EE2703: Applied Programming Lab
Assignment 7

Devaganthan S S
EE19B018

June 9, 2021

0.1 Abstract

This assignment aims to focus on two powerful capabilities of python:

- Symbolic Algebra
- Analysis of Circuits Using Laplace Transforms

0.2 Introduction

Python can be used to solve Circuits, get Transfer Functions, by formulating Matrices. In this assignment, we try to find Output to a High and Low pass Filters, with majorly using sympy functions. The idea is to form expressions of Transfer Functions or Output, with help of the **'symbol()'** function, and then convert it into an LTI system. With the help of **'lsim()'** or **'step()'** functions, we can find the output response.

0.3 Results and Implementation

0.3.1 Step Response of LPF:

We first find the expression for the transfer function of the LPF, assuming the input(Vi) as 1. Then with the help of the **'step()'** function, we find the step response of the LPF. The below code accomplishes the above tasks and plots the Step Response.

```
def lowpass(R1, R2, C1, C2, G, Vi):
    s = sy.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])  # Source Vetor
    V = A.inv()*B
    return A, B, V


def tranferfuncConverter(Vo):
    n, d = fraction(Vo)
    n = Poly(n, s)
    d = Poly(d, s)
    nCoeff = n.all_coeffs()
    dCoeff = d.all_coeffs()
    nCoeff = [float(i) for i in nCoeff]
    dCoeff = [float(i) for i in dCoeff]
    H = lti(nCoeff, dCoeff)
    return H


s = symbols('s')  # Intializing
# Vi is the matrix containg the expressiosn of the Node voltages
A, B, Vi = lowpass(10e3, 10e3, 1e-9, 1e-9, 1.586, 1)
t = np.arange(0, 0.1, 1e-7)
Vo = Vi[3]  # Transfer Function
# Q1
H = tranferfuncConverter(Vo)  # H is the Transfer Function
time, voStep = sp.step(H, None, linspace(0, 0.001, 10**4))
plot(time, voStep)
title('Step Response of the LPF')
xlabel('Time')
ylabel('Output')
show()
```

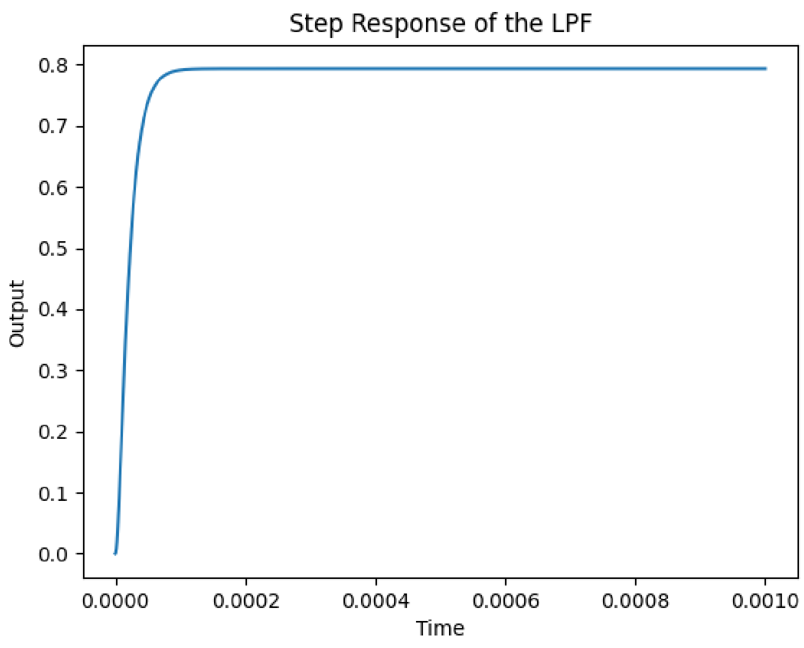

Figure 1

0.3.2 Output to the Given Signal:

For the given input, $v_i(t) = (sin(2000\pi t) + cos(2.10^6\pi t))u_o(t)$, the output can be found using the **'lsim()'** function. The transfer function H is obtained from the previous part. The below code finds the output and plots the same. From the plot, we can see that the output has only low frequency, $(2000\pi sine)$.

```
# Q2
vi = np.sin(2000*pi*t)+np.cos(2e6*pi*t)  # Given Input
time, Vo_HP, svec = lsim(H, vi, t)  # Vo_HP is the output (
    Filtered Output)
plot(t, vi, label='Input Signal')
plot(t, Vo_HP, label='Filtered Signal')
title('Output to the given Signal')
xlabel('Time')
ylabel('Output/Input')
legend()
xlim(0, 1e-3)
show()
```

