Project Report for ECE 351

Lab 02 - User-Defined Functions

Skyler Corrigan

September 8, 2022

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

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

Contents

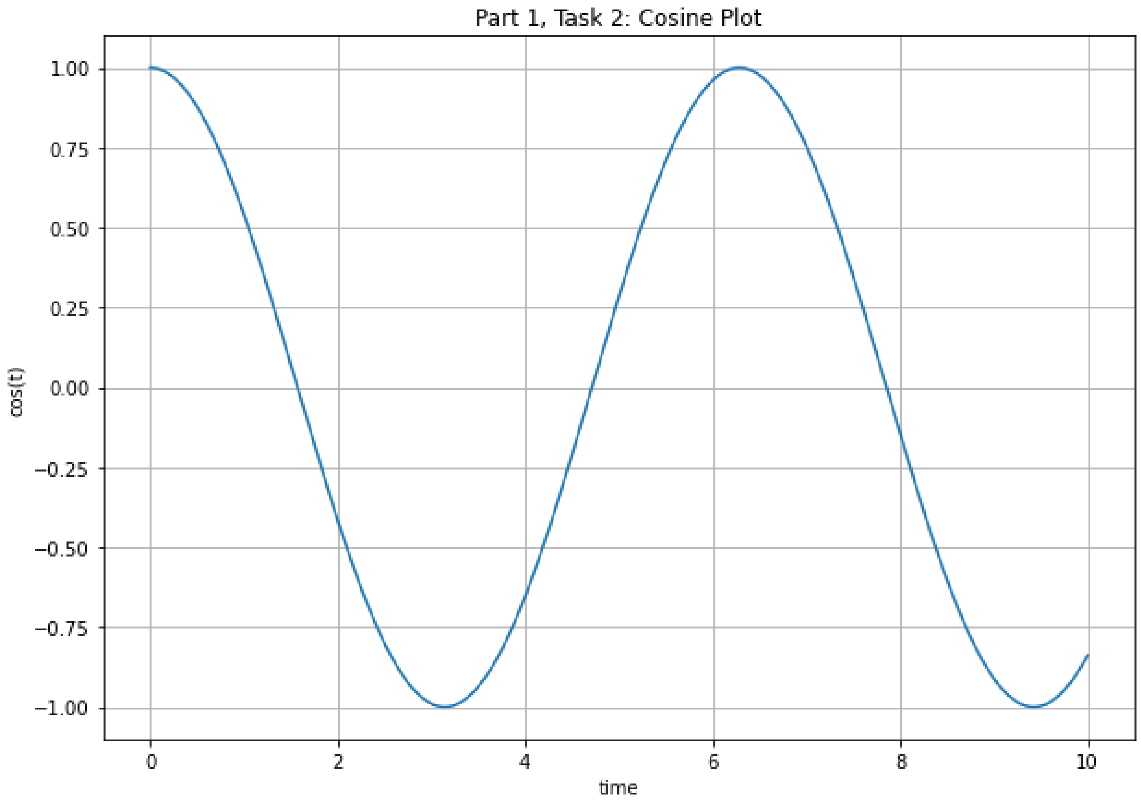
1 Part 1

Part 1 is all about how to make plots. Essentially, the more steps per time unit the better the resolution. In this example, a Cosine graph was plotted with the following code:

Cosine Code

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

#Make a cosine wave using an array t (time)
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 the cosine function
    return y

#Step Size Definition
steps = 1*(10**-5)

#Part 1, Task 2: Cosine Plot------------------------------------------#
#Range of 't'
t = np.arange(0, 10+ steps, steps)

#Function Call
func1 = cosine(t)

