Project Report for ECE 351

Lab 04 - System Step Response Using Convolution

Skyler Corrigan

September 28, 2022

ECE351 Code Repository:
$https://github.com/ElfinPeach/ECE351_Code.git$

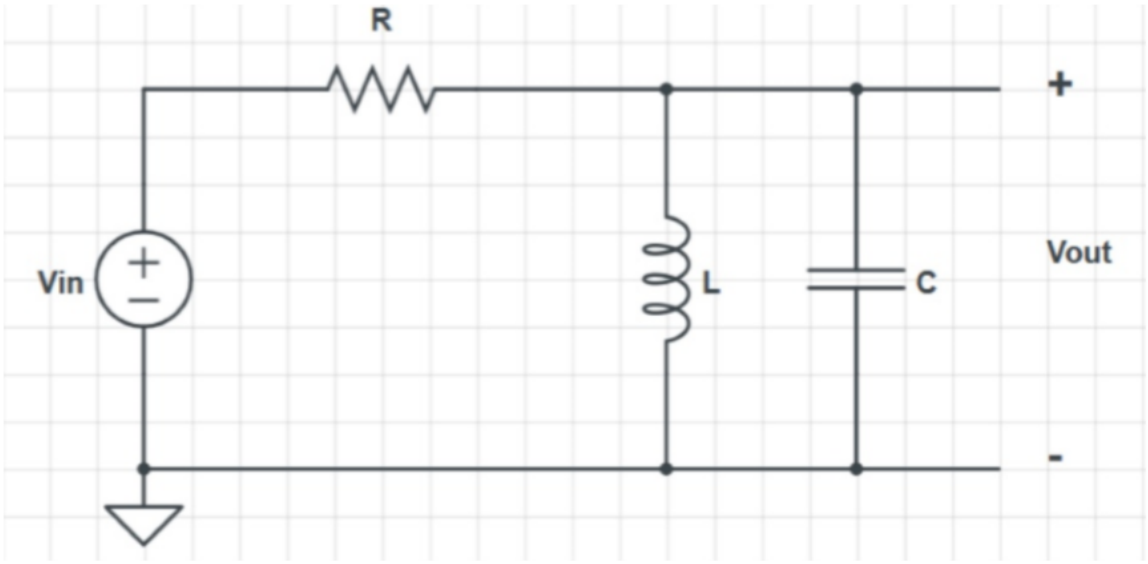
ECE351 Report Repository:
$https://github.com/ElfinPeach/ECE351_Report.git$

Contents

1 Objectives

This lab revolves around the following circuit:

Figure 1: Circuit



R = 1kΩ, L = 27mH, C = 100n

For the first part, we had to find the impulse response using Laplace Transformations and compare it with the inborn method in Python. The second part revolves around using the inborn step function and evaluating the equations with the Final Value Theorem in the Laplace domain.

2 Part 1

The Laplace transformed function found can be found in the following equation:

$$H(s) = \frac{10^4 * s}{s^2 + 10^4 * s + 3.7 * 10^8}$$

However, for the code everything was kept in symbolic notation and various parts were used for the final equation calculations. These equations can be seen below.

$$p = \frac{-1}{2RC} + 0.5 * \sqrt{(\frac{1}{RC})^2 - 4(\frac{1}{LC})}$$
$$p = \alpha + j\omega$$
$$|g| = \sqrt{(-.5\frac{1}{RC})^2 + (\frac{1}{2RC}\sqrt{(1/RC)^2 - 4(\frac{1}{LC})}}$$
$$\angle g = tan^{-1}(\frac{\frac{1}{2RC}\sqrt{(1/RC)^2 - 4\frac{1}{LC}}}{-.5(1/RC)^2}$$

The final equation can be found using everything in these equations, as shown below.

$$h(t) = \frac{|g|}{\omega} e^{\alpha t} \sin(\omega t + \angle g) u(t)$$

These equations were put into Python along with the built in impulse equation, resulting in the following code:

