University of Idaho

ECE 351 - Section #53

Lab #2 Report

User-Defined Functions

Submitted To:
Kate Antonov
University of Idaho
kantonov@uidaho.edu

Submitted By :
Macallyster S. Edmondson
University of Idaho
edmo7033@vandals.uidaho.edu
github.com/mac-edmondson

Tuesday 1st February, 2022

Contents

1 Introduction

The goal of this lab was to introduce the idea of user-defined functions in Python and utilize them to create causal functions we've learned about in **ECE 350**. We then utilized these functions to demonstrate signal operations including time shifting, time scalin, time reversal, signal addition, and discrete-differentiation. In order to complete this lab, the Python programming language was used with the *Spyder-IDE* downloaded using *Python Anaconda3*. The packages used in the completion of this lab were `numpy` for definitions of mathematical functions, and `matplotlib.pyplot` to plot outputs of functions seen later in this lab. All code for this lab, including this report, can be found on my Github.

2 Equations

The equations used within this lab are shown in this section. The equations will be referenced by number throughout the rest of the report.

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

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

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

3 Methodology

3.1 Lab: Part 1

In Part 1 of this lab, the goal was to implement $y = \cos(t)$ and plot the output based on the example code give in the lab handout. The main goal of this part was to find an optimal step size for good resolution curves. The code for this part of the lab can be seen below and the graph can be seen in Figure 1.

```
# PART 1:

def func1(t) :
    return np.cos(t)

step_size = .1
t = np.arange(0, 10 + step_size, step_size)
y = func1(t)

plt.subplot(1, 1, 1)
```

```
plt.plot(t, y)
plt.grid()
plt.ylabel('cos(t)')
plt.xlabel('t')
plt.title('cos(t) v. t ; from 0 to 10, inclusive')
plt.show()
```

3.2 Lab: Part 2

In Part 2 of this lab, the implementation of Equation (3) was the overall goal. Equation (3) was derived from a given graph in the lab handhout. In order to to implement this equation, Equations (1) & (2) needed to be defined in Python. Implementation of all three of the equations in Python can be seen in the source code below. (Code for plots is not shown to reduce report length. See Github for full code.) Each equation was implemented in it's own function, with the unit step function and ramp functions being defined with if-else statements. Then, the implementation of Equation (3) utilized both of those function definitions. The plots the code generates can be seen in Figures 2 & 3. As can be seen in those plots, each function was working as expected.

```
# PART 2:

def u(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

def r(t) :
    y = np.zeros(t.shape)

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

    return y

def fig2_func(t) :
  y = r(t) - r(t-3) + 5 * u(t-3) - 2 * u(t-6) - 2*r(t - 6)
  return y
```

3.3 Lab: Part 3

In Section 3 of this lab, we used the function that implemented Equation (3) to perform time-shifting and scaling operations. Additionally, we took the discrete derivative of the function using the `numpy.diff()` function. As seen in Figures 4, 5, & 6, basic time reversal, time-shift, and time scaling operations were performed on Equation (3), respectively, using Python. This was as simple writing it out by hand, just modifying the input to the defined function before plotting it as seen in the code below.

```
#PART 3:

step_size = 1e-3

#1
t = np.arange(-10, 5 + step_size, step_size)
y = fig2_func(-t)

#2

t = np.arange(-1, 14 + step_size, step_size)
y = fig2_func(t-4)

#3

t = np.arange(-1, 10*2 + step_size, step_size)
y = fig2_func(t/2)
```

