Assignment 7: Circuit Analysis Using Sympy

Ayush Jamdar EE20B018

April 6, 2022

1 Aim

The goal of this assignment is to learn and apply two powerful capabilities of Python - Symbolic Algebra and Circuit analysis using Laplace transforms.

2 Introduction: A Circuit Analysis

To get a hang of sympy and recall the use of the `scipy.signal` toolkit, we start by analysing a Low Pass Filter realised with an active RC circuit shown in Figure 1.

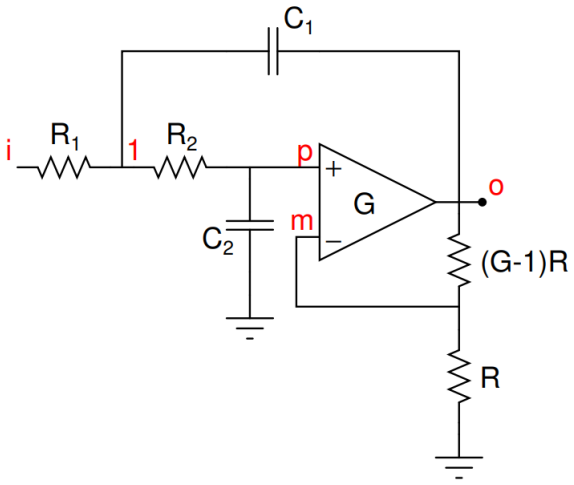

Figure 1: Low Pass Filter Circuit

Using KVL, we get

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

$$V_p = V_1(\frac{1}{1 + j\omega R_2 C_2}) \tag{2}$$

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

$$\frac{V_i - V_1}{R_1} + \frac{V_p - V_1}{R_2} + j\omega C_1(V_o - V_1) = 0 \tag{4}$$

This can be represented in a matrix as

$$\begin{bmatrix} 0 & 0 & 1 & \frac{-1}{G} \\ \frac{-1}{1+sR_2C_2} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ -\frac{R_1+R_2+sC_1R_1R_2}{R_1R_2} & \frac{1}{R_2} & 0 & sC_1 \end{bmatrix} \times \begin{matrix} V_1 \\ V_p \\ V_m \\ V_o \end{matrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ V_i(s)/R_1 \end{bmatrix} \tag{5}$$

Now to solve the matrix equation I implement the following function. Note that 's' has been used as a `symbol` in this function (for the variables) and the rest of the assignment too. The function basically takes the resistance and capacitance values, opamp dc gain and input voltage signal (which could be a function of 's', in general). Based on this, The A, b, V matrices satisfying $A.V = b$ are generated and returned.

```
def lowpass(R1, R2, C1, C2, G, Vi):
    # A*V = b
    # R1 R2 C2 C2 are with reference to Figure 1 of assignment
    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)
```

Lets find the frequency response to this filter. The input would be $\delta(s)$. In Laplace domain, it is just 1. The function returns a solution in the 's' space. It needs to molded into a function for further processing. The `lambdify` function treats `V_o` and returns a function. Then I convert s to $j\omega$ and plot the frequency response.

```
A, b, V = lowpass(10000, 10000, 1e-9, 1e-9, 1.586, 1)
Vo = V[3]  # output voltage in s domain
print('Vo(s) = {}'.format(Vo))
ww = np.logspace(0, 8, 801)
# frequency range under observation
ss = 1j * ww  # s = j\omega
transfer_func_lowpass = lambdify(s, Vo, 'numpy')
v = transfer_func_lowpass(ss)

plt.loglog(ww, abs(v), lw=2)
```

```
plt.title('Frequency Response of Lowpass Filter')
plt.xlabel('Frequencies on log scale')
plt.ylabel('Magnitude of H(j\omega) on a log scale')
plt.grid()
plt.show()
```

