Project Report for ECE 351

Lab 04 - System Step Response Using Convolution

Skyler Corrigan

September 20, 2022

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

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

Contents

1 Part 1

This section required me to create three plots for the following step functions:

$$h_1(t) = e^-2t[u(t) - u(t-3)]$$
$$h_2(t) = u(t-2) - u(t-6)$$
$$h_3(t) = \cos\omega_0 t u(t)$$

This was done with the following code:

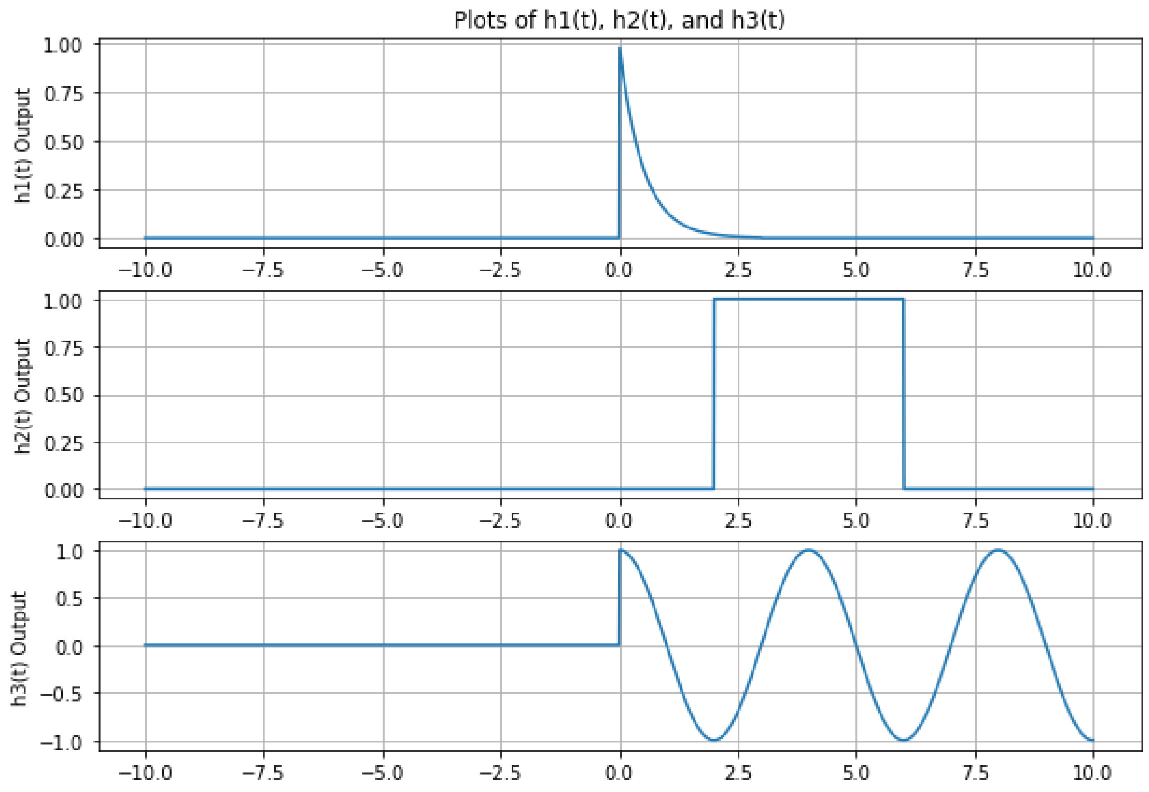
Impulse Code

```
import numpy as np
import matplotlib.pyplot as plt
import math

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



#----------------------------PART 1------------------------------------#

    #Define step size
steps = 1e-2

    #t for part 1
start = -10
stop = 10

    #convert frequency to rad/s
w=2*np.pi*0.25 #w roughlty equals 1.57

    #Define a range of t_pt1. Start at 0 and go to 20 (+a step)
```

```
t_pt1 = np.arange(start, stop + steps, steps)

    #Make h1(t) = e^(-2t)[u(t)-u(t-3)]
h1 = eExpo(t_pt1, 1, -2) * (stepFunc(t_pt1, 0, 1) - stepFunc(t_pt1, 3, 1))

    #Make h2(t) = u(t-2) - u(t-6)
h2 = stepFunc(t_pt1, 2, 1) - stepFunc(t_pt1, 6, 1)

    #Make h3(t) = cos(wt)*u(t) --> frequency = 0.25
h3 = cosine(w * t_pt1) * stepFunc(t_pt1, 0, 1)

#-----MAKE PLOTS--------#
    #Make plots for Part 1----------#
plt.figure(figsize=(10,7))
plt.subplot(3,1,1)
plt.plot(t_pt1,h1)
plt.grid()
plt.ylabel('h1(t) Output')
plt.title('Plots of h1(t), h2(t), and h3(t)')

plt.subplot(3,1,2)
plt.plot(t_pt1,h2)
plt.grid()
plt.ylabel('h2(t) Output')

plt.subplot(3,1,3)
plt.plot(t_pt1,h3)
plt.grid()
plt.ylabel('h3(t) Output')

plt.show()
```

