# ECE351-51: LAB 4

OCTOBER 1, 2019

Submitted By:

Adriana Oliveira

## Introduction

This lab expanded on our understanding of convolutions to include solving a system's step response. This was achieved by using a previously created step functions to define three transfer functions.

## Methodology

### Part 1: Create three transfer functions

We started by adding our peripheral libraries and pasting our previously defined step function from Lab 2. We then created user defined function to express the following transfer functions as seen in task 1 of appendix.

**Created transfer equations:**

$$h_1(t) = e^{2t} * u * (1-t),$$

$$h_2(t) = u * (t-2) - u * (t-6),$$

and

$$h_3(t) = cos(_o t) * u(t), for f_o = 25Hz$$

These equations were plotted in subplots from t = -10s to t = 10s with steps of .1.

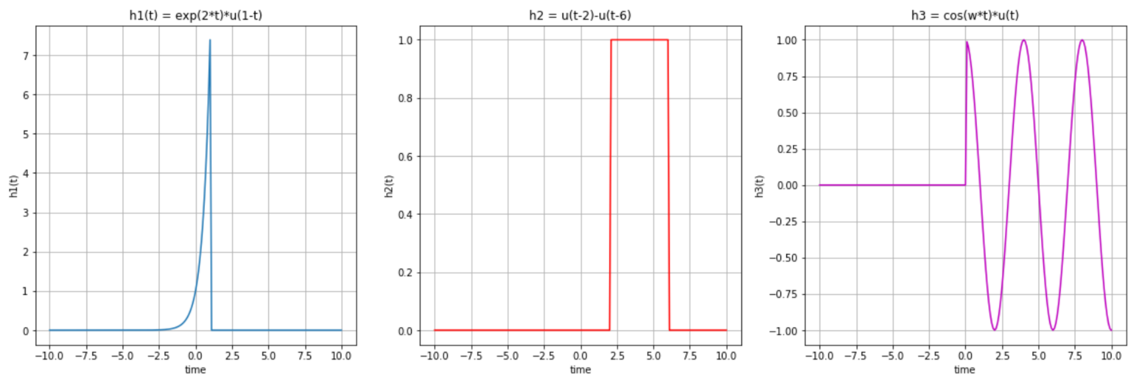

**Figure 1:** Plots for $h_1(t), h_2(t)$, and $h_3(t)$

## Part 2: Plot step responses

We first plotted the step response using the convolution function created in lab 3. To do so, the step function had to be rewritten as an array in order to find its convolution with the transfer function arrays. This process is seen in the code below.Please see appendix task 2 for code related to the following plots.

**Listing 1:** Redefined step function

```
#Function to make unit step an array in order to convolve.
def vect_u(t):
    y = np.zeros((len(t), 1))
    for i in range(len(t)):
        y[i] = u(t[i])
    return y
```

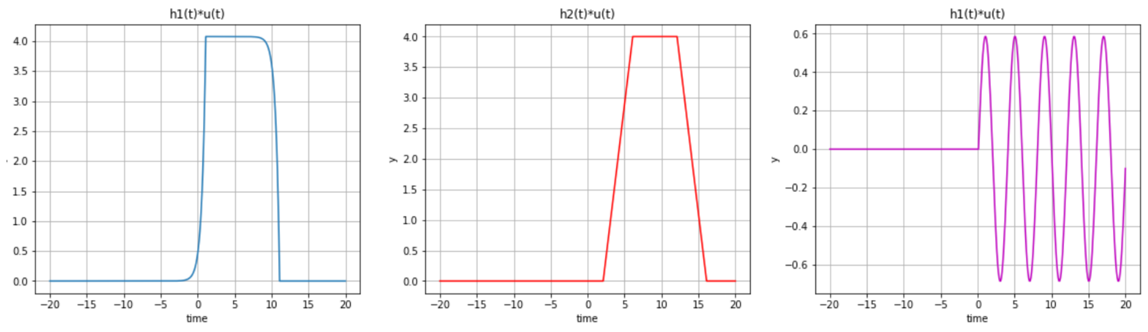

**Figure 2:** Python calculated step response plots

The next step was to verify with hand calculations. Please see the equations section for full details. The results from all three convolutions are seen below.

**Hand calculated convolutions**

$$y_1(t) = .5 * (e^{2*t} * u(1-t) + e^2 * u(t-1))$$

$$y_2(t) = (t-2) * u(t-2) - (t-6) * u(t-6))$$

$$y_3(t) = (1/(_o) * sin(_o * t) * u(t)$$

Functions for each of these equations was made in order to plot them. Here we had to take care in knowing that since the answer in a convolution, it needed a larger array then the transfer functions and unit step function needed.