Implementation of the discrete derivative was more complex. It is important to remember $y'(t) = \frac{\Delta y(t)}{\Delta t}$. Then, that derivative is still plotted against time. Notice in the code below, t has to be shortened to the length of `dy` as there is one less value in the `dy` & `dt` arrays from the `numpy.diff()` function. (Seek `numpy` documentation for more details.) Additionally, it is important to set the bounds of the y-axis using the `matplotlib.pyplot.ylim()` function, due to the very large rate of change from step functions in the discrete derivative. The plot of the discrete derivative can be found in Figure 7.

```
#4

t = np.arange(0, 10 + step_size, step_size)
y = fig2_func(t)

dt = np.diff(t, axis=0)
dy = np.diff(y, axis =0)

plt.figure(figsize = (10, 7))
plt.subplot(1, 1, 1)
plt.plot(t[range(len(dy))], dy/dt)
```

```
12  plt.ylim(-3, 10)
13  #...
```

4 Results

The results of this lab are very straightforward. The implementation of all functions worked as expected and there is not much to discuss that wasn't covered in the Methodology Section of this report.

It is important to note the difference in the discrete derivative of Equation (3) found using python, Figure 7, and the hand-drawn version of the same derivative, Figure 8. These derivatives look very similar but in the discrete derivative, there are large spikes in the rate-of-change where there are discontinuities in the hand-drawn plot. The reason for this is that the discrete derivative does not have an *absolute* instantaneous rate-of-change since there are a limited number of steps in `y`. With theoretical math, we know the rate of change is instantaneous at the points of discontinuity from Equation (1). This being said, these large spikes in rate of change can be considerd as discontinuities.

