Signals and Systems Lab 2

Zachary Pfaff
github.com/ZachPfaff

February 1st, 2022

1 Introduction

Throughout Signals and Systems lab, students will be creating various user defined functions for graphing. It is the goal of the following lab to make students familiar with the syntax of user defined functions as well as provide more opportunity to program in python. Provided alongside this lab is some example code used for plots, this block of sample code will be used as reference for the lab tasks.

2 Cosine Function

The goal for this section is to recreate the following function by utilizing user defined functions and graphing it.

$$y = cos(t)$$

By following the code template provided from lab instructions, the function was able to be created. In order to make the graph high resolution, an array of steps were created between the values of 0 and 5. Each step was very small so although python is plotting thousands of lines, to the viewer it looks like a nice cosine graph. The function was then defined in the for loop by changing a line of code from a sin function to a cos function. The if statement was also removed since the graph is only concerned with one function. Listing 1 below shows the code used to implement the cosine graph.

```
steps = 1e-2
t = np.arange(0, 5 + steps, steps)
print('Number of elements: len(t) = ', len(t), '\nFirst Element: t[0] = ', t[0],
      ' \nLast Element: t[len(t) - 1] = ', t[len(t) - 1])
def cos(t):
    y = np.zeros(t.shape)

    for i in range(len(t)):
            y[i] = np.cos(5*t[i])

    return y
y = cos(t)
```

Listing 1: Python example

This change in the code template resulted in the proper cosine graph shown in Figure 1. Notice that, unlike the example graph shown in the lab handout, the cosine function starts at zero due to only one function being inside the for loop.

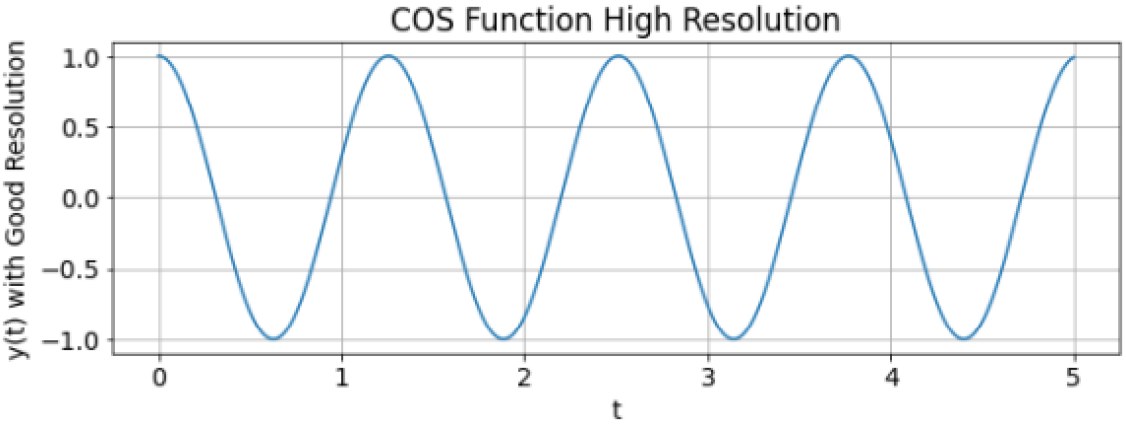

Figure 1: User Created Cosine Function

3 Step and Ramp Functions

With our newly learned user defined functions, the following section will focus on creating two new functions as commonly used in signals and systems. The goal of this section is to create a step and ramp function, then use both functions to re-create the plot in the lab instruction handout.

The first function I created was the step function. To create this function I referred to the following definition of a step function.

$$u(t) = \begin{cases} 0 & \text{for } [t < 0] \quad (1) \\ 1 & \text{for } [t \geq 0] \quad (2) \end{cases}$$

In order to implement this function into my code, I had to use an if statement inside the for loop to describe the two possible outputs from a given input. To do this, I looked at every index for input t and set the function equal to either 0 or 1 based on if that index was positive or negative. In doing so, I was able to create the following graph below in Figure 2. Listing 2 will also show my step function code.