#Make and show plot
plt.figure(figsize=(10, 7))
plt.plot(t,func1)
plt.grid()
plt.ylabel('cos(t)')
plt.xlabel('time')
plt.title('Part␣1,␣Task␣2:␣Cosine␣Plot')
```

The following figure is the cosine graph. *Figure 1*



2 Part 2

Part 2 is about user defined functions.
For the given graph, the derived equation is:

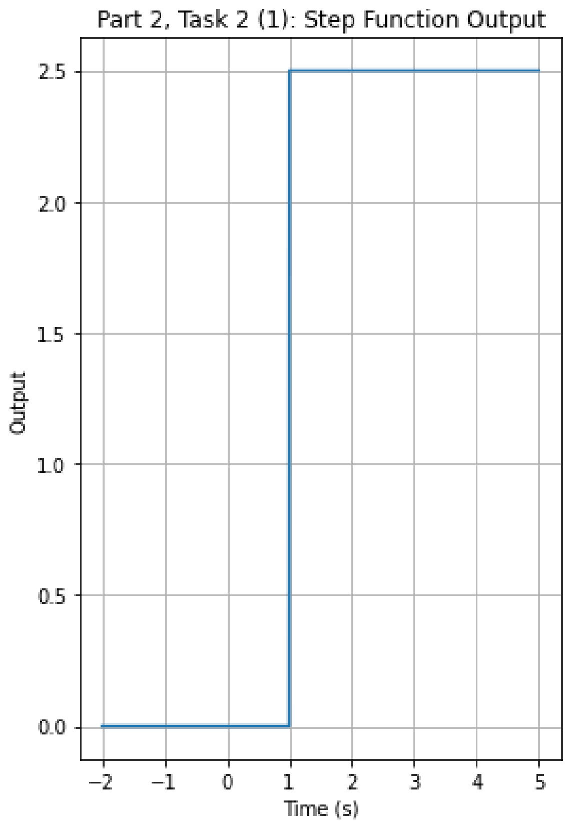
$$f(t) = r(t) - r(t-3) + 5u(t-3) - 2u(t-6) - 2r(t-6)$$

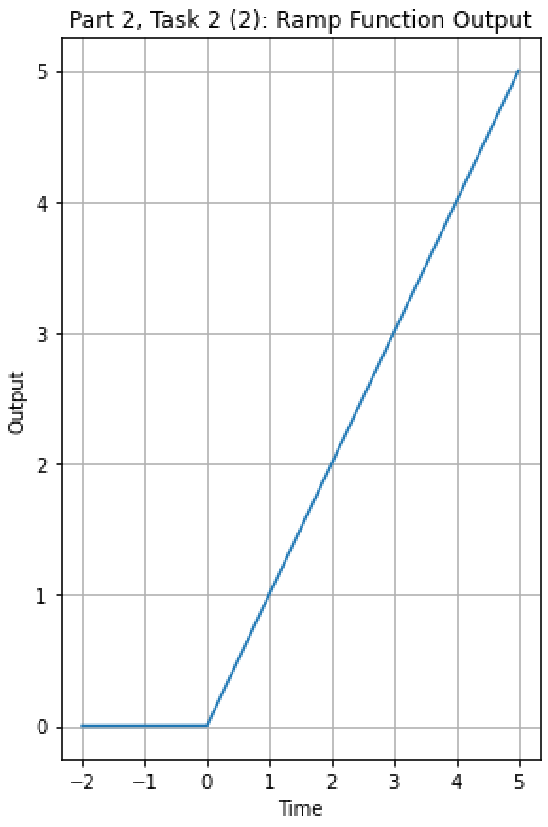
The Step and Ramp Functions were defined with the following code:
User Defined Functions Code

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

#-----------------FUNCTIONS FOR TASKS------------------------------#
#Make a step function using an array t, stepTime, and stepHeight
def Step(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 Ramp(t, startTime, slope):
    y=np.zeros(t.shape)
    for i in range(len(t)):
        if(t[i]>=startTime):
            y[i]=slope * (t[i]-startTime)
    return y

#Step Size Definition
steps = 1*(10**-5)


#Part 2, Task 2------------------------------------------------
#Define a range of t. Start at -2, go to 5 (+a step)
    w/ a stepsize of step
t = np.arange(-2, 5+ steps, steps)

#Step functions
y=Step(t, 1, 2.5)
yNeg2=Step(t, -1, -2)

#Ramp functions
yRamp=Ramp(t, 0, 1)
yNegRamp=Ramp(t, 2, -2.5)

#Make the plot for step functions
plt.figure(figsize=(10, 7))

#Plot step function
plt.subplot(1, 2, 1)
plt.plot(t, y)
#plt.plot(t,yNeg2) #negative step function
plt.title('Part␣2,␣Task␣2␣(1):␣Step␣Function␣Output')
plt.ylabel('Output')
plt.xlabel('Time␣(s)')
plt.grid()

#Plot ramp function
plt.subplot(1,2,2)
plt.plot(t, yRamp)
#plt.plot(t,yNegRamp) #negative ramp function
plt.title('Part␣2,␣Task␣2␣(2):␣Ramp␣Function␣Output')
plt.ylabel('Output')
plt.xlabel('Time')
plt.grid()
```