This created the following figure:

Figure 1



2 Part 2

For this section, the previous impulse response functions needed to be convolved with a standard step function, using the user created convolution function from last lab. Then, the same convolutions were hand calculated and both the user defined convolution and hand calculated convolution were plotted.

The code for the entirety of this Task can be found below.
Impulse Code

```
import numpy as np
import matplotlib.pyplot as plt
import math

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

    #Define step size
steps = 1e-2

#-----------------------------PART 2-------------------------------------#
    #t for part 2
start = -10
stop = 10
    #Define a range of t_pt1. Start at -10 and go to 10
t_pt2 = np.arange(start, stop + steps, steps)


    #Make h1(t) = e^(-2t)[u(t)-u(t-3)]
h1 = eExpo(t_pt2, 1, -2) * (stepFunc(t_pt2, 0, 1) - stepFunc(t_pt2, 3, 1))

    #Make h2(t) = u(t-2) - u(t-6)
h2 = stepFunc(t_pt2, 2, 1) - stepFunc(t_pt2, 6, 1)

    #Make h3(t) = cos(wt)*u(t)
h3 = cosine(w * t_pt2) * stepFunc(t_pt2, 0, 1)
```

```
    #Make forcing function: f(t) = u(t)
forceFunc = stepFunc(t_pt1, 0, 1)

    #Make convolutions for step response
h1StepResponse = convolve(h1, forceFunc) * steps

h2StepResponse = convolve(h2, forceFunc) * steps

h3StepResponse = convolve(h3, forceFunc) * steps

#Make a tConv range to plot the convolve functions
#This should be the same as the size for all convolutions for this lab
tConv = np.arange(0, ((len(h3StepResponse) -1) * steps) + steps, steps) -2


#-----MAKE PLOTS----------#
    #Make plots for Part 2----------#
plt.figure(figsize=(10,7))
plt.subplot(3,1,1)
plt.plot(tConv,h1StepResponse)
plt.grid()
plt.ylabel('h1(t)␣conv.␣Output')
plt.title('h1(t),␣h2(t),␣and␣h3(t)␣User␣Conv')

plt.subplot(3,1,2)
plt.plot(tConv,h2StepResponse)
plt.grid()
plt.ylabel('h2(t)␣conv.␣Output')


plt.subplot(3,1,3)
plt.plot(tConv,h3StepResponse)
plt.grid()
plt.ylabel('h3(t)␣conv.␣Output')


#hand colvolutions:
h1hand = 0.5 * ((1 - eExpo(tConv, 1, -2)) * stepFunc(tConv, 0, 1)) - 0.5 *
h2hand = (tConv - 2) * stepFunc(tConv, 2, 1) - (tConv - 6) * stepFunc(tCon
h3hand = (1/w) * cosine(w * tConv - np.pi/2) * stepFunc(tConv, 0, 1)

plt.figure(figsize=(10,7))
plt.subplot(3,1,1)
plt.plot(tConv,h1hand)
plt.grid()
plt.ylabel('h1(t)␣conv.␣Output')
plt.title('h1(t),␣h2(t),␣and␣h3(t)␣Hand␣Convolutions')

plt.subplot(3,1,2)
plt.plot(tConv,h2hand)
plt.grid()
plt.ylabel('h2(t)␣conv.␣Output')

plt.subplot(3,1,3)
plt.plot(tConv,h3hand)
plt.grid()
plt.ylabel('h2(t)␣conv.␣Output')

plt.show()
```

