ECE 351 - Section 51

User-Defined Functions

Lab 1

Submitted By:
Tristan Denning

Contents

1 Introduction

The purpose of this lab is to practice creating user-defined functions in Python and performing various signal operations including time shifting, time scaling, time reversal, signal addition, and discrete differentiation.

*All of the code used to accomplish the goals of this lab can be accessed at my Github page: `https://github.com/Tristan-Denning`

2 Equations

Unit Step Function

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

Ramp Function

$$r(t) = \begin{cases} t & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{2}$$

User-Defined Function - Derived from equations (1) and (2)

$$y(t) = r(t) - r(t-3) + 5u(t-3) - 2u(t-4) - 2r(t-6) \tag{3}$$

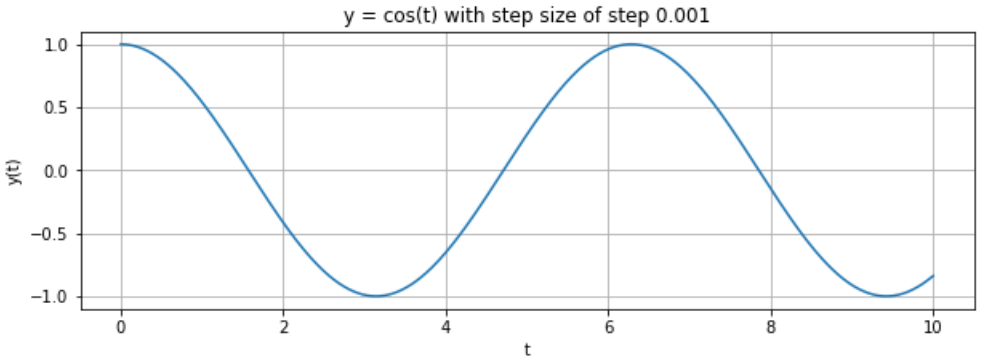
3 Methodology

The time shifting, scaling and linearity properties of the Unit Step (1) and Ramp (2) Functions were used to derive the user defined function y(t). Scaling the Unit Step function changes the max value of the function. Scaling the Ramp function changes its slope. For both the Unit Step and Ramp functions, time shifting can be done by adding or subtracting a constant from the time variable. In part three of this lab, the time reversal property is also used. For time reversal, the variable t is set to its negative value, and the corresponding function is essentially mirrored across the y-axis. Part three also relies on fundamental derivative geometry - i.e. a horizontal line differentiates to 0, and a sloped line differentiates to a horizontal line at the value of the slope.

4 Results

Part 1

Task 2: Graph of cos(t)