Impulse Code

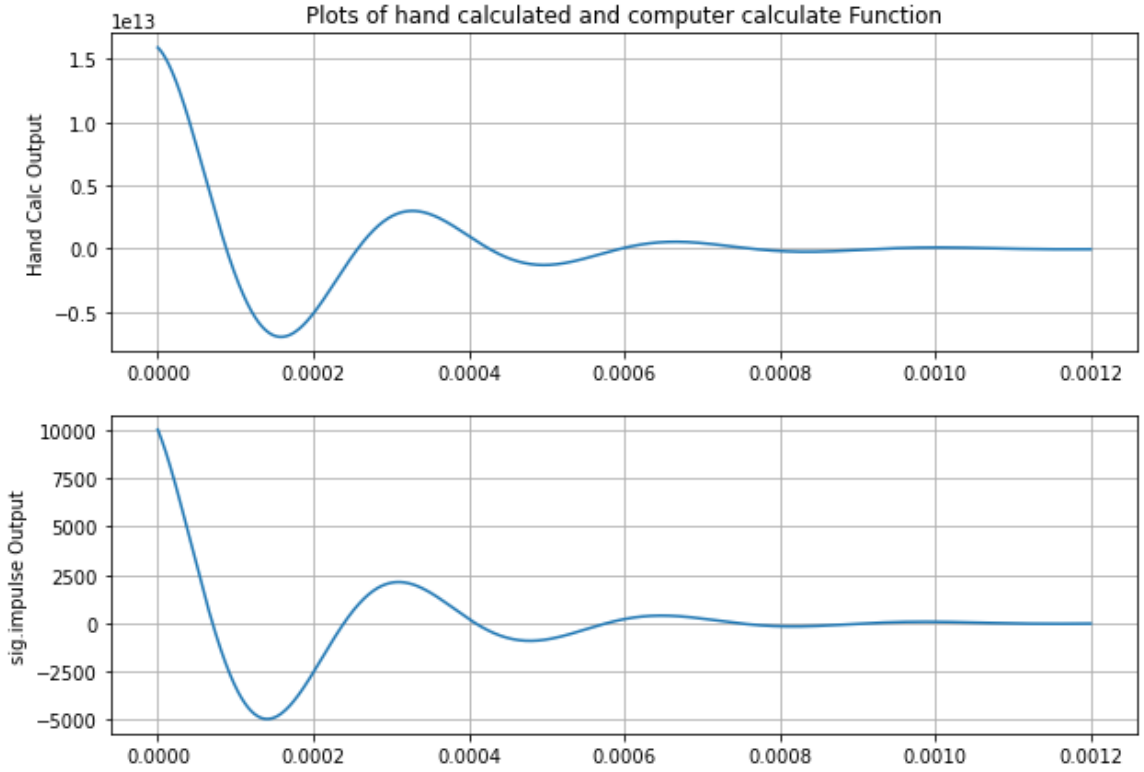
```
import numpy as np
import matplotlib.pyplot as plt
import math
import scipy.signal as sig

#-----------FUNCTIONS-----------------------------------------------------#

def cosine(t): # The only variable sent to the function is t
    y = np.zeros(t.shape) # initialze y(t) as an array of zeros
    for i in range(len(t)): # run the loop once for each index of t
        y[i] = np.cos(t[i])
    return y

#Make a step function using an array t, stepTime, and stepHeight
def stepFunc(t, startTime, stepHeight):
    y = np.zeros(t.shape)
    for i in range(len(t)):
        if(t[i] >= startTime):
            y[i] = stepHeight
    return y

#Make a ramp function using an array t, startTime, and slope
def rampFunc(t, startTime, slope):
    y = np.zeros(t.shape)
    for i in range(len(t)):
        if(t[i] >= startTime):
            y[i] = t[i]-startTime
    y = y * slope
    return y

#Make e^at function using array t, startTime, and a (alpha)
def eExpo(t,amplatude,alpha):
    y = np.zeros(t.shape)
    for i in range(len(t)):
        y[i]=amplatude * math.exp(alpha * (t[i]))

    return y
```

```
#Convolution function
def convolve(f1, f2):

    #Both functions need to be the same size or else the universe will exp
    Nf1 = len(f1)
    Nf2 = len(f2)

    f1Extend = np.append(f1,np.zeros((1,Nf2-1)))
    f2Extend = np.append(f2,np.zeros((1,Nf1-1)))

    y = np.zeros(f1Extend.shape)

    for i in range(Nf1 + Nf2 - 2):
        y[i] = 0

        for j in range(Nf1):
            if (i-j+1 >0):
                try:
                    y[i] += f1Extend[j] * f2Extend[i-j+1]

                except:
                    print(i,j)
    return y

def funcPlot(t, R, L, C):

    X = (1/(R*C))
    tempR = -0.5 * ((1/(R*C))**2) #real part of g
    tempI = 0.5 * X * (math.sqrt((X**2)) - (4/(L*C))) #imaginary part of g

    gMag = math.sqrt( (tempR**2) + (tempI**2))

    tempTanX = 0.5 * (math.sqrt((X**2)) - (4/(L*C))) #numerator of angle g
    tempTanY = -0.5 * ((1/(R*C))**2) #denominator for angle g

    gDeg = math.atan(tempTanX / tempTanY) #g angle

    w = 0.5 * np.sqrt(X**2 - 4*(1/np.sqrt(L*C))**2 + 0*1j)
    a= (-0.5) * X

    y = (4e4)*(gMag/np.abs(w)) * np.exp(a * t) * np.sin(np.abs(w) * t + gD

    return y
```

```
#----------------END FUNCTIONS----------------------------------------#

#----------------DEFINITIONS------------------------------------------#

steps = 1e-6

#t for part 1
start = 0
stop = 1.2e-3
#Define a range of t. Start at 0 and go to 20 (+a step)
t = np.arange(start, stop + steps, steps)


#For circuit----------------RLC DEF-----------
R = 1e3 #Ohms
L = 27e-3 #Henries
C = 100e-9 #Ferrets :)

y = funcPlot(t, R, L, C)

#----------------END DEFINITIONS--------------------------------------#

#----------------Part One---------------------------------------------#
num = [0, L, 0] #Creates a matrix for the numerator
den = [C*R*L, L, R] #Creates a matrix for the denominator

tout1 , ySig = sig.impulse ((num , den), T = t)


#Make plots
plt.figure(figsize=(10,7))
plt.subplot(2,1,1)
plt.plot(t,y)
plt.grid()
plt.ylabel('Hand Calc Output')
plt.title('Plots of hand calculated and computer calculate Function')

plt.subplot(2,1,2)
plt.plot(t,ySig)
plt.grid()
plt.ylabel('sig.impulse Output')

plt.show()
```