**Listing 2:** User defined functions for hand calculated convolutions

```
#Function to make unit step an array in order to convolve.
#Functions for hand calculated convolution
def y1(tExtended):
    a = np.zeros((len(tExtended), 1))

    for i in range(len(tExtended)):
        a[i] = .5*((np.exp(2*tExtended[i])*u(1-tExtended[i]))+
        (np.exp(2)*u(tExtended[i]-1)))
    return a

def y2(tExtended):
    b = np.zeros((len(tExtended), 1))

    for i in range(len(tExtended)):
        b[i] = ((tExtended[i]-2)*u(tExtended[i]-2))-((tExtended[i]-
        6)*u(tExtended[i]-6))
    return b

def y3(tExtended):
    c = np.zeros((len(tExtended), 1))

    f = 0.25
    w = f*2*np.pi
    c = np.zeros((len(tExtended), 1))
    for i in range(len(tExtended)):
        c[i] = (math.sin(w*tExtended[i])*u(tExtended[i]))/w
    return c
```

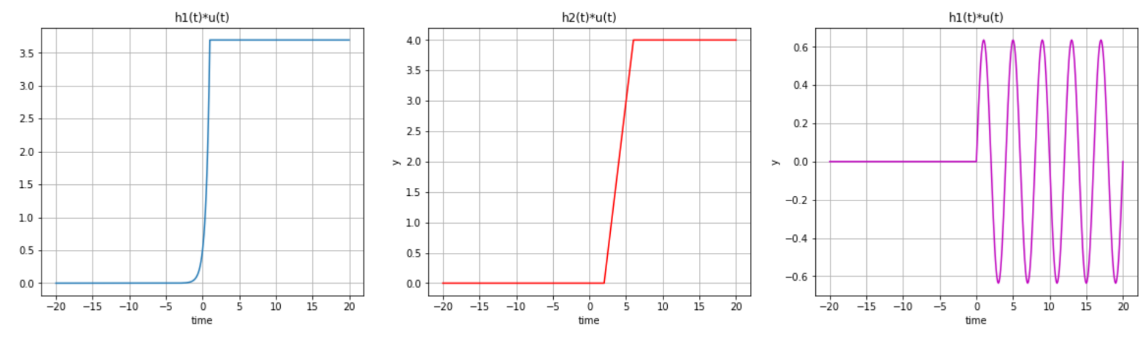

**Figure 3:** Hand calculated step response plots

## EQUATIONS

**Hand calculated convolutions**

$$\int h_1(t) * u(t)$$

$$= \int e^{2T} * u * (1 - T) * u(t - T) dT$$

$$= y_1(t) = .5 * (e^{2*t} * u(1 - t) + e^2 * u(t - 1)),$$

$$\int h_2(t) u(t)$$

$$= \int (u * (T - 2) - u * (T - 6)) * u(t - T) dT$$

$$= y_2(t) = (t - 2) * u(t - 2) - (t - 6) * u(t - 6)),$$

and

$$\int h_3(t) * u(t)$$

$$= \int cos(w_o T) * u(T) * u(t - T) dT$$

$$y_3(t) = (1/(w_o) * sin(w_o * t) * u(t)$$

## RESULTS

From the plots we can see how the original transfer functions change predictably after their convolution with the step function. Notice that the hand calculated convolution plots are overtaken by the unit step, whereas the python calculated convolution plots are more exact and return .

## CONCLUSION

Using the skills built from previous labs, we were able to compare hand calculated and python derived convolutions of transfer function.

## APPENDIX

**Listing 3:** Part 1: Task 1

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

#Defined unit function
def u(t):
    if t < 0:
        return 0
    if t >= 0:
        return 1

#Defined h(t) functions
def h1(t):
    a = np.zeros((len(t), 1))
    for i in range(len(t)):
        a[i] = np.exp(2*t[i])*u(1-t[i])
    return a

def h2(t):
    b = np.zeros((len(t), 1))
    for i in range(len(t)):
        b[i] = u(t[i]-2)-u(t[i]-6)
    return b

def h3(t):
    f = 0.25
    w = f*2*np.pi
    c = np.zeros((len(t), 1))
    for i in range(len(t)):
        c[i] = math.cos(w*t[i])*u(t[i])
    return c
```

**Listing 4:** Part 1: Task 2

```
steps = .1
t = np.arange(-10,10+steps,steps)

a = h1(t)
b = h2(t)
c = h3(t)