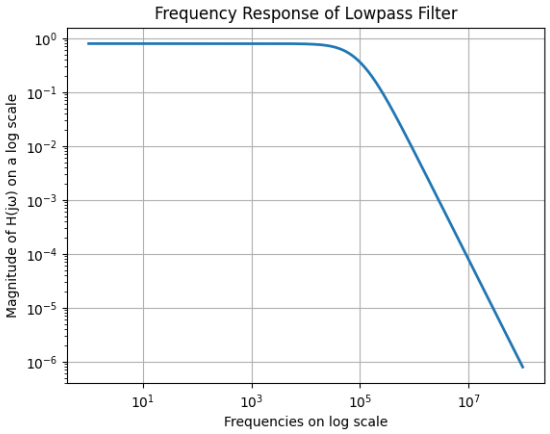

Figure 2: Low Pass Filter's Frequency Response

3 The Assignment

3.1 Question 1: Obtaining The Step Response

For the same low pass filter, I need to find the step response. The input signal will now be $V_i(s) = 1/s$. The `sp.impulse()` function operates on an LTI function to give the inverse Laplace Transform over the given time interval. Here `V_q1[3]` is the unit response in s domain. A challenge is to convert the `sympy` function to an LTI object. This is done by the function `convert_sympy_to_lti`. It parses the numerator and denominator polynomial coefficients and `splti` returns the transfer function.

```
def convert_sympy_to_lti(symbolic_exprn, s = symbols('s')):
    num, den = simplify(symbolic_exprn).as_numer_denom()
    # expressions in reduced form,
    then separate numerator and denominator
    # print(num,den)
    p_num_den = poly(num, s), poly(den, s)  # polynomials
    # print(p_num_den)
    c_num_den = [p.all_coeffs() for p in p_num_den]
    # coefficients
```

```
    # elements of c_num_den are <class 'sympy.core.numbers.One'>,
     not float
    l_num, l_den = [lambdify((), c)() for c in c_num_den]
    # convert to floats
    return sp.lti(l_num, l_den)
```

Now to get the time domain response, I need to use `sp.impulse()`, which works on an LTI object given by the above function. Then I plot the output.

```
conductance_matrix_q1, I_q1, V_q1 = lowpass(10000, 10000,
 1e-9, 1e-9, 1.586, 1/s)
print('Unit Response Vo(s) = {}'.format(V_q1[3]))
# Vo_func = lambdify(s,V_q1[3],'numpy')
t, unit_response_in_t = sp.impulse(convert_sympy_to_lti
   (V_q1[3]), None, np.linspace(0, 0.005, 1000))
plt.plot(t, unit_response_in_t, label='Output')
plt.plot(t,np.ones(1000),'red',label='Input')
plt.xlabel('Time')
plt.ylabel('voltage')
plt.title('Step Response of LPF')
plt.grid()
plt.legend()
plt.show()
```

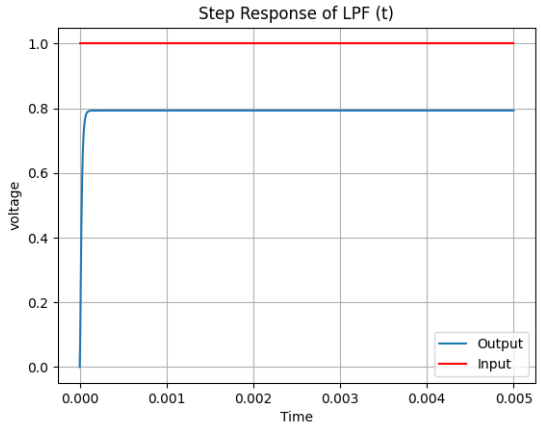

Figure 3: Step Response Of LPF

3.2 Question 2: Sinusoidal Input Signal

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

This signal is fed to the Low Pass Filter. I start by obtaining the transfer function from the introductory part, then perform the `sp.lsim` magic that processes the arguments - transfer function and time domain input over the given interval. The code:

```
t = np.linspace(0,0.005,100000)
Vi_t_q2 = np.sin(2000*np.pi*t) + np.cos(2e6*np.pi*t)
transfer_func_lowpass = convert_sympy_to_lti(Vo)
# reference to Question 0
t,Vo_t_q2,_ = sp.lsim(transfer_func_lowpass, Vi_t_q2, t)

plt.plot(t,Vi_t_q2,'red', label='input')
plt.plot(t, Vo_t_q2, label='output')
plt.xlabel('time')
plt.ylabel('voltage')
plt.title('LPF Input - Output')
plt.legend()
plt.grid()
plt.show()
```

In the plot, Figure 4, it can be seen that the high frequency signal is filtered out.

