ECE 351 Lab 2

Andrew Hartman

September 2019

https://github.com/HartmanAndrew

Contents

1 Introduction

The purpose of this lab is to further learn the uses of the matplotlib.pyplot library when it comes to plotting functions including cosine, step function, ramp function, and derivatives. This lab also covers generation of user defined functions and then calling those functions for use later.

2 Equations

This lab included a few different equations that were programmed into individual python functions and then graphed. The main graph given can be defined using combinations of step and ramp functions.

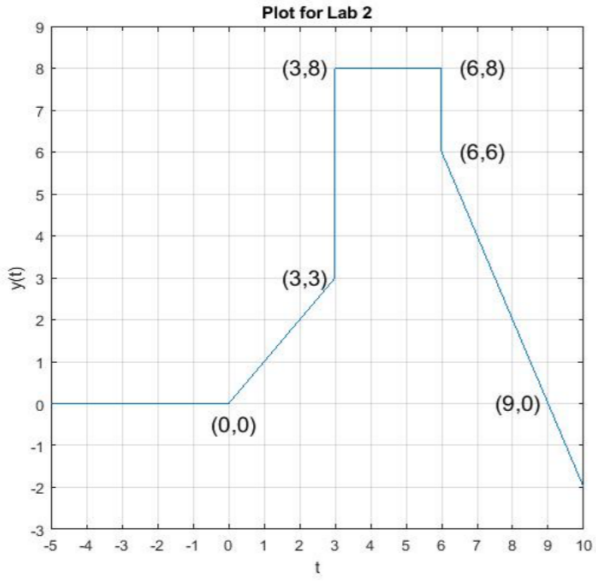
The step function is defined as:

$$u(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 0 \end{cases}$$

And the ramp function is defined as:

$$r(t) = \begin{cases} 0 & t < 0 \\ t & t \geq 0 \end{cases}$$

Using these two equations it is possible to create a function to define the following given graph.



Using the step and ramp functions as shown above the graph can be represented by the equation:

$$f(t) = r(t) - r(t-3) + 5u(t-3) - 2u(t-6) - 2r(t-6)$$

3 Methodology

3.1 Part 1

To begin the lab I looked through the provided example and then modified it to be able to graph $cos(t)$. In order to do this with a decent resolution the array of t values to graph across needed to be defined with fairly small step sizes.

```
steps = 0.01
t = np.arange(-5,10+steps,steps)
```

I then modified the example function given to graph $cos(t)$ using the matplotlib.pyplot library to graph. All of the functions in this lab were graphed using the same code just modifying the function being graphed and changing the axis values.

```
plt.figure(figsize=myFigSize)
plt.subplot(1,1,1)
plt.plot(t,func1(t))
plt.grid(True)
plt.ylabel('y(t)')
plt.title('Part1 Task2')
plt.xlabel('t')
plt.show()

def func1(t):
    y=np.zeros((len(t),1))
    for i in range(len(t)):
        y[i] = np.cos(t[i])
    return y
```

This function takes an array as an input defining the t values to evaluate the function across. It then creates an empty y array of zeros of the same length to assign the results of cos(t) to. Finally it loops through every element of t evaluating cos(t) and then assigning it to the appropriate y value.

3.2 Part 2

After this I defined functions to implement a step function and a ramp function in order to graph the equation for the graph given above.

```
def step(t):
    y = np.zeros((len(t),1))
    for i in range(len(t)):
        if t[i] > 0:
            y[i] = 1
        else :
            y[i] = 0
    return y
```

```
def ramp(t):
    y = np.zeros((len(t),1))
    for i in range(len(t)):
        if t[i] > 0:
            y[i] = t[i]
        else:
            y[i] = 0
    return y
```

These two functions work very similarly to the example and my user defined function. They loop through each value of t, evaluate and assign the proper value to y. During the evaluation though they check if the value of t is less than 0 or greater than or equal to 0 and specifies the proper value for the piecewise function.