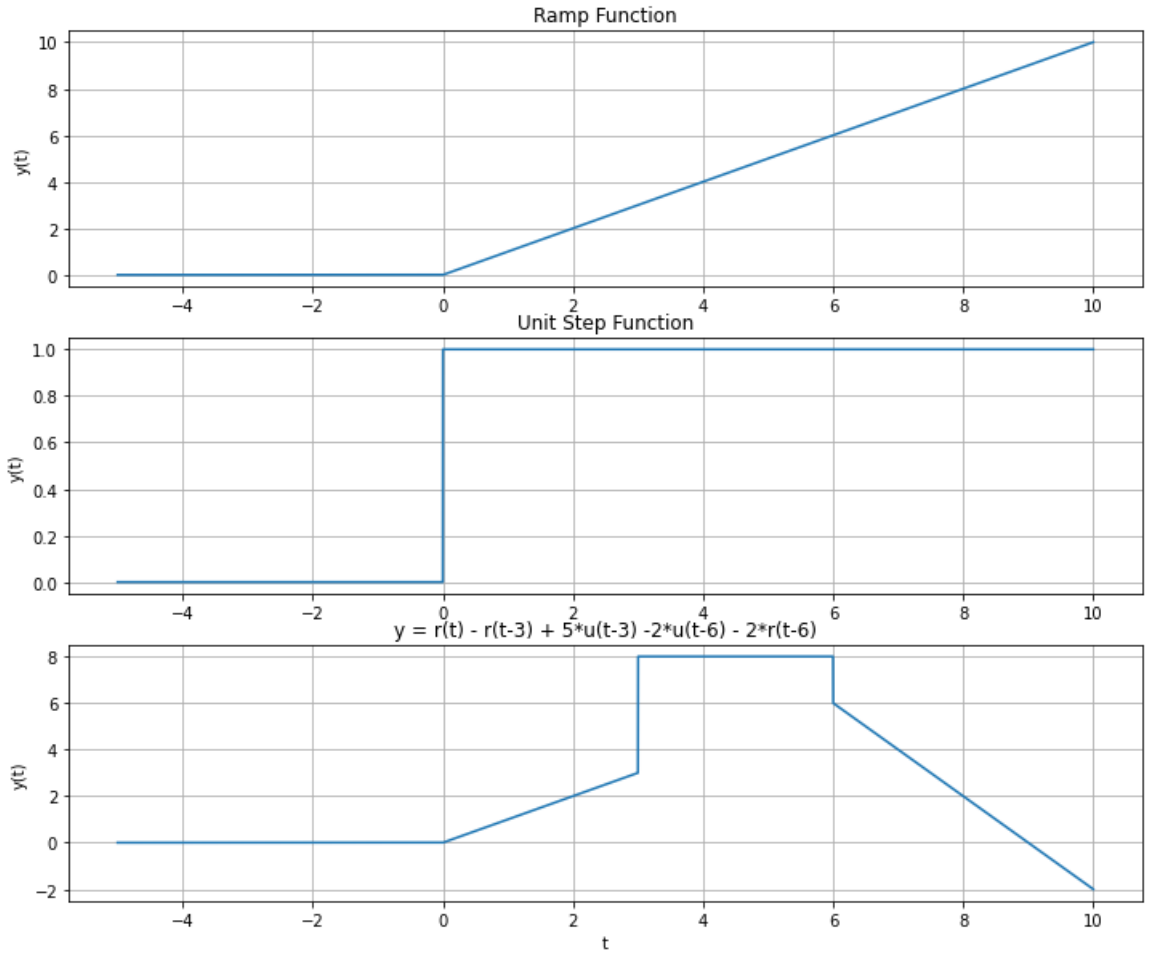
Task 2: Code for y = cos(t)

```
            def CosFunc(t):
                y = np.zeros(t.shape)
                for i in range(len(t)): #Runs the loop for each
increment of t
                    y[i] = np.cos(t[i]) #y = cos(t) for t on [0,10]
                return y
            y = CosFunc(t) #calls the cosine function
            # --- Create The Plot ---
            plt.figure(figsize = (10, 7))
            plt.subplot(2, 1, 1)
            plt.plot(t, y)
            plt.grid()
            plt.ylabel('y(t)')
            plt.xlabel('t')
            plt.title('y = cos(t) with step size of step 0.001')

```

Part 2

Task 1: See equation (3) and methodology.

Task 2 and 3: Plots for u(t), r(t), and y(t)



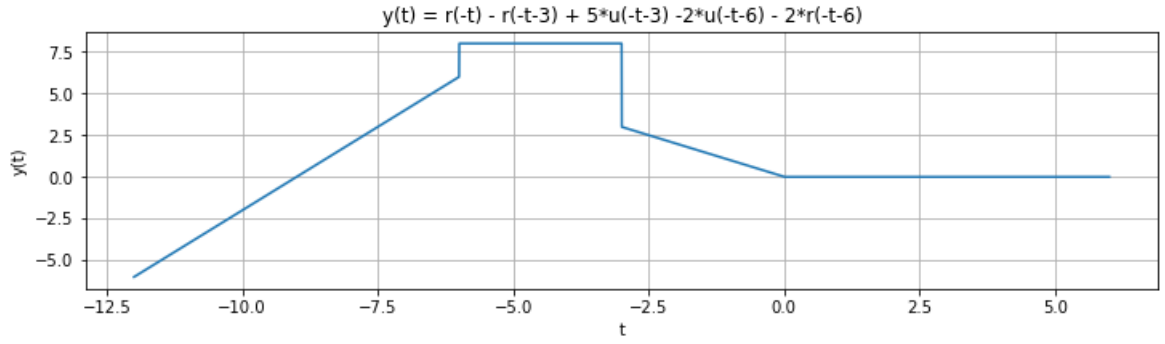
Code for u(t), r(t), y(t) and their plots