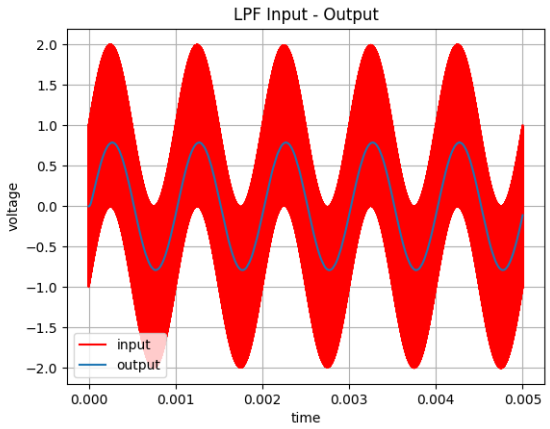

Figure 4: Question 2

3.3 Question 3: High Pass Filter

Now that we analysed the working of a low pass filter, its time to study the high pass filter. An active HPF circuit is shown in Figure 5. $R_1 = R_3 = 10kW, C_1 = C_2 = 1nF$, and $G = 1.586$ See Figure 5.

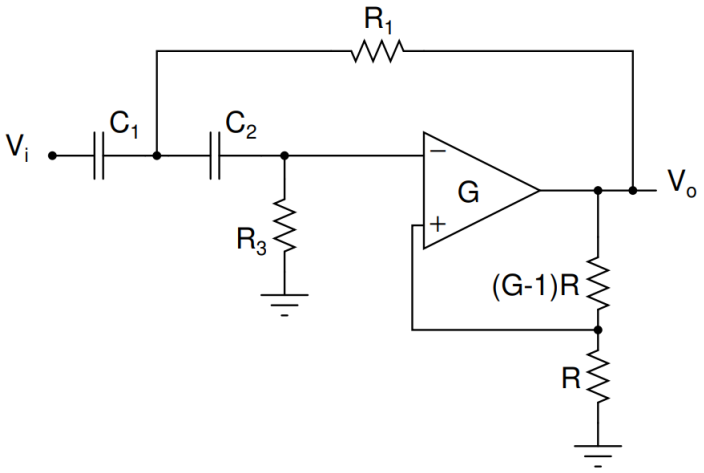

Figure 5: High Pass Filter

Start by defining a `highpass` function that solves the opamp circuit and returns the voltages at each node.

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

The HPF is characterized by its transfer function - the output voltage signal at $V_i(s) = 1$. `labmdify` converts the s symbolic function to a Python function. Then I plot it.

```
conductance_matrix_q3, variable_voltages_q3, b_q3 =
   highpass(1e4, 1e4, 1e-9, 1e-9, 1.586,1)
transfer_func_highpass = lambdify(s, variable_voltages_q3[3],
   'numpy')  # output at Vi(s) = 1
plt.loglog(ww, abs(transfer_func_highpass(ss)), lw=2)
plt.xlabel('frequencies')
plt.ylabel('|H(jw)|')
plt.title('Transfer Function Magnitude')
plt.grid()
plt.show()
```

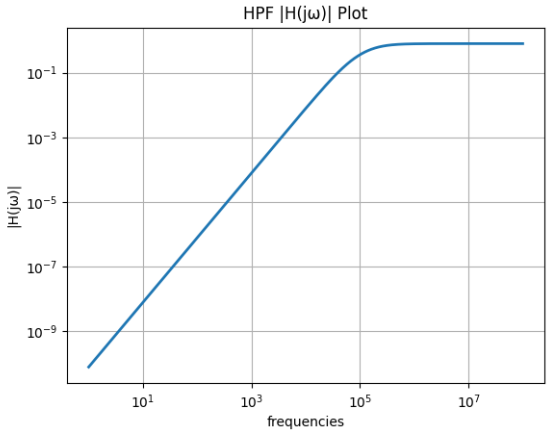

Figure 6: Transfer Function of High Pass Filter

3.4 Question 4: Damped Sinusoid

To analyse the response of the HPF to a damped sinusoid, I study its response to first, a low frequency input, then a high frequency.
Again, `sp.lsim` finds the time domain output. The two signals are

$$V1_i(t) = e^{-5t}\cos(20\pi t)$$

$$V2_i(t) = e^{-50000t}\cos(2 * 10^8 \pi t)$$

```
damping_fac = 5
t1 = np.linspace(0,1,1000)
V1_q4 = np.exp(-damping_fac*t1)*np.cos(20*np.pi*t1)
H = convert_sympy_to_lti(variable_voltages_q3[3])
t1,Vo1_q4,_ = sp.lsim(H, V1_q4,t1)

plt.plot(t1,V1_q4,label='input')
plt.plot(t1,Vo1_q4,label='output')
plt.xlabel('time')
plt.ylabel('voltage')
plt.title('Low Frequency Input')
plt.legend()
plt.grid()
plt.show()
```

One can expect this function to not pass through, the plot is seen in Figure 7.

Next, a high frequency damped input. Since I have used a frequency higher than the pole frequency, this should pass through the filter. See Figure 8.