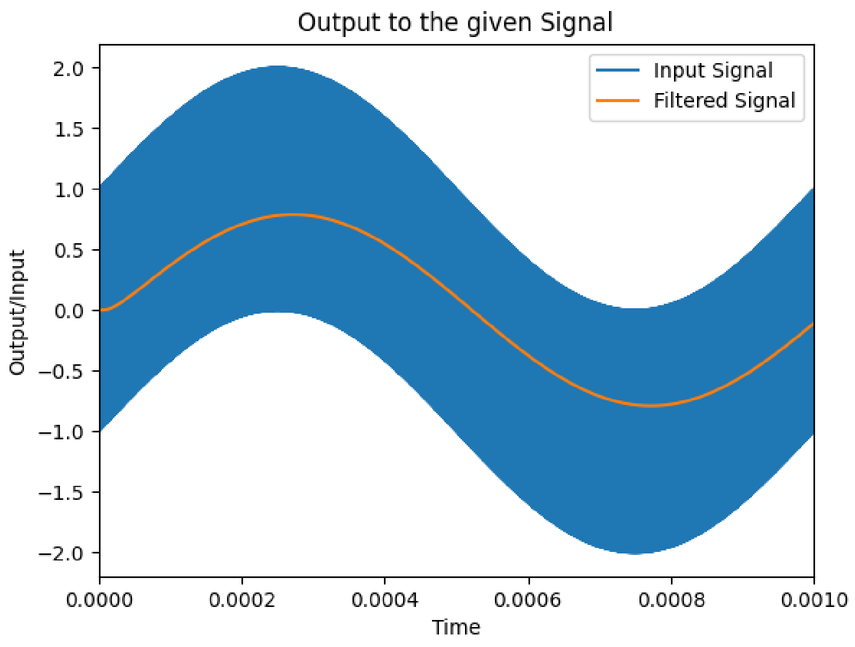

Figure 2

0.3.3 Magnitude Plot of HPF

Adopting the same procedure as before for the LPF, we find the Transfer function for the HPF. We plot the Magnitude vs Phase Plot. The below code accomplishes the above tasks.

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

    B = Matrix([0,
                0,
                0,
                -Vi*s*C1])
    V = A.inv()*B
    return A, B, V