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

        for i in range(len(t)): #Runs the loop for each increment
    of t
            if (t[i] > 0):
                y[i] = t[i]
            else:
                y[i] = 0
            #y[i] = (t[i] if t[i] >=0 else 0)#y = cos(t) for t on
    [0,10] # This line is an alternative definition

        return y
```

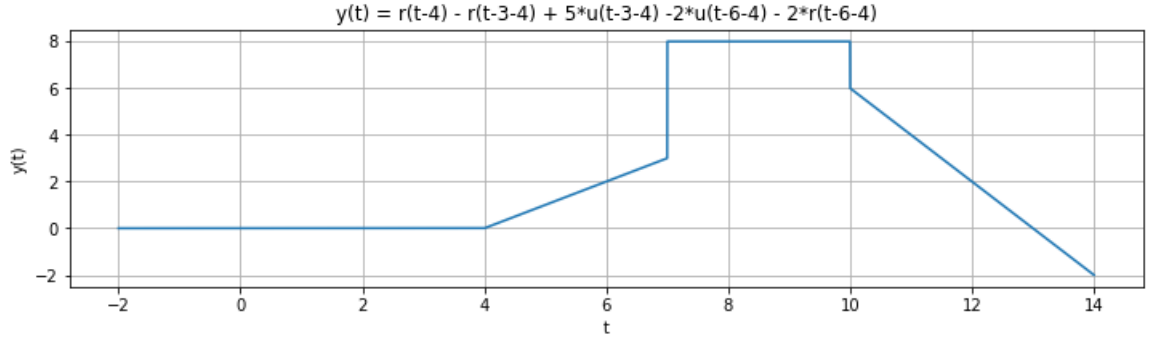
```
12
13      def unitstep(t):
14          y = np.zeros(t.shape)
15
16          for i in range(len(t)):
17              if (t[i] > 0):
18                  y[i] = 1
19              else:
20                  y[i] = 0
21          return y
22
23      r = ramp(t)
24      u = unitstep(t)
25
26      y = ramp(t) - ramp(t-3) + 5*unitstep(t-3) -2*unitstep(t-6) -2*
        ramp(t-6)
27
28      plt.figure(figsize = (12, 10))
29
30      plt.subplot(3, 1, 1)
31      plt.plot(t, r)
32      plt.grid()
33      plt.ylabel('y(t)')
34      plt.title('Ramp Function')
35
36      plt.subplot(3, 1, 2)
37      plt.plot(t, u)
38      plt.grid()
39      plt.ylabel('y(t)')
40      plt.title('Unit Step Function')
41
42      plt.subplot(3, 1, 3)
43      plt.plot(t, y)
44      plt.grid()
45      plt.ylabel('y(t)')
46      plt.xlabel('t')
47      plt.title('y = r(t) - r(t-3) + 5*u(t-3) -2*u(t-6) - 2*r(t-6)')