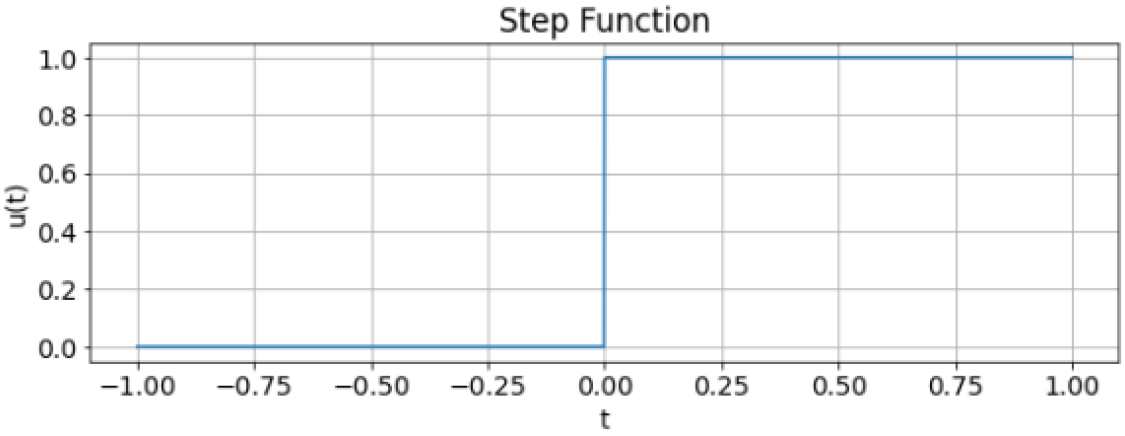

Figure 2: User Created Step Function

```
def step(t):
    y = np.zeros(t.shape)

    for i in range(len(t)):
        if t[i] < 0:
            y[i] = 0
```

```
        else:
            y[i] = 1

    return y

y = step(t)
```

Listing 2: Step Function Code

The next function I implemented was the ramp function which was similar to the step function. The only difference here was the output of the function would be equal to the input if the input is a positive number. The definition of the ramp function is shown below.

$$r(t) = \begin{cases} 0 & \text{for } [t < 0] \\ t & \text{for } [t \geq 0] \end{cases} \tag{3}$$
$$\tag{4}$$

The only difference in my code compared to the step function is line 8. By changing the output to "t" instead of 1 I was able to create the following ramp plot in Figure 3. Listing 3 shows the code used to implement the ramp function.

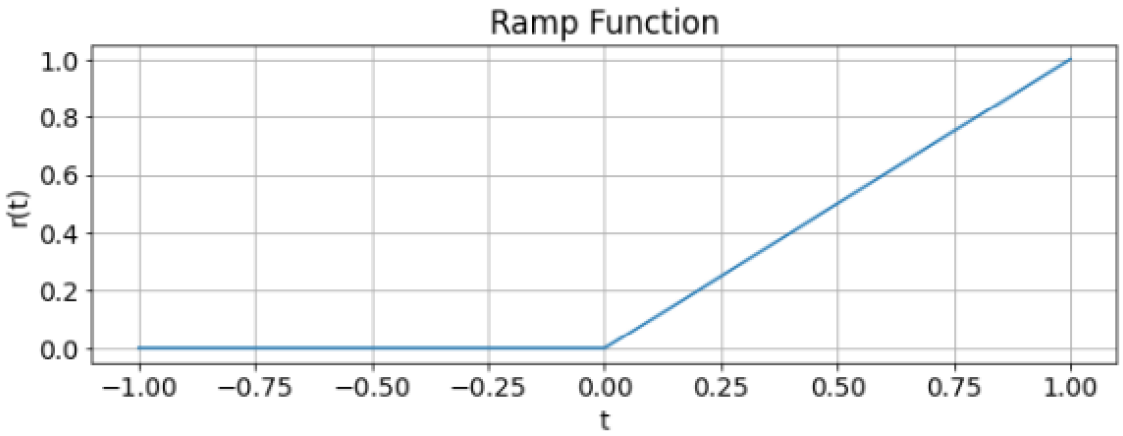

Figure 3: User Created Ramp Function

```
def ramp(t):
    y = np.zeros(t.shape)

    for i in range(len(t)):
        if t[i] < 0:
            y[i] = 0
        else:
            y[i] = t

    return y

y = ramp(t)
```

Listing 3: Ramp Function Code

The last task for this part of the lab was to utilize both my step and ramp functions to re-create the graph in the lab handout. The graph ranges from -5 to 10 and has ramp and step functions of various magnitudes. The equation below is used to create this graph.

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

In order to implement this function into python, I used the following code in Listing 4. Notice that it uses both the user defined step and ramp functions.

```
def lab(t)
    y = ramp(t) - ramp(t-3) + 5*step(t-3) - 2*step(t-6) - 2*ramp(t-6)
    return y
y = lab(t)
```

Listing 4: Lab Plot Code

Plotting this function perfectly recreated the plot from the lab instruction handout. The plot is shown below in Figure 4.