# Q3
A, B, V = highpass(10e3, 10e3, 1e-9, 1e-9, 1.586, 1)  # V is the
    Unknown Vector
Vo = V[3]  # Transfer Function for the HPF
H = tranferfuncConverter(Vo)  # Converting the Expression
w = logspace(0, 8, 801)
ss = 1j*w
f = lambdify(s, Vo, 'numpy')
loglog(w, abs(f(ss)), lw=2)  # Plotting the Maginute vs Frequency
    plot
title('Bode Magnitude Plot for HPF(Logscale)')
xlabel('Frequency')
ylabel('Magnitude')
show()
```

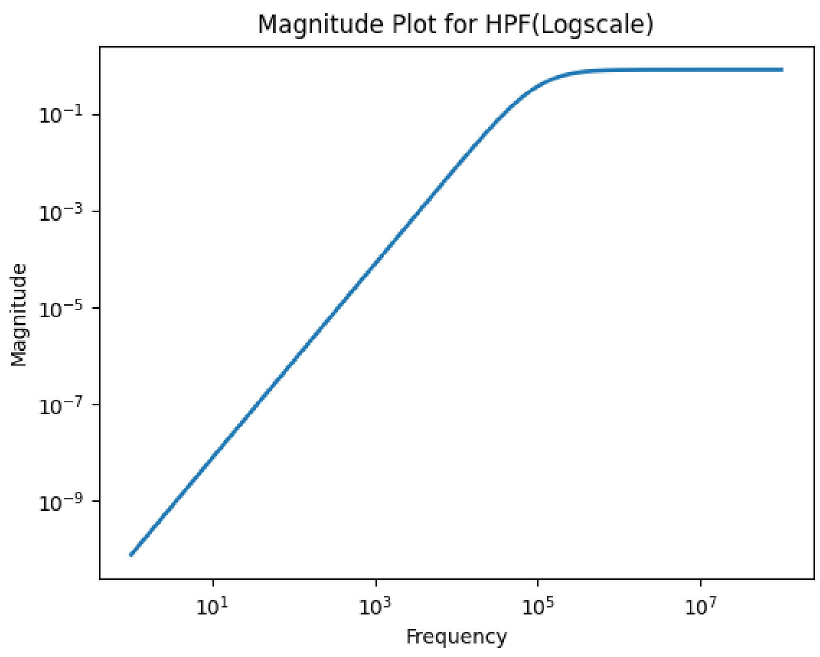

Figure 3

0.3.4 Output for a Damped Sinusoid for the HPF

Using the Transfer Function from the previous part and using the function **'lsim()'** we can find the response to the Damped Sinusoid for the HPF. The below code computes the output and plots the same. From the plots, we can see that the high frequencies amplitude have been reduced a bit, but whereas the low frequencies are completely suppressed.

```
# Q4
Vi1 = np.sin(1e3*pi*t)*np.exp(-1e2*t)  # Low Frequency Input
    signal
Vi2 = np.sin(1e6*pi*t)*np.exp(-1e4*t)  # High Frequency Input
    Signal
time, Vo1, svec = lsim(H, Vi1, t)  # Output to the Low Freq
time, Vo2, svec = lsim(H, Vi2, t)  # Output to the High Freq
plot(t, Vi1, label='Input Signal')
plot(time, Vo1, label='Filtered Signal')
xlabel('Time')
ylabel('Output/Input')
xlim(0, 1e-2)
legend()
show()
```

```
plot(t, Vi2, label='Input Signal')
plot(time, Vo2, label='Filtered Signal')
xlabel('Time')
ylabel('Output/Input')
legend()
xlim(0, 0.4*1e-4)
show()
```

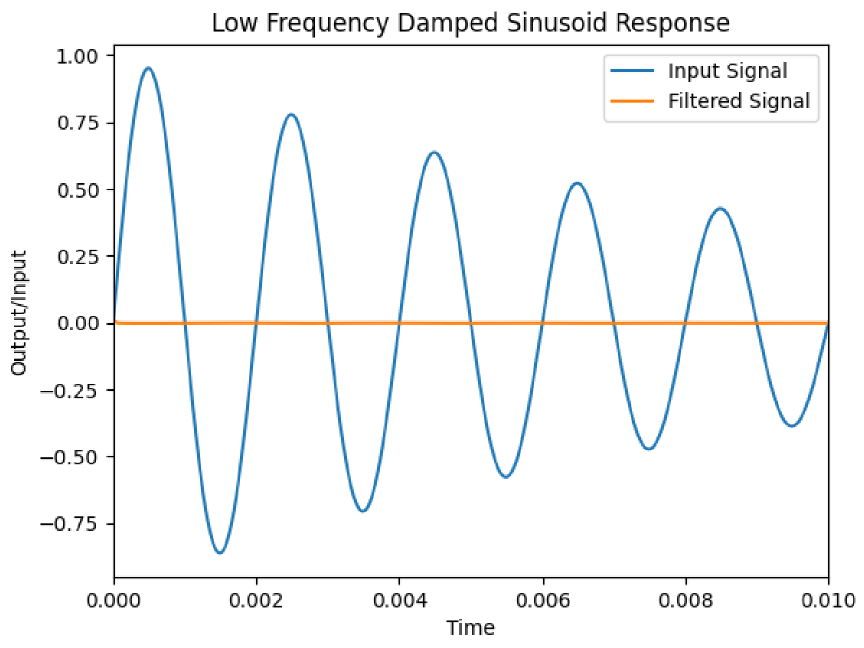

Figure 4

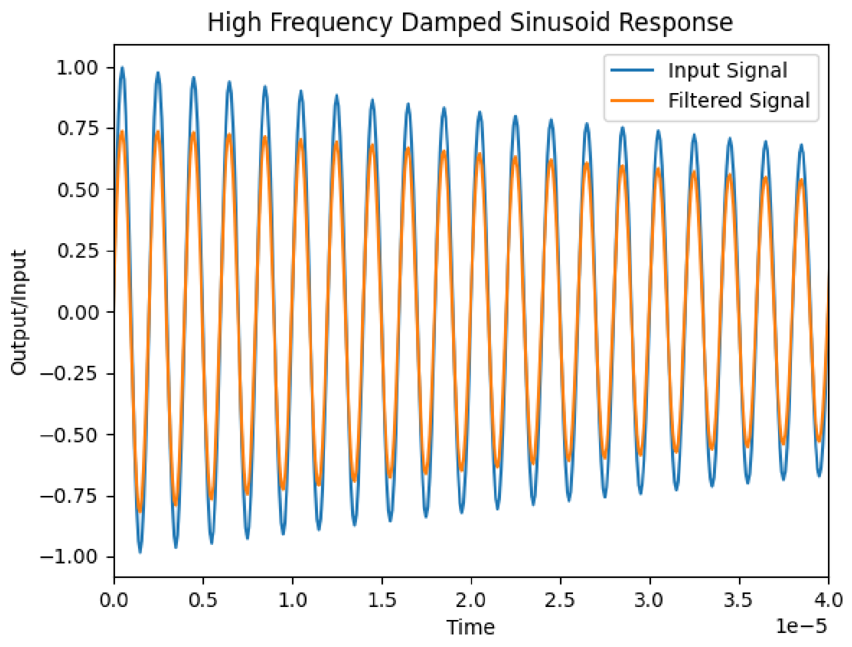

Figure 5

0.3.5 Step Response for the HPF:

The same procedure as the LPF case has been followed to find the step response. The below computes the step response and plots the same.