These two functions were then tested and graphed before being implemented in func_stepRamp, my function that graphs the given graph above.

```
def func_stepRamp(t):
    func = (ramp(t))-(ramp(t-3))+(5*step(t-3))-(2*step(t-6))-(2*ramp(t-6))
    return func
```

This function just uses the ramp and step functions shifted or multiplied by values and then added together to form the final array func that is returned.

3.3 Part 3

Once the step and ramp functions were used to create the combined function, this function was then used to experiment how time shifting and scaling can be graphed. There were 6 different experiments outlined in the lab that were tested and plotted as subplots. The first test was to apply a time reversal to the function by replacing all t's with -t's. After that both f(t-4) and f(-t-4) were graphed to test shifting the function around. The next part of the lab asked to test ways to scale the function using f(t/2) and f(t*2). Finally the numpy equation diff() was utilized to graph the derivative of the function.

```
dt = np.diff(t)
funcTask5 = np.diff(func_stepRamp(t),axis=0)/dt
```

4 Results

4.1 Part 1

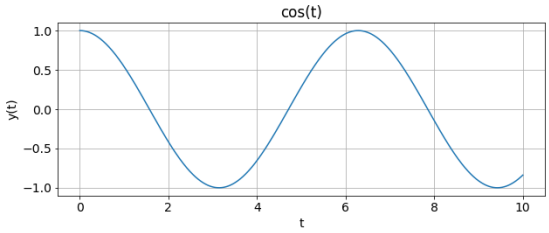


Figure 1: Part 1 Task 2 Cosine

Figure 1 Shows the outcome of func1 above that plots the cosine wave across the array t.

4.2 Part 2

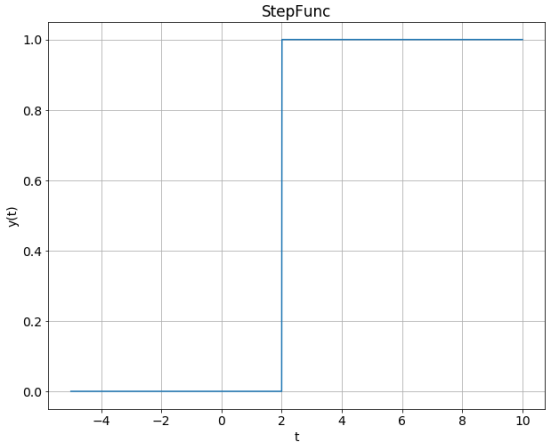


Figure 2: Part 2 Task 2 Step

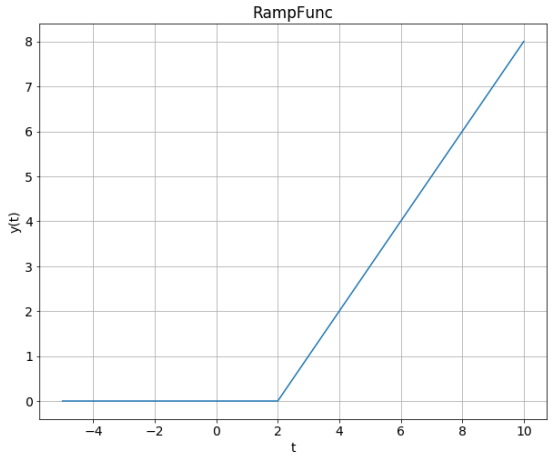


Figure 3: Part 2 Task 2 Ramp

Figure 2 is a plot of my step function, u(t-2), which shows that it can be shifted over 2 and still function properly. Figure 3 is a plot of my ramp function, r(t-2), which shows that this function also performs properly when shifted over 2.