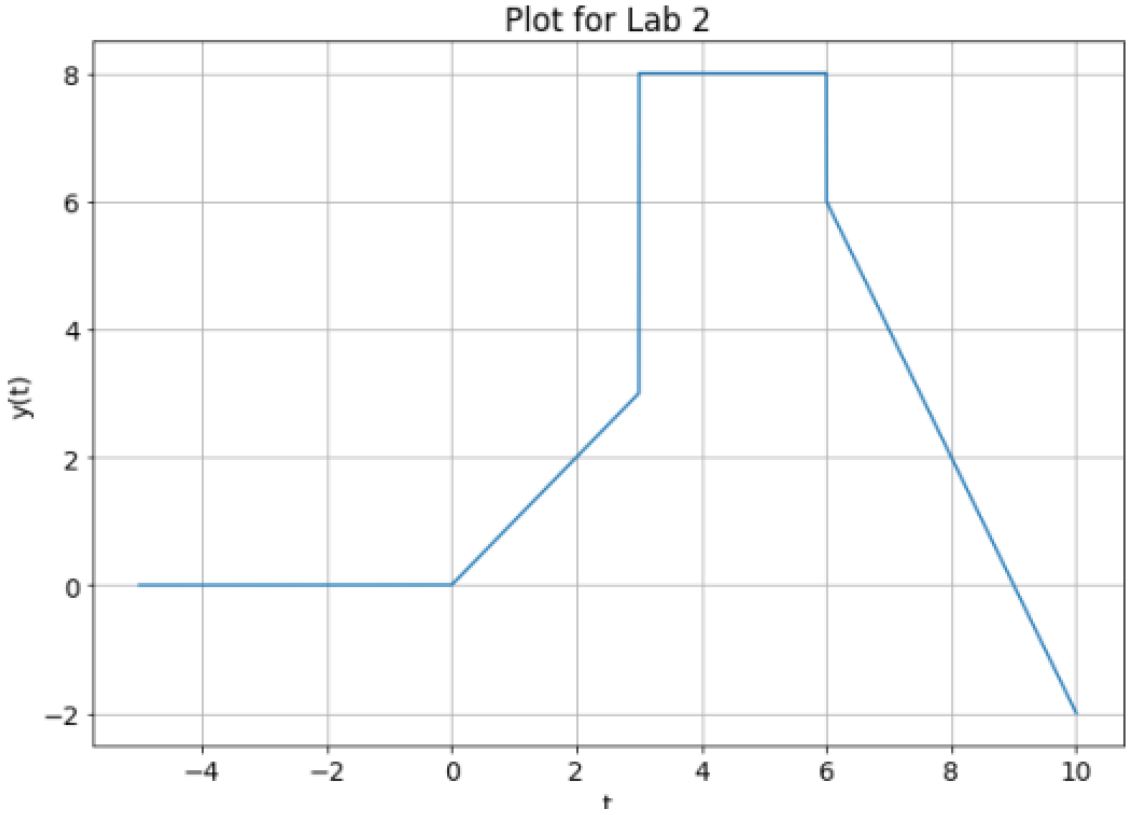

Figure 4: Plot for Part 2

4 Time Shifting and Scaling Operations

The final section of this lab will focus on time shifting and scaling the plot from the previous section. The first task is to apply a time reversal to the plot, which can be done by inverting the function. The next operation is a time shift to the plot. The third operation is to apply scaling to the plot. Each plot and their respected operation are shown below in Figure 5. Note that I had to adjust the parameters of each plot to fully display the graph.

