=1, R1: float = 10e3, R3: float = 10e3, C1: float = 1e-9,
    A, b, V = highpass(R1, R3, C1, C2, G, Vi)
    V = A.inv()*b
    Vo = V[3]
    print(Vo)
    ww = p.logspace(0, 8, 801)
    ss = 1j*ww
    s = sm.symbols("s")
    hf = sm.lambdify(s, Vo, "numpy")
    v = hf(ss)
    p.loglog(ww, abs(v), lw=2)
    p.grid(True)
    p.show()
    return Vo
```

Running the above code we get the following transfer function graph.

0.3.4 Question 4

Now we define a damped sinusoidal input with frequency of 1e6 Hz and damping factor of 500.

```
def vi_ds(t, w: float = 1e6, a: float = 500):
    return np.where(t < 0, 0, np.sin(w*t)*np.exp(-1*a*t))
```

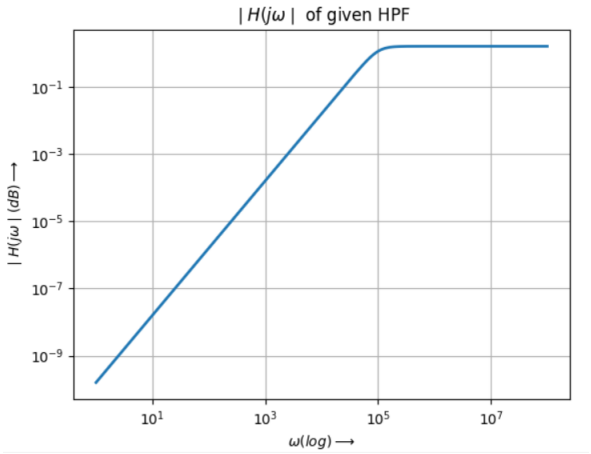


Figure 3: transfer function graph

We now find the output for Low-pass filter and High-pass filter for the above input using the code similar to the one used in question-1.
And we get the following outputs.