The graphs for these can be found in the Figure below. *Figure 2*

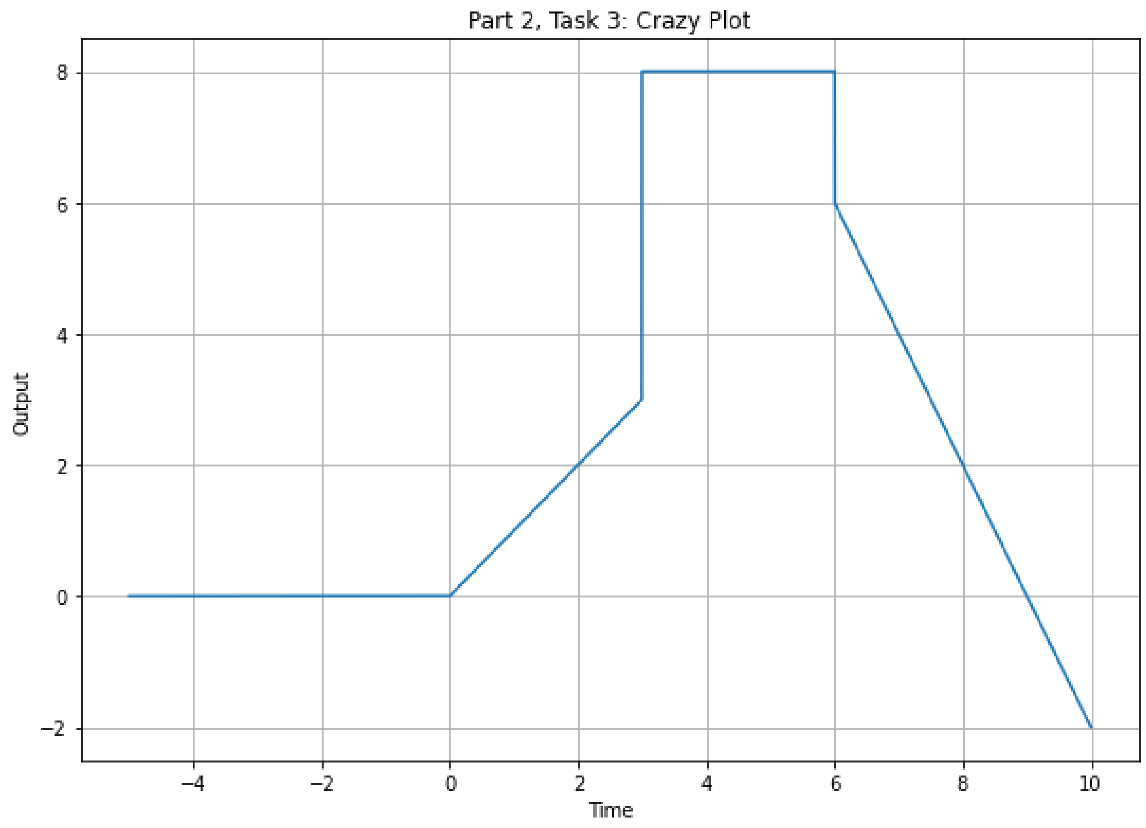




In order to make the graph that was detailed in the equation shown at the beginning of this section, the following code was used:
Crazy Graph Code