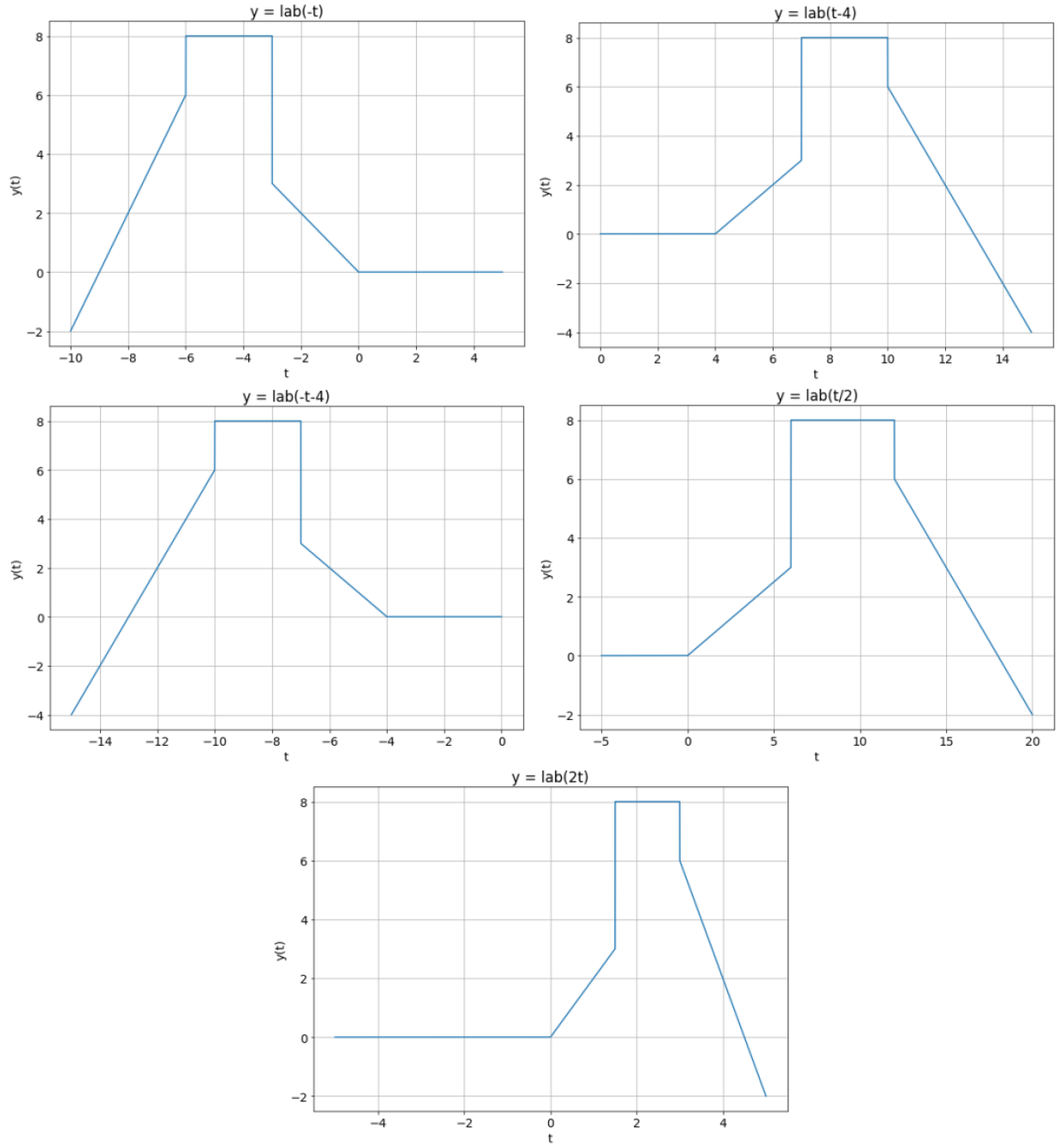

Figure 5: Plots with Time Shifts and Scaling Operations

Finally, the derivative of the user defined function will be calculated and plotted in python. By looking at the plot I created a graph of what I expected the derivative to look like. I treated the ramp functions as step functions, since the derivative of a ramp function is a step function, while the step functions were treated as impulses. After it was drawn out, I used the numpy.diff() command to calculate and plot the derivative. Figure 2 below compares the plot I made by hand to the plot python created in steps of 1. Notice that python graphed plot requires a ramp to graph each point whereas the hand drawn plot has very sharp points.