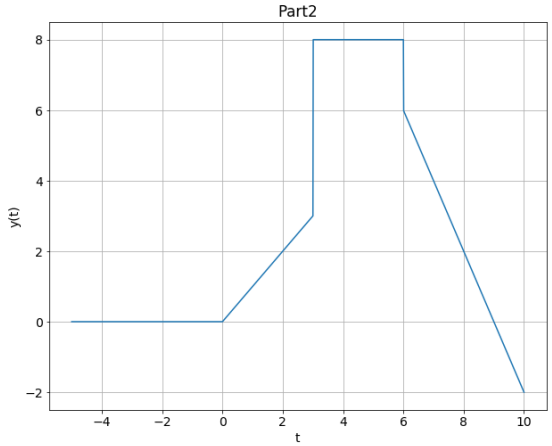


Figure 4: Part 2 Task 3

Using the step and ramp function the equation for the given graph was plotted as shown in Figure 4.

4.3 Part 3

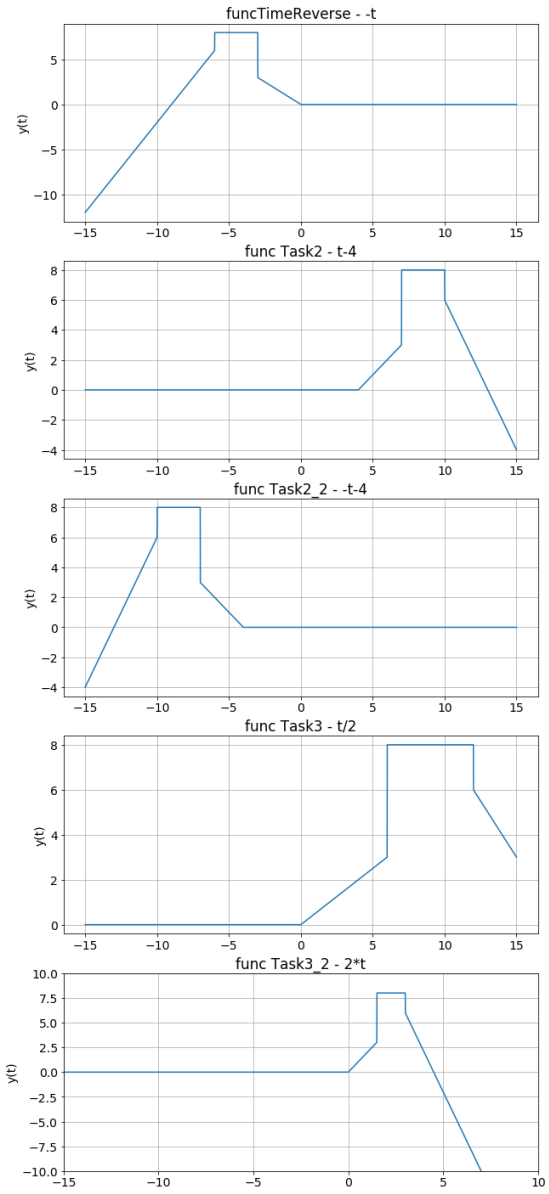


Figure 5: Part 3 Tasks 1-3

Figure 5 contains 5 different subplots of the different time shift and scale tests as explained above. The first graph is of f(-t) which flips the function about the y axis. This is followed by f(t-4) and f(-t-4) which shift the function right 4 and

also flip that new function about the y axis respectively. The last 2 graphs show f(t/2) and f(t*2) respectively which just scale up and down the given function along the x axis.