2.1 Equations

These were the steps I used to calculate the first convolution.

$$\begin{aligned} y_x(t) &= \textstyle\int_{-\infty}^{\infty} h_1(t-\tau)u(t)\,d\tau \\ &= \textstyle\int_0^t h_1(\tau)\,d\tau \\ &= \textstyle\int_0^t e^-2\tau\,d\tau \\ &= (1/2)(1-e^{-2t}) \end{aligned}$$

This convolution only covers the $u(t)$ part of this impulse response. However, since the $u(t-3)$ part is the same, only with a time shift, the entire equation can be represented by the function:

$$y_1(t) = .5[(1-e^{-2t})u(t) - (1-e^{-2(t-3)})u(t-3)]$$

These were the steps I used to calculate the second convolution.
Since both parts of it are step responses with time shift, I just used the unshifted step response, then added the time shift back in at the end.

$$\begin{aligned} y_y(t) &= \textstyle\int_{-\infty}^{\infty} h_2(t-\tau)u(t)\,d\tau \\ &= \textstyle\int_0^t h_2(\tau)\,d\tau \\ &= \textstyle\int_0^t \,d\tau \\ &= tu(t) \end{aligned}$$

Thus, the entire equation (after adding the time shifts back in) can be represented by:

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

These were the steps I used for the final convolution.

$$\begin{aligned} y_y(t) &= \textstyle\int_{-\infty}^{\infty} h_3(t-\tau)u(t)\,d\tau \\ &= \textstyle\int_0^t h_3(\tau)\,d\tau \\ &= \textstyle\int_0^t \cos\omega_0\tau\,d\tau \\ &= (1/\omega_0)\sin(\omega_0 t)u(t) \end{aligned}$$

Note: I only had a cosine function ready in Python and didn't feel like making a sine function, so I used a $-90°$ (or $-\pi/2$) phase shift to change the sine into a cosine function. Thus, the final equation looks like this:

$$y_3(t) = (1/\omega_0)\cos(\omega_0 t - \pi/2)u(t)$$

2.2 Graphs

The convolution using the user-defined function and the hand-convolution can be found below.