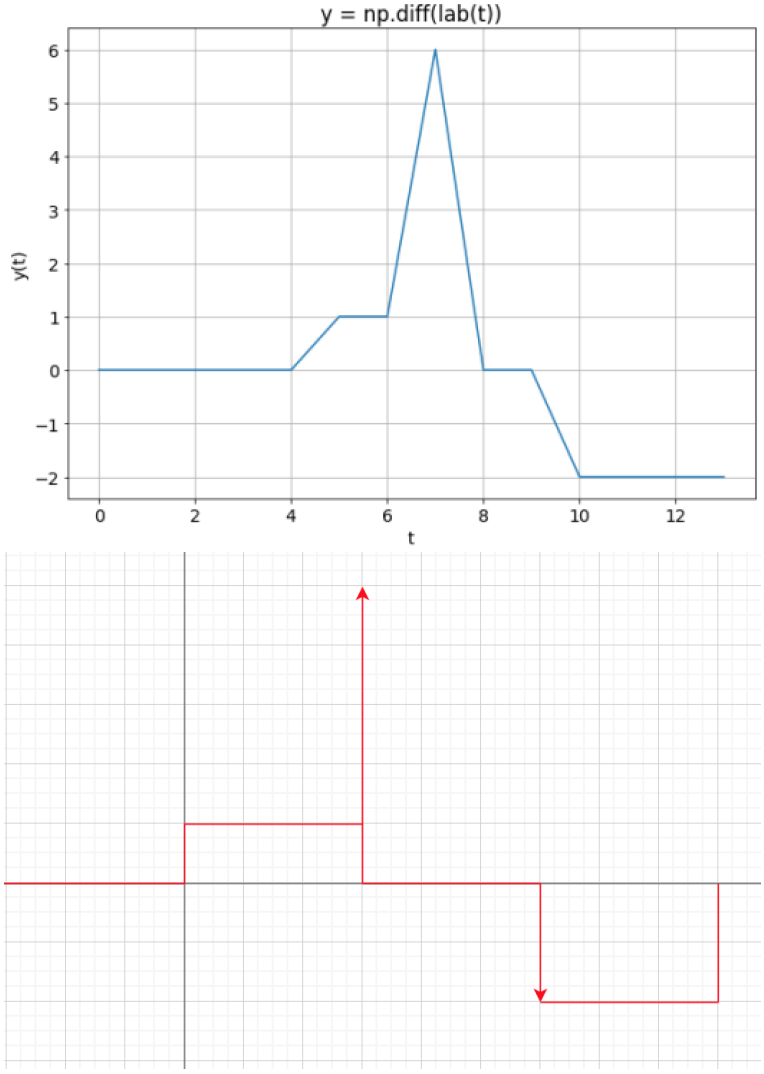

Figure 6: Comparison of the expected vs graphed derivative functions

5 Questions

5.1 Are the graphed and hand drawn plots identical? Is it possible for them to match?

The two separate plots are identical in that both treats the ramp function derivative as a step function, and the step function as an impulse. The difference between the two is the resolution which can be adjusted, but even then it will not be possible to perfectly match each other.

5.2 How does the correlation between the two plots change if you were to change the step size within the time variable?

By changing the step size within the time variable, the graphed function will have sharper impulses but it's ramp functions will be more shallow and last longer on the time interval. Because the step size is increased, the ramp functions need to cover the area while also having a longer t interval to work with. For this reason, a graph with significantly more steps will completely hide the ramp functions as shown in Figure 7 below.

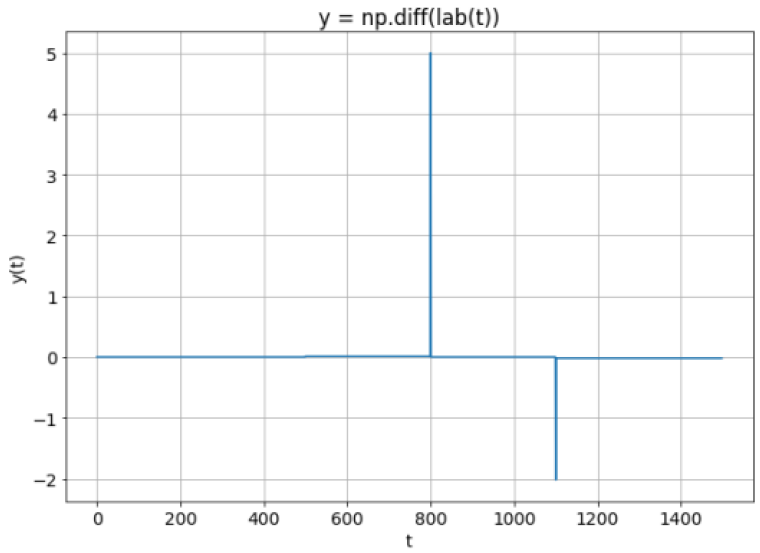

Figure 7: Derivative Plot With More Steps

5.3 Lab Feedback

Some difficulty I faced for this lab was figuring out how to work the numpy.diff() function. I could not find the syntax for it in the lab folders, so I had to use the internet to find out how to do it.