This resulted in the following image:

Figure 2: Impulse Graphs



As is plainly seen, these graphs are identical.

3 Part 2

This section shows the built in step function in Python for this circuit and compares it to the built in impulse function. The code for this can be found below.

Impulse Code

```
import numpy as np
import matplotlib.pyplot as plt
import math
import scipy.signal as sig

#-----------------FUNCTIONS-----------------------------------------------------#

def cosine(t): # The only variable sent to the function is t
    y = np.zeros(t.shape) # initialze y(t) as an array of zeros
    for i in range(len(t)): # run the loop once for each index of t
        y[i] = np.cos(t[i])
    return y
```

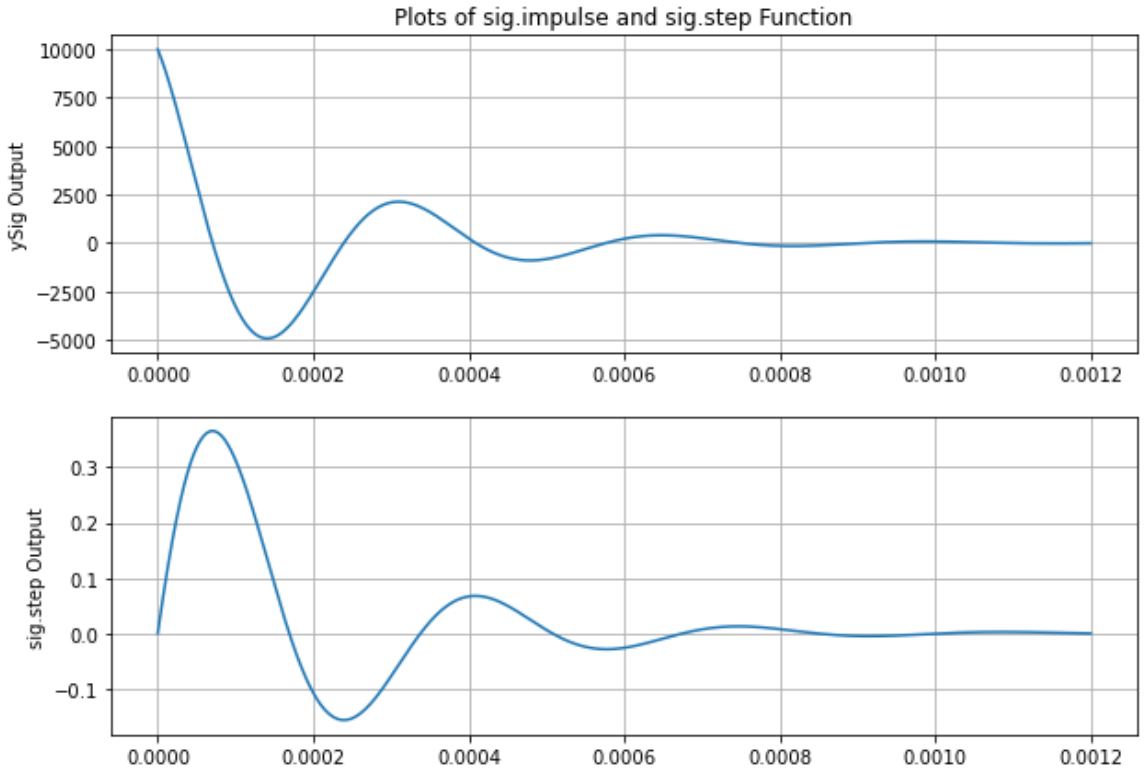
```
#Make a step function using an array t, stepTime, and stepHeight
def stepFunc(t, startTime, stepHeight):
    y = np.zeros(t.shape)
    for i in range(len(t)):
        if(t[i] >= startTime):
            y[i] = stepHeight
    return y

#Make a ramp function using an array t, startTime, and slope
def rampFunc(t, startTime, slope):
    y = np.zeros(t.shape)
    for i in range(len(t)):
        if(t[i] >= startTime):
            y[i] = t[i]-startTime
    y = y * slope
    return y

#Make e^at function using array t, startTime, and a (alpha)
def eExpo(t,amplatude,alpha):
    y = np.zeros(t.shape)
    for i in range(len(t)):
        y[i]=amplatude * math.exp(alpha * (t[i]))

    return y

#Convolution function
def convolve(f1, f2):

    #Both functions need to be the same size or else the universe will exp
    Nf1 = len(f1)
    Nf2 = len(f2)

    f1Extend = np.append(f1,np.zeros((1,Nf2-1)))
    f2Extend = np.append(f2,np.zeros((1,Nf1-1)))

    y = np.zeros(f1Extend.shape)

    for i in range(Nf1 + Nf2 - 2):
        y[i] = 0

        for j in range(Nf1):
            if (i-j+1 >0):
                try:
                    y[i] += f1Extend[j] * f2Extend[i-j+1]
```

```
                except:
                    print(i,j)
    return y

def funcPlot(t, R, L, C):

    X = (1/(R*C))
    tempR = -0.5 * ((1/(R*C))**2) #real part of g
    tempI = 0.5 * X * (math.sqrt((X**2)) - (4/(L*C))) #imaginary part of g

    gMag = math.sqrt( (tempR**2) + (tempI**2))

    tempTanX = 0.5 * (math.sqrt((X**2)) - (4/(L*C))) #numerator of angle g