#Graphs for each h(t) function
myFigSize = (20,6)
plt.figure(figsize=myFigSize)

plt.subplot(1,3,1)
plt.plot(t,a)
plt.grid(True)
plt.xlabel('time')
plt.ylabel('h1(t)')
plt.title('h1(t)␣=␣exp(2*t)*u(1-t)')

plt.subplot(1,3,2)
plt.plot(t,b,'r')
plt.grid(True)
plt.xlabel('time')
plt.ylabel('h2(t)')
plt.title('h2␣=␣u(t-2)-u(t-6)')

plt.subplot(1,3,3)
plt.plot(t,c, 'm')
plt.grid(True)
plt.xlabel('time')
plt.ylabel('h3(t)')
plt.title('h3 = cos(w*t)*u(t)')
plt.show()
```

**Listing 5:** Part 2: Task 1

```
#Convolution Function (from lab 3)
def my_convo(h1,h2):
    length_h1 = np.size(h1)  #size of first function
    length_h2 = np.size(h2)  #size of second function

    #Result with be as large as the added lengths of the functions
    #-1 because they share t = 0 on the timeline.

    result = np.zeros(length_h1 + length_h2 -1)

    for m in np.arange(length_h1):          #for any time in the f1,
        for n in np.arange(length_h2):      #take the same time in f2.
            result[m+n] = result[m+n] + h1[m]*h2[n]
            #multiply these points and add with previous results.
            #This produces an integral

    return result
#Function to make unit step an array in order to convolve.
def vect_u(t):
    y = np.zeros((len(t), 1))
    for i in range(len(t)):
        y[i] = u(t[i])
    return y

    #Graphs for user defined convolution
steps = .1
t = np.arange(-10,10+steps,steps)
NN = len(t)
tExtended = np.arange(-20, 2*t[NN-1]+steps, steps)
```

```
a = h1(t)
b = h2(t)
c = h3(t)
y = vect_u(t)

my_conv_1 = my_convo(a,y)
my_conv_2= my_convo(b,y)
my_conv_3= my_convo(c,y)

myFigSize = (20,5)
plt.figure(figsize=myFigSize)

plt.subplot(1,3,1)
plt.plot(tExtended,my_conv_1*steps)
plt.grid(True)
plt.xlabel('time')
plt.ylabel('y')
plt.title('h1(t)*u(t)')

plt.subplot(1,3,2)
plt.plot(tExtended,my_conv_2*steps, 'r')
plt.grid(True)
plt.xlabel('time')
plt.ylabel('y')
plt.title('h2(t)*u(t)')

plt.subplot(1,3,3)
plt.plot(tExtended,my_conv_3*steps,  'm')
plt.grid(True)
plt.xlabel('time')
plt.ylabel('y')
plt.title('h1(t)*u(t)')
plt.show()
```

**Listing 6:** Part 2: Task 1

```
#Functions for hand calculated convolution
def y1(tExtended):
    a = np.zeros((len(tExtended), 1))

    for i in range(len(tExtended)):
        a[i] = .5*((np.exp(2*tExtended[i])*u(1-tExtended[i]))+(np.exp(2)
        *u(tExtended[i]-1)))
    return a

def y2(tExtended):
    b = np.zeros((len(tExtended), 1))

    for i in range(len(tExtended)):
        b[i] = ((tExtended[i]-2)*u(tExtended[i]-2))-((tExtended[i]-6)
        *u(tExtended[i]-6))
    return b

def y3(tExtended):
    c = np.zeros((len(tExtended), 1))

    f = 0.25
    w = f*2*np.pi
    c = np.zeros((len(tExtended), 1))
    for i in range(len(tExtended)):
        c[i] = (math.sin(w*tExtended[i])*u(tExtended[i]))/w
    return c

    #Graphs for hand calculated convolutions
steps = .1
t = np.arange(-10,10+steps,steps)
NN = len(t)
tExtended = np.arange(-20, 2*t[NN-1]+steps, steps)

a = y1(tExtended)
```

```
b = y2(tExtended)
c = y3(tExtended)

myFigSize = (20,5)
plt.figure(figsize=myFigSize)

plt.subplot(1,3,1)
plt.plot(tExtended,a)
plt.grid(True)
plt.xlabel('time')
plt.ylabel('y')
plt.title('h1(t)*u(t)')

plt.subplot(1,3,2)
plt.plot(tExtended,b,'r')
plt.grid(True)
plt.xlabel('time')
plt.ylabel('y')
plt.title('h2(t)*u(t)')

plt.subplot(1,3,3)
plt.plot(tExtended,c,'m')
plt.grid(True)
plt.xlabel('time')
plt.ylabel('y')
plt.title('h1(t)*u(t)')
plt.show()
```