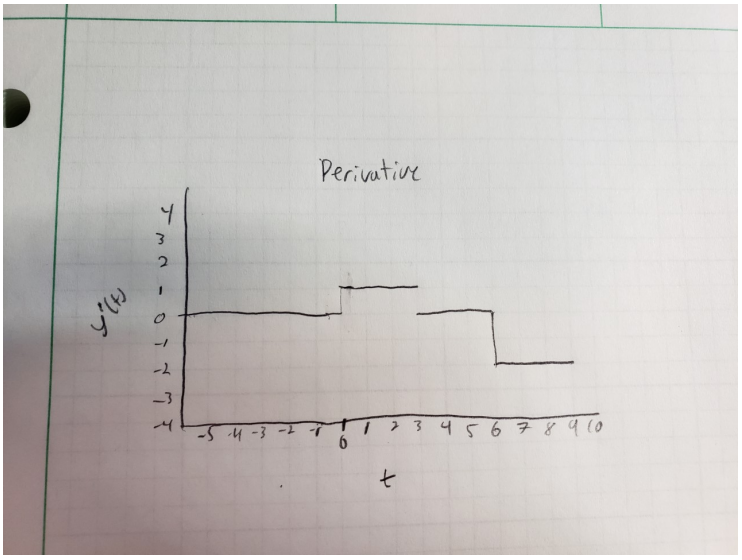


Figure 6: Part 3 Task 4 Derivative

Figure 6 shows my hand calculated derivative of the given graph. For the points where the graph goes vertical the derivative is technically undefined so I graphed it as jumps.

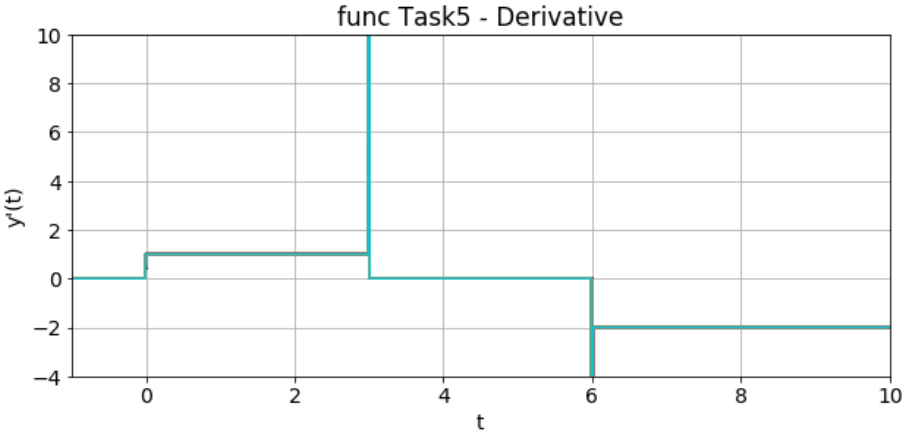


Figure 7: Part 3 Tasks 5

Figure 7 shows how python plots the derivative of the graph using the numpy.diff() function. Due to the derivative being undefined at the vertical spots it shoots off to infinity and negative infinity.

5 Error Analysis

Due to this lab being all simulation and plotting of simple equations no errors arose. I did have some difficulties in lab though figuring out how to utilize the numpy.diff() function, mainly having to modify the plt.plot() function t value to be able to plot the derivative as well.

6 Questions

1. Are the plots from Part 3 Tasks 4 and 5 identical? Is it possible for them to match? Explain why or why not.

The plots from task 4 and 5 are slightly different due to not technically being able to take the derivative of a step function so python graphs two parts to infinity and negative infinity. When graphed by hand it is possible to create jumps in the function but the plotting function can not handle it. Therefore, the computer generated graph would never be identical to my hand drawn graph.

2. How does the correlation between the two plots (from Part 3 Tasks 4 and 5) change if you were to change the step size within the time variable in Task 5? Explain why this happens.

If the step size were to change lower the undefined points of infinity and negative infinity would be graphed as more triangle shaped points of lower value due to not being able to evaluate directly at the point where the step function steps up. Where as if it were a smaller step it would graph the points more towards infinity.

3. In what way can the expectations and tasks be more clearly explained for this lab?

No ways.

7 Conclusion

In this lab I learned how to utilize the matplotlib.pyplot library a lot better than I previously knew as well as how equations can be defined in order to be graphed. Utilizing 2 different arrays of values, one to represent x values and the other to represent y values makes plotting simple. In future labs this will help with plotting and signals and equations and being able to properly define bounds and dimensions of the plots.