    tempTanY = -0.5 * ((1/(R*C))**2) #denominator for angle g

    gDeg = math.atan(tempTanX / tempTanY) #g angle

    w = 0.5 * np.sqrt(X**2 - 4*(1/np.sqrt(L*C))**2 + 0*1j)
    a= (-0.5) * X

    y = (4e4)*(gMag/np.abs(w)) * np.exp(a * t) * np.sin(np.abs(w) * t + gD

    return y


#-----------------END FUNCTIONS-----------------------------------------#

#-----------------DEFINITIONS-----------------------------------------#

steps = 1e-6

#t for part 1
start = 0
stop = 1.2e-3
#Define a range of t. Start at 0 and go to 20 (+a step)
t = np.arange(start, stop + steps, steps)


#For circuit-----------------RLC DEF-----------
R = 1e3 #Ohms
L = 27e-3 #Henries
C = 100e-9 #Ferrets :)

y = funcPlot(t, R, L, C)
```

```
#-------------------END DEFINITIONS-------------------------------------#

#-------------------------------Part Two--------------------------------#
tout2 , yStep = sig.step ((num , den), T = t)

plt.figure(figsize=(10,7))
plt.subplot(2,1,1)
plt.plot(t,ySig)
plt.grid()
plt.ylabel('ySig Output')
plt.title('Plots of sig.impulse and sig.step Function')

plt.subplot(2,1,2)
plt.plot(t,yStep)
plt.grid()
plt.ylabel('sig.step Output')

plt.show()
```

This resulted in the following graph:

Figure 3: Other Graphs



These graphs are similar, but it seems that the step function is phase shifted.

3.1 Final Value Theorem

To evaluate the theoretical Final Value using Laplace equivalent function, it looks like this:

$$\lim_{t\to\infty} h(t) = \lim_{s\to 0} sH(s)$$

Looking back at the previously found Laplace equation:

$$H(s) = \frac{10^4 * s}{s^2 + 10^4 * s + 3.7 * 10^8}$$
$$\lim_{s\to 0} sH(s) = \lim_{s\to 0} \frac{10^4 * s^2}{s^2 + 10^4 * s + 3.7 * 10^8}$$

Looking at this equation, when the limit of s approaches 0, the numerator goes to 0, whereas the denominator goes to $3.7 * 10^8$, thus the $\lim_{s\to 0} sH(s) = 0$. This means that the function h(t) will reach 0, which is shown in the graphs in Figures 2 and 3.

4 Questions

Explain the result of the Final Value Theorem from Part 2 Task 2 in terms of the physical circuit components.

The inductor and capacitor has energy that stored/released as time progresses, eventually settling out. Thus the gain eventually goes to 0 as time goes to infinity.