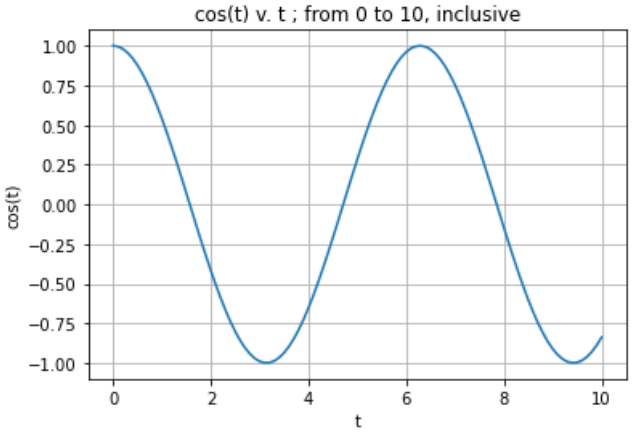

Figure 1: Part 1, Task 2 - Cosine Function Implementation

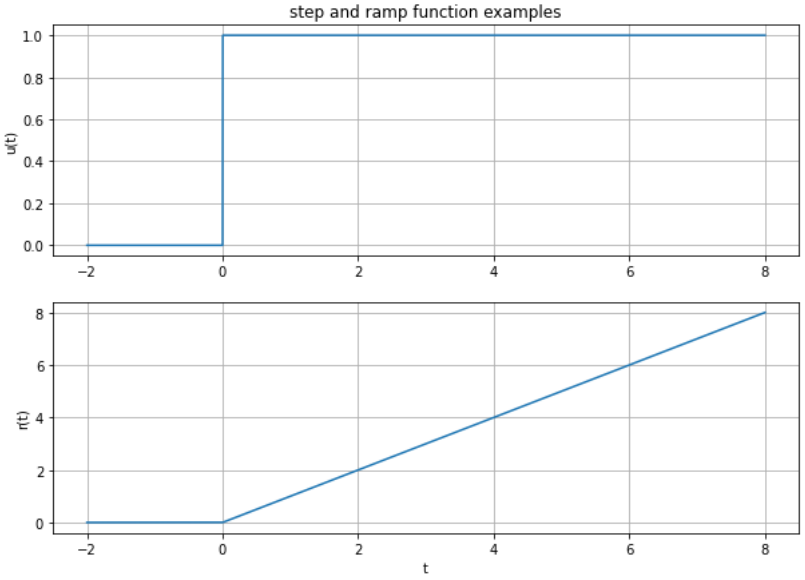

Figure 2: Part 2, Task 2 - Step and Ramp Function Implementation

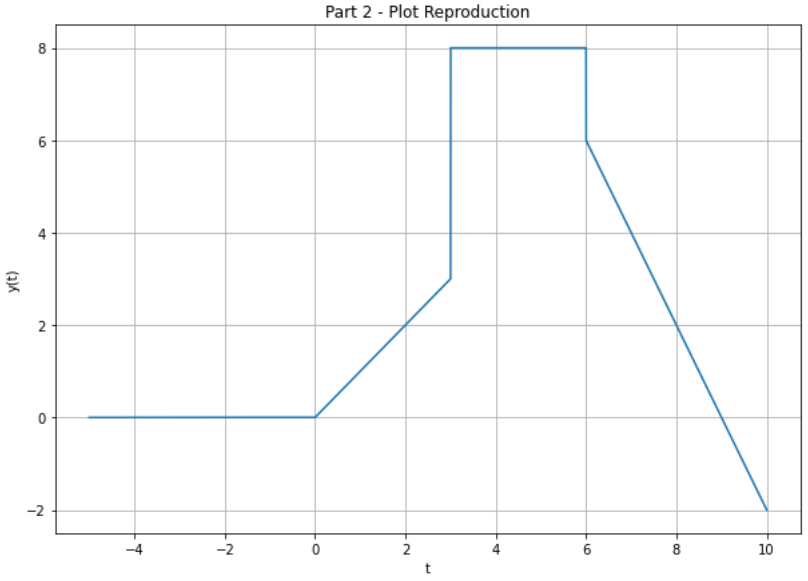

Figure 3: Part 2, Task 3 - Implementation of Equation (3)