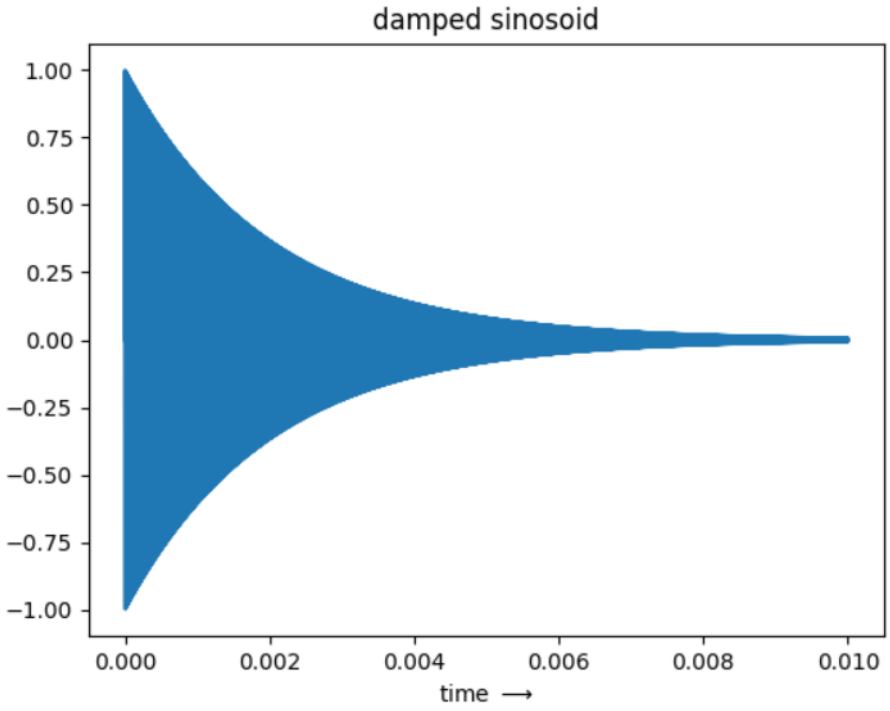


Figure 4: Damped sinosoid wave

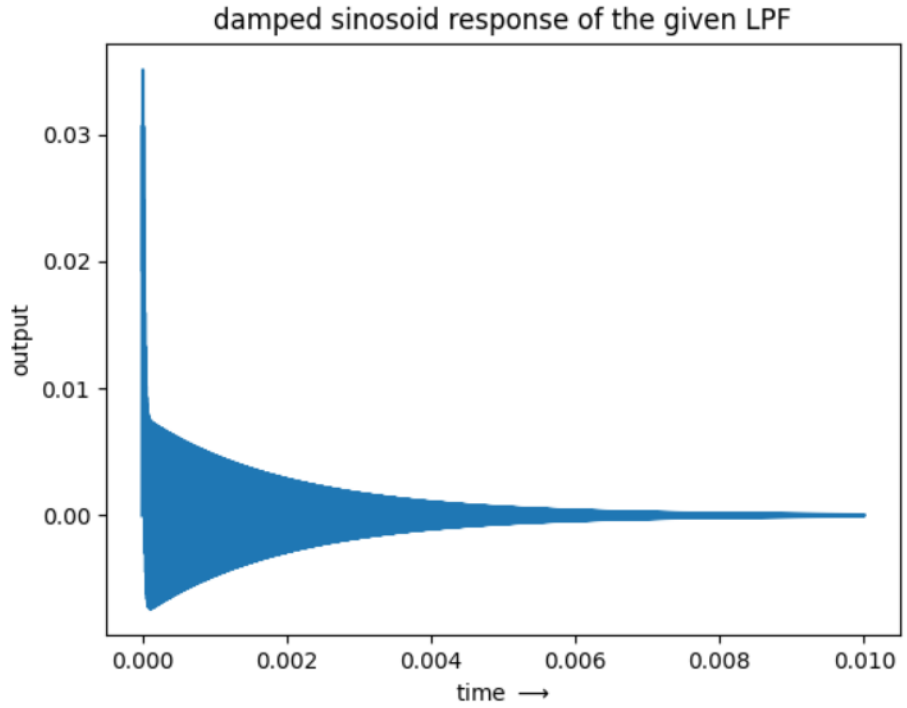


Figure 5: Damped sinosoid wave response for Low-pass filter

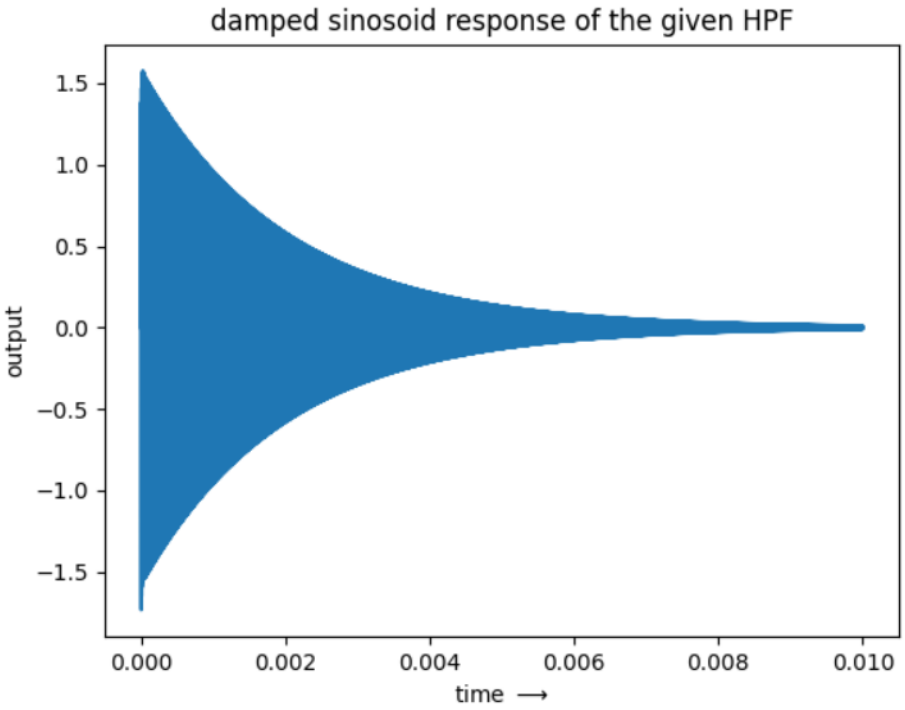


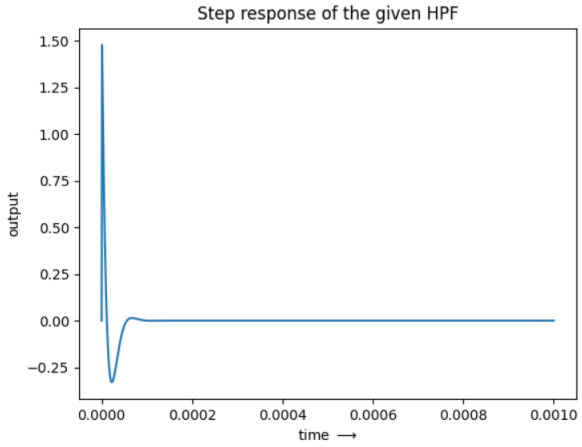
Figure 6: Damped sinosoid wave response for High-pass filter

0.3.5 Question 5

Now we have to plot the step response of the second order high pass filter. So we write the following code.

```
s = sm.symbols("s")
Vo = transfer_function_of_HPF()
H = symExpToLTI(Vo)
_, u_resp, _ = sp.lsim(H, u(t), t)
p.plot(t, u_resp)
p.title('Step response of the given HPF')
p.xlabel(r'time $\longrightarrow$')
p.ylabel("output")
p.show()
```

And we get the following graph as the step response.



0.4 Conclusions

We understood how to use the simpy library along with scipy.signal library to analyze second order High-pass filters and second order Low-pass filters.