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

#---------------------FUNCTIONS FOR TASKS--------------------------------#

#Make a step function using an array t, stepTime, and stepHeight
def Step(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 Ramp(t, startTime, slope):
    y=np.zeros(t.shape)
    for i in range(len(t)):
        if(t[i]>=startTime):
            y[i]=slope * (t[i]-startTime)
    return y

#Step Size Definition
steps = 1*(10**-5)

#Part 2, Task 3-------------------------------------------------
#Define a range of t. Start at -5, go to 10 (+a step)
    w/ a stepsize of step
t = np.arange(-5, 10+ steps, steps)

#Get an array of the function to plot
#Make my function!
ramp1 = Ramp(t, 0, 1)
negRamp = Ramp(t, 3, -1)
fiveStep = Step(t, 3, 5)
negTwoStep = Step(t, 6, -2)
negTwoRamp = Ramp(t, 6, -2)

func2Plot = ramp1 + negRamp + fiveStep + negTwoStep + negTwoRamp
#Ploting the functionToPlot for part 2, task 3


plt.figure(figsize=(10,7))
plt.plot(t, func2Plot)
plt.title('Part 2, Task 3: Crazy Plot')
plt.ylabel('Output')
plt.xlabel('Time')
plt.grid()
```

The graph can be found in the image below.
Figure 3



3 Part 3

This part was all about manipulating the plot made in part 2!
Task 4 required making a derivative plot outside of Python, so that graph was made with drawio.
The code for all the manipulations (which include time shifts, time scalars, and a derivative) can be found here:
Part 3 Code

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

#-----------------FUNCTIONS FOR TASKS-----------------------------#

#Make a step function using an array t, stepTime, and stepHeight
def Step(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 Ramp(t, startTime, slope):
    y=np.zeros(t.shape)
    for i in range(len(t)):
        if(t[i]>=startTime):
            y[i]=slope * (t[i]-startTime)
    return y

#Time reversal using t and a function plot
def timeReversal(ary):

    #Make an array to return time reversal plot
    timeReverse = np.zeros(ary.shape)

    #Goes from index 0 to index len(f)-1
    for i in range(0, len(ary)-1):
        timeReverse[i] = ary[(len(ary) - 1)-i]

    #Return the time reversed array
    return timeReverse

#Time shift of a plot
def timeShift(timePlot, shift):

    timePlot += shift
    return timePlot

#Time scale
def timeScale(t, scale):
    for i in range(0, len(t)-1):
        t[i]=t[i] * scale

    return t
#----------------------------END FUNCTIONS----------------------------#

#Step Size Definition
steps = 1*(10**-5)

#Part 3, Task 1-------------------------------------
#Apply time reversal
reverseTimeFunction = timeReversal(func2Plot)

#Ploting reverseTimeFunction
```

```
plt.figure(figsize=(10, 7))
plt.plot(t, reverseTimeFunction)
plt.ylabel('Output')
plt.xlabel('Time')
plt.title('Part 3, Task 1: Crazy Plot w/Time Reversal')
plt.grid()

#Part3, Task 2--------------------------------------
tScale = timeShift(t,4)

#Ploting f(t-4)
plt.figure(figsize=(10, 7))
plt.subplot(1,2,1)
plt.plot(tScale, func2Plot)
plt.ylabel('Output')
plt.xlabel('Time')
plt.title('Part 3, Task 2 (1): Crazy Plot w/ f(t-4)')
plt.grid()

plt.subplot(1,2,2)
plt.plot(tScale,reverseTimeFunction)
plt.xlabel("Time")
plt.title("Part 3, Task 2 (2): Crazy Plot w/ f(-t-4)")
plt.grid()

#Part 3, Task 3--------------------------------------
#Define a range of t. Start at -5, then go to 10 (+a step)
    w/ a stepsize of step
t = np.arange(-5, 10+ steps, steps)
#Create 1/2 time scale
tScaleHalf = t * (1/2)

#Plotting f(t/2)
plt.figure(figsize=(10, 7))
plt.subplot(1,2,1)
plt.plot(tScaleHalf, func2Plot)
plt.ylabel('Output')
plt.xlabel('Half Time')
plt.title('Part 3, Task 3 (1): Crazy Plot w/ Half Time')
plt.grid()

#time scale of 2t!
tScaleDouble = t * 2

#Plotting f(2t)
```

```
plt.subplot(1,2,2)
plt.plot(tScaleDouble, func2Plot)
plt.ylabel('Output')
plt.xlabel('Double Time')
plt.title('Part 3, Task 2 (2): Crazy Plot w/ Double Time')
plt.grid()

#Part 2, Task 5------------------------------------------------
#Calculate and plot the deritive of func2Plot
func2PlotDir = np.diff(func2Plot)
tMod = np.arange(-5, 10, steps)
plt.figure(figsize=(10, 7))
plt.plot(tMod, func2PlotDir)
plt.ylabel('Output')
plt.xlabel('Time')
plt.title("Part 3, Task 5: Crazy Plot, Now Feturing the Derivitive!")
plt.grid()
```