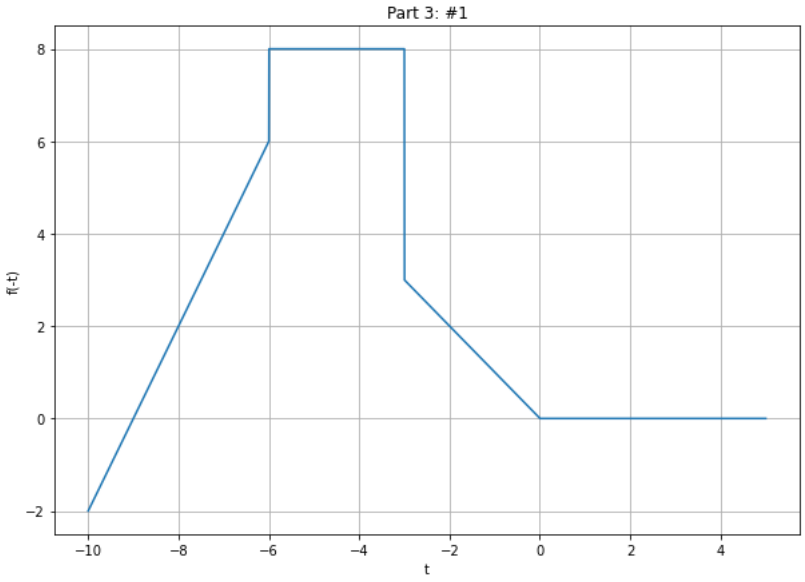

Figure 4: Part 3, Task 1 - Time Reversal Operations

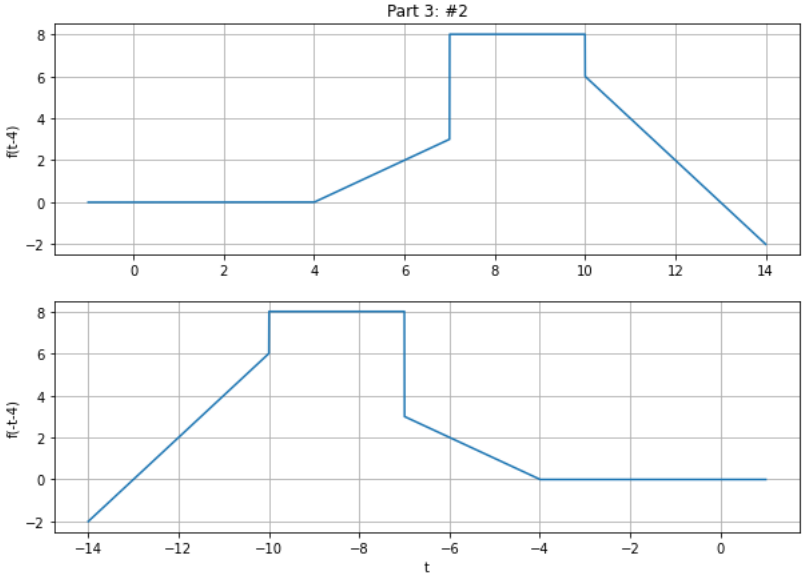

Figure 5: Part 3, Task 2 - Time-Shifting Operations

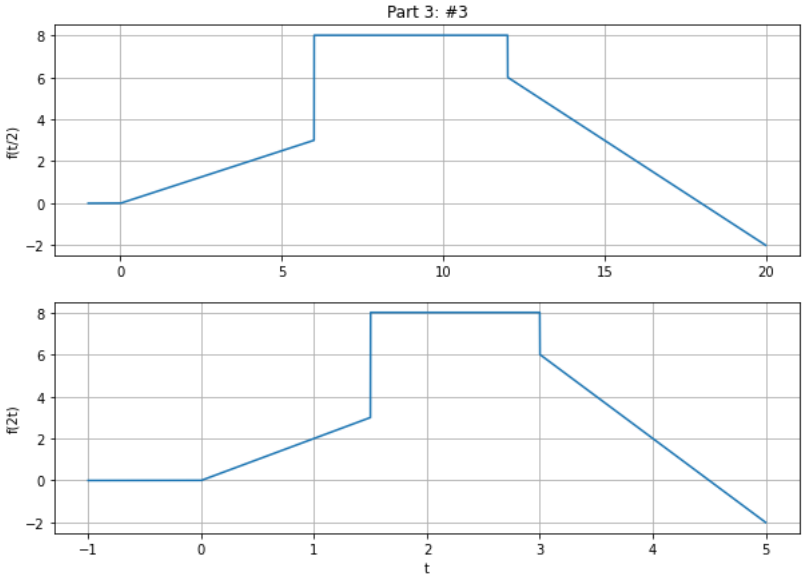

Figure 6: Part 3, Task 1 - Time Scaling Operation

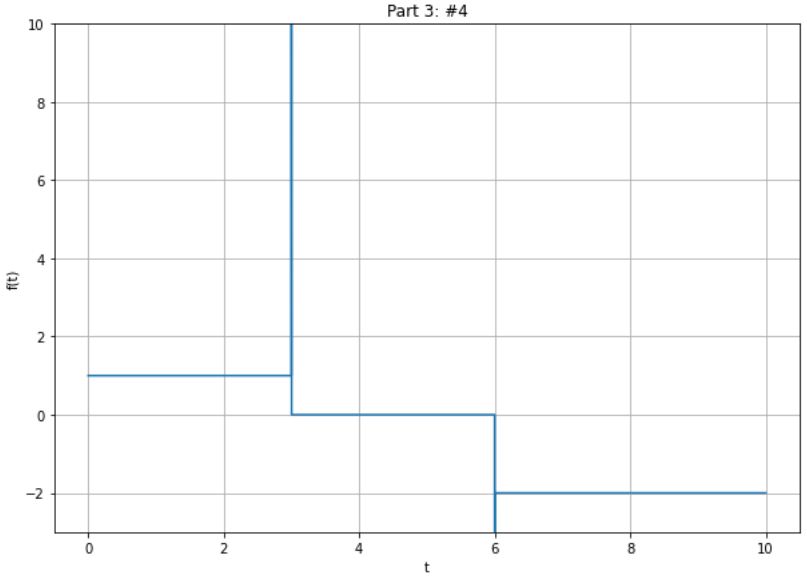

Figure 7: Part 3, Task 4 - Discrete Derivative of Equation (3)