48
```

Part 3

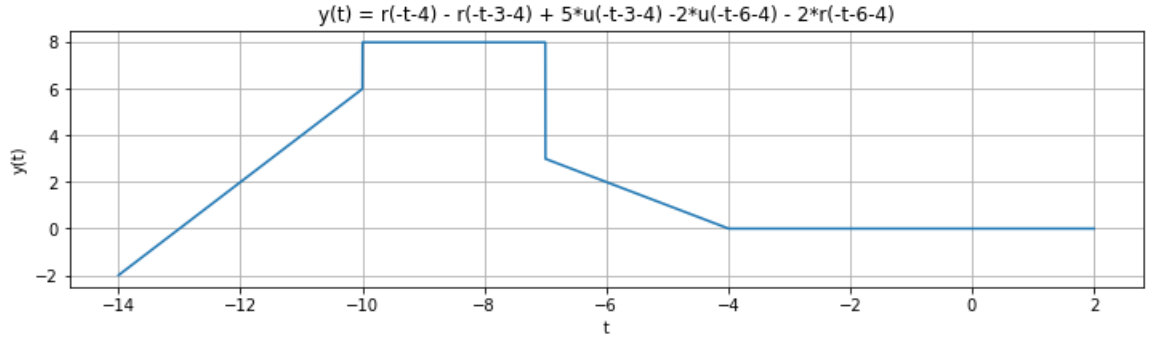
Task 1: Time Reversal of y(t)



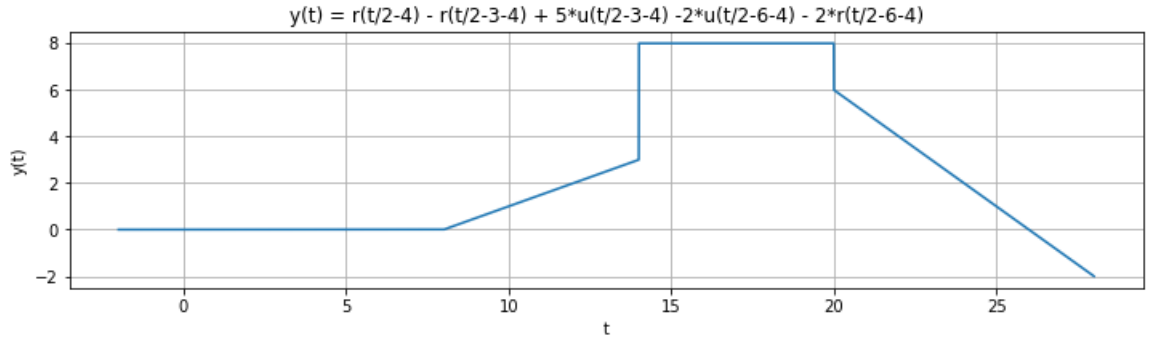
Task 2.1: Implement the time shift t-4 to y(t)



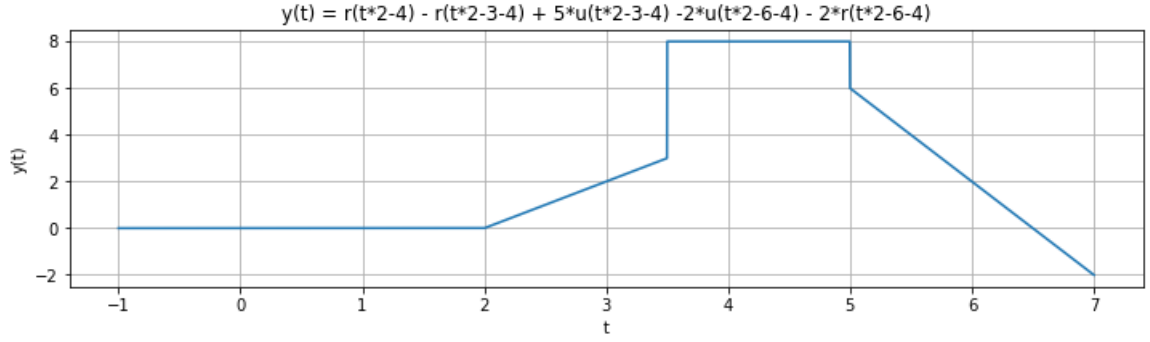
Task 2.2: Implement the time shift and reversal -t-4 to y(t)



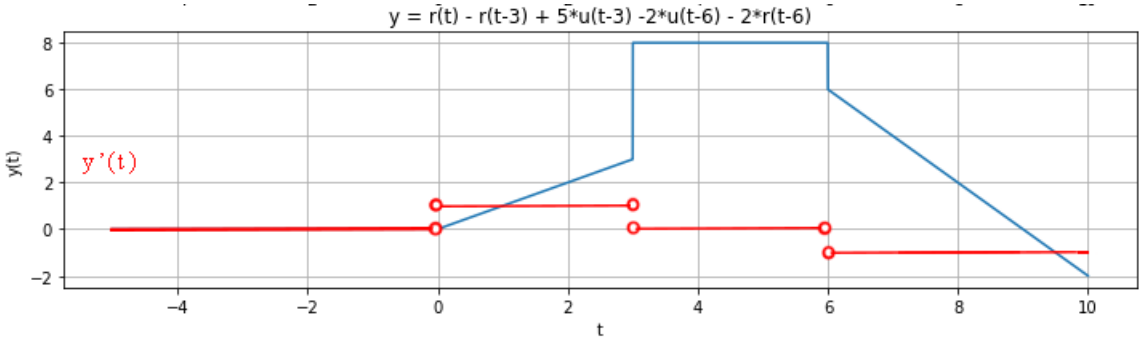
Task 3.1: Apply time scale operation t/2 to y(t)



Task 3.2: Apply time scale operation 2t to y(t)

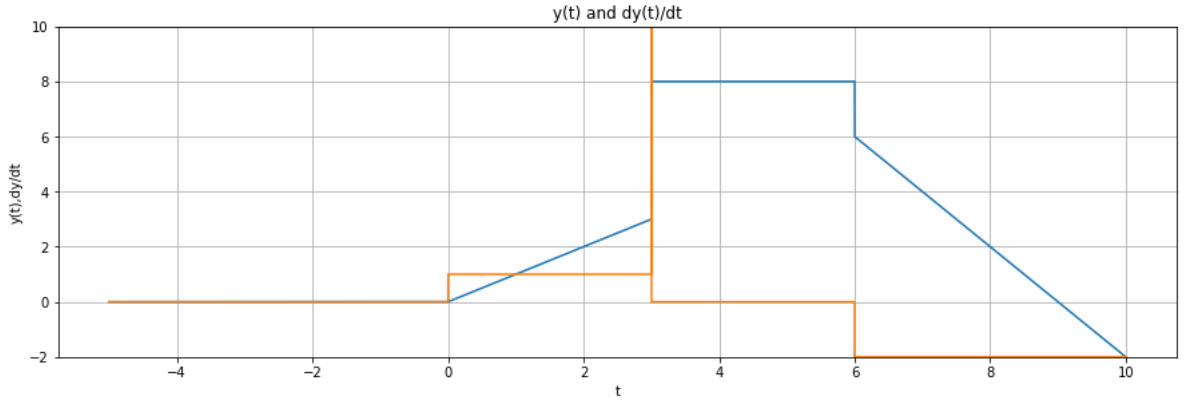


Task 4: Hand-plotted derivative of y(t)



"Hand Plotted" on top of the original y(t) using Adobe Photoshop

Task 5: Use numpy.diff() to plot the derivative of y(t).



5 Error

I only experienced difficulty during this lab when trying to plot the derivative of y(t). I kept receiving the compiler error "x and y must have same first dimension, but have shapes (15001,) and (15000,)". This occurred while trying to plot the numpy.diff(y(t)) on the range of t - In other words I was trying to plot dy(t)/dt as a function of time. The numpy.diff() uses an algorithm that decreases the range of the new function by one increment. So I was able to fix the error by plotting dy(t)/dt as a function of t on the range of len(dy/dt). The new plot statement looked like this:

```
1 plt.plot(t[range(len(dy))] , dy )
```

So instead of plotting a function with a range of 15000 on a time interval of 15001, the time interval was now 15000.

6 Questions

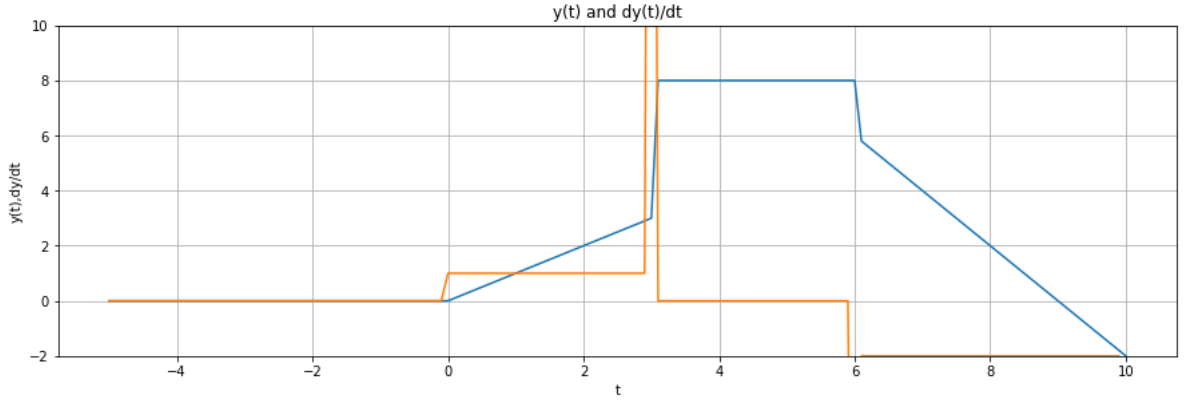
1. Are the plots from Part 3 Task 4 and Part 3 Task 5 identical? Is it possible for them to match? Explain why or why not.

   (a) No, it is not possible for the hand drawn plot to match the computer generated plot. The computer sees vertical lines as having a slope of some rise/0. The numpy.diff() function seems to treat a fraction with zero in the denominator as infinity - this is useful for some applications, but incorrect for simple derivation. For regular functions, any "sharp corners" or vertical lines should yield a discontinuity on the derivative

graph, as these points are not differentiable. A smarter numpy.diff would represent these discontinuities with open circles, as I have done on my hand-plot.

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

   (a) When the number of steps within the time variable are decreased, the derivative graph shows sloped lines rather than nearly vertical lines at the discontinuities. This is because the program has less values to sample from while approaching these points.



Step size changed from 0.001 to 0.1

7 Conclusion

The objectives for this lab were fulfilled as I was able to successfully define various functions, perform time-scaling and time-shifting operations on them, take the derivative and plot the results for each operation. In this Lab I learned how to create functions in python and how to deal with the associated nuances, like the weird syntax for creating and plotting a derivative. I am certain that derivatives will make an appearance in future labs, so I am glad to have learned the code for them early on.