All graphs can be found in the figures below.

Figure 4

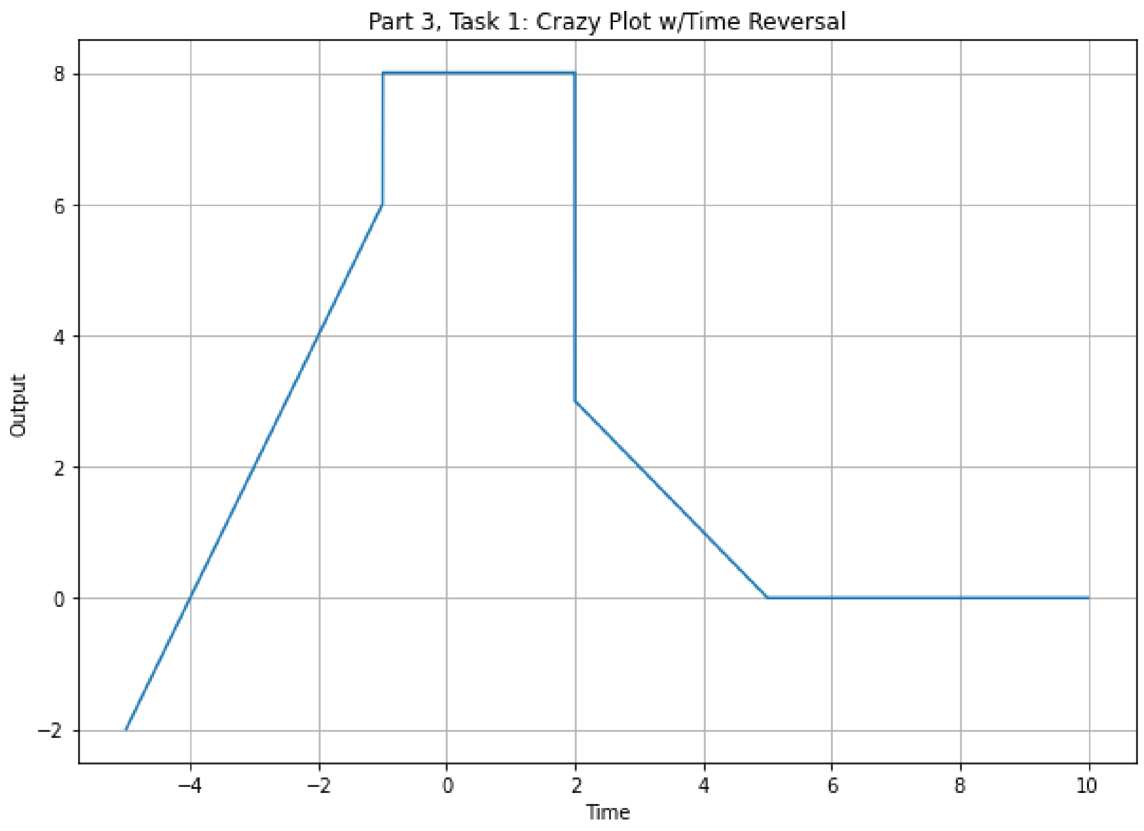


Figure 5