Figure 2: User-Defined Convolution

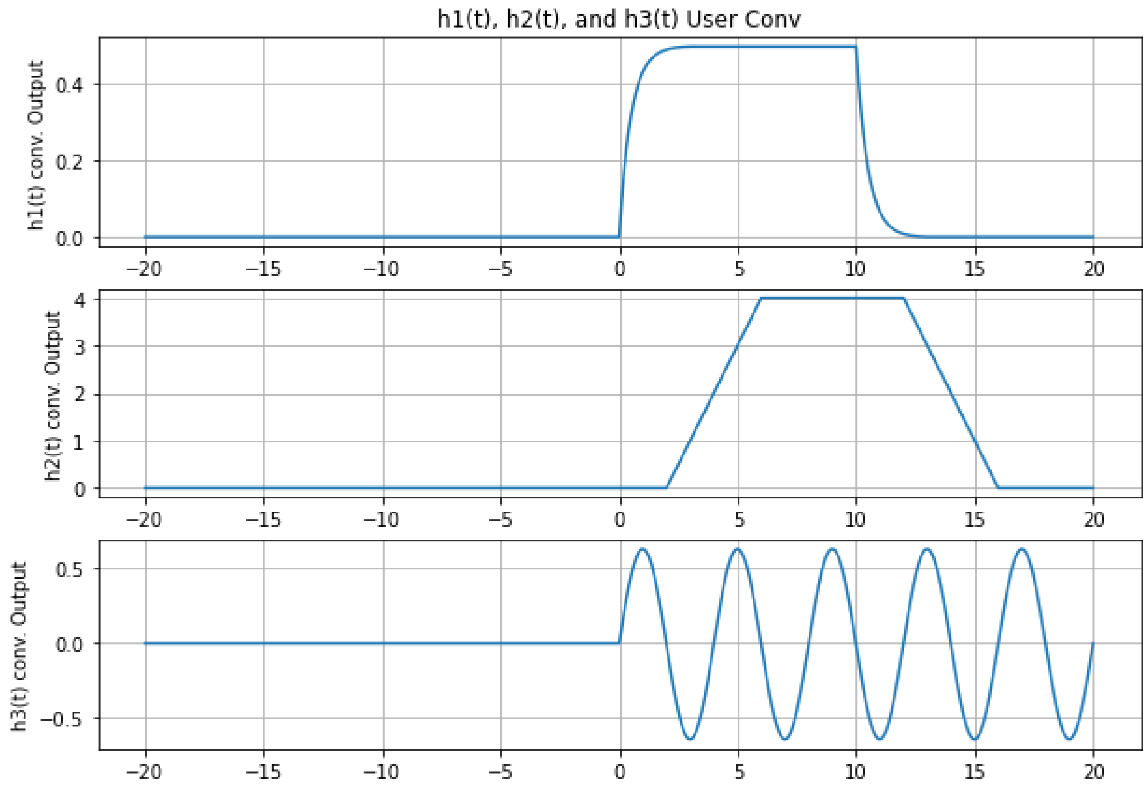
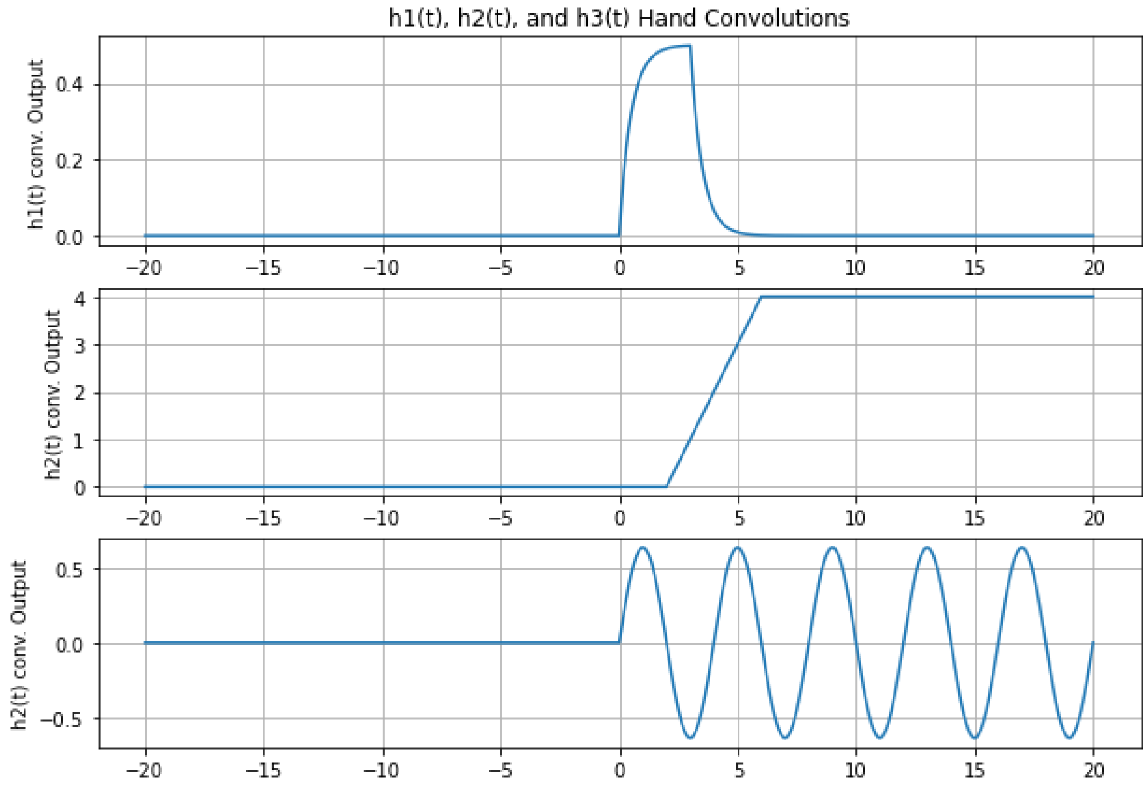


Figure 3: Hand Convolution



3 Question

Leave any feedback on the clarity of lab tasks, expectations, and deliverables.

Due to the (kind of) annoying way to get the TEX file from Overleaf (downloading a zip file then extracting the file), it may be a good idea to instead submit a zip file, that way all we have to do is copy the PDF and PY files into it (similar to what we do in 331, though the image files also come in the zip file).

Just an idea, I personally don't care either way :). There probably isn't any time saves either way anyway.