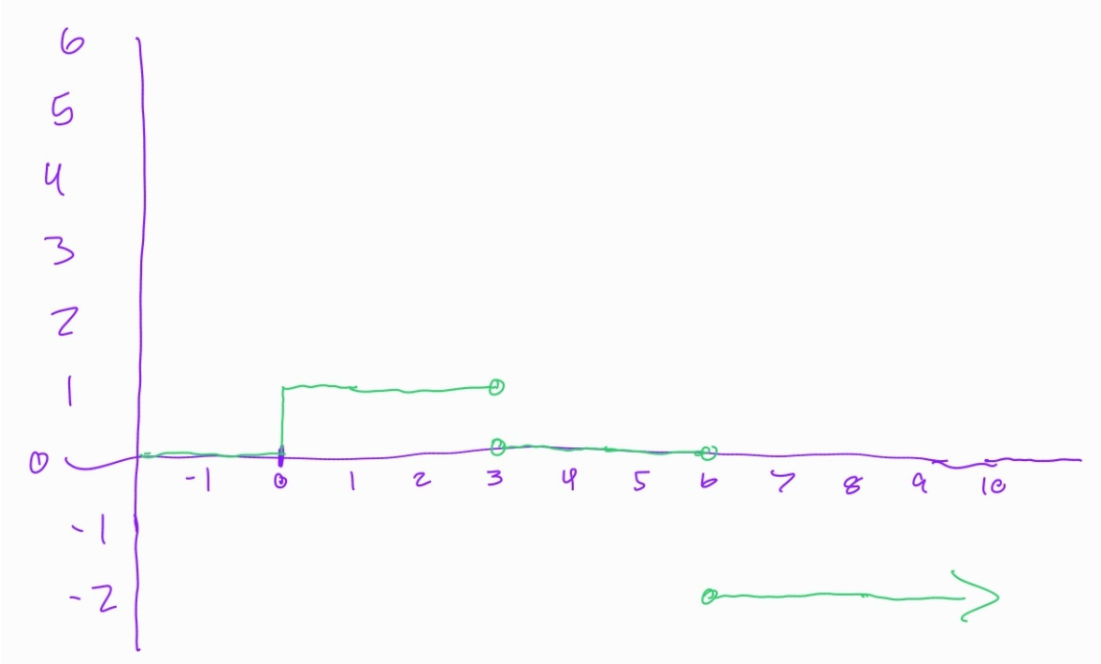

Figure 8: Part 3, Task 4 - Hand Drawn Derivative of Equation (3)

5 Error Analysis

One problem that occurred during this lab which took up most of the lab time was figuring out how to take the discrete derivative of Equation (3). This problem was solved by talking to the Lab TA, looking at the `numpy` documentation, and refering to the very important equation, $y'(t) = \frac{\Delta y(t)}{\Delta t}$. Other than this, I had no real issues.

6 Questions

- See the second paragrah of Reports Section for the answer to this question.
- If the step size were changed for the discrete derivative of Equation (3) in Python, the main difference that would be seen in its plot would be larger magnitude spikes where the rate of change should be instantaneous. This is again due to the fact that there are a discrete amount of values in the output of the python functions.
- This lab was very clear and I had a lot of fun doing it. I have already used python to create plots for other classes and really enjoy it! All tasks, expectations, and deliverables for this lab have been very clear. I do hope I get feedback on how to better organize my reports if the organization of this report is not up to standards.

7 Conclusion

In conclusion, I feel this lab was very successful. No doubt, this was a great introduction to plotting and defining functions in Python. I really enjoyed the discrete derivative that was performed with Python and I think this should be an introductary lab for all future semesters. I have no feedback on possible sources of improvement.

Bibliography