```
# Q5
time, voStep = sp.step(H, None, linspace(0, 0.001, 10**4))
plot(time, voStep)
show()
```

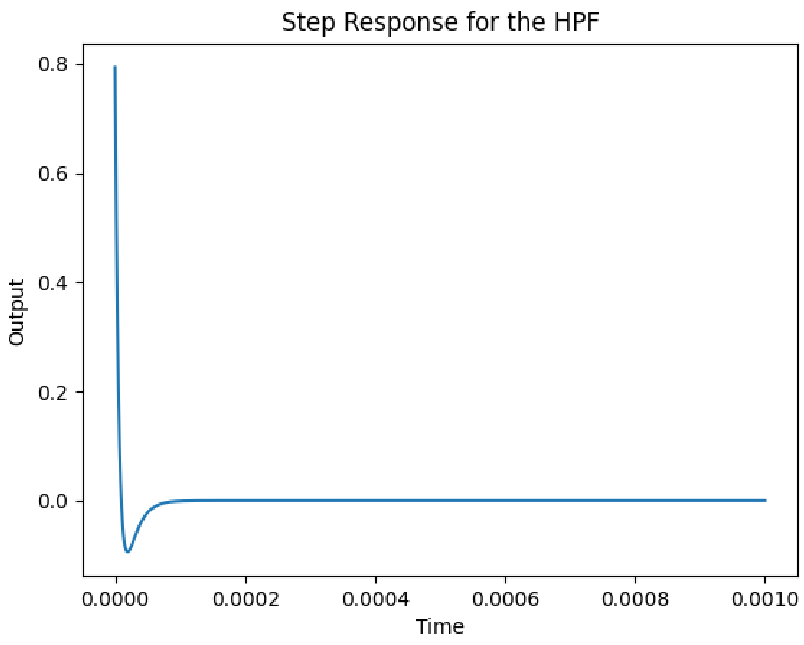

Figure 6

0.4 Conclusion

Symbolic Algebra and Signal toolbox are powerful tools that help to solve Complex Circuit Problems. Laplace Transform Analysis of circuits can be effectively done in python.