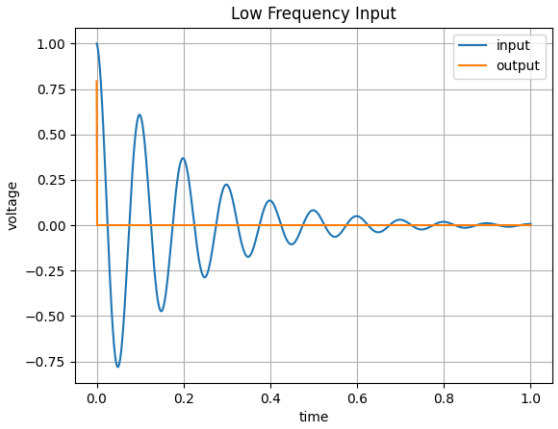

Figure 7: Low Frequency Damped Input

```
damping_fac = 50000
t2 = np.linspace(0,0.0001,1000)
V2_q4 = np.exp(-damping_fac*t2)*np.cos(2e8*np.pi*t2)
t2,Vo2_q4,_ = sp.lsim(H, V2_q4,t2)

plt.plot(t2,V2_q4,label='input')
plt.plot(t2,Vo2_q4,label='output')
plt.xlabel('time')
plt.ylabel('voltage')
plt.title('High Frequency Input')
plt.legend()
plt.grid()
plt.show()
```

3.5 Question 5: Unit Step Response

Unit step at the input is $V_i(s) = 1/s$ which is given to `highpass` to obtain the frequency domain sympy output. This is converted to time domain by `sp.impulse`. The plot is Figure 9.

```
conductance_matrix_q5, variable_voltages_q5, V_q5 =
highpass(10000, 10000, 1e-9, 1e-9, 1.586, 1/s)
print('Unit Response Vo(s) = {}'.format(V_q1[3]))
# Vo_func = lambdify(s,V_q1[3],'numpy')
t, unit_response_hpf = sp.impulse(convert_sympy_to_lti
(variable_voltages_q5[3]), None, np.linspace(0, 0.001, 10000))
plt.plot(t, unit_response_hpf, label='Output')
plt.plot(t,np.ones(len(t)),'red',label='Input')
```

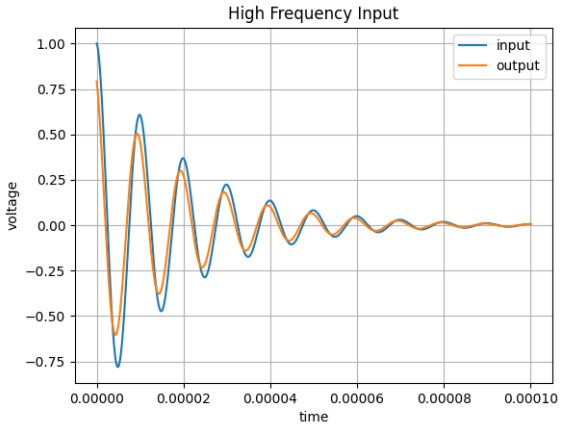

Figure 8: High Frequency Damped Input

```
plt.xlabel('Time')
plt.ylabel('voltages')
plt.title('Step Response of HPF')
plt.legend()
plt.grid()
plt.show()
```

The zero output is because the capacitors will quickly ($< 0.1ms$) block DC and no signal will reach the opamp.

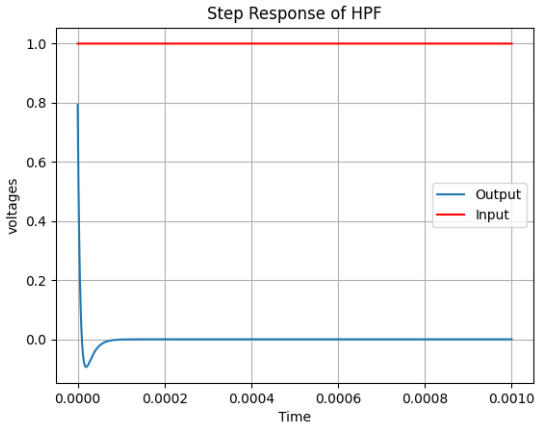

Figure 9: HPF Unit Step Response

4 Conclusion

The goal that I started the assignment with, I now seen to have been accomplished. Symbolic algebra is implemented in Python by the `sympy` library. Combining the use of symbols in functions to analyse filters and their transfer functions is what most of the assignment was about. I also observed the behavior of these filters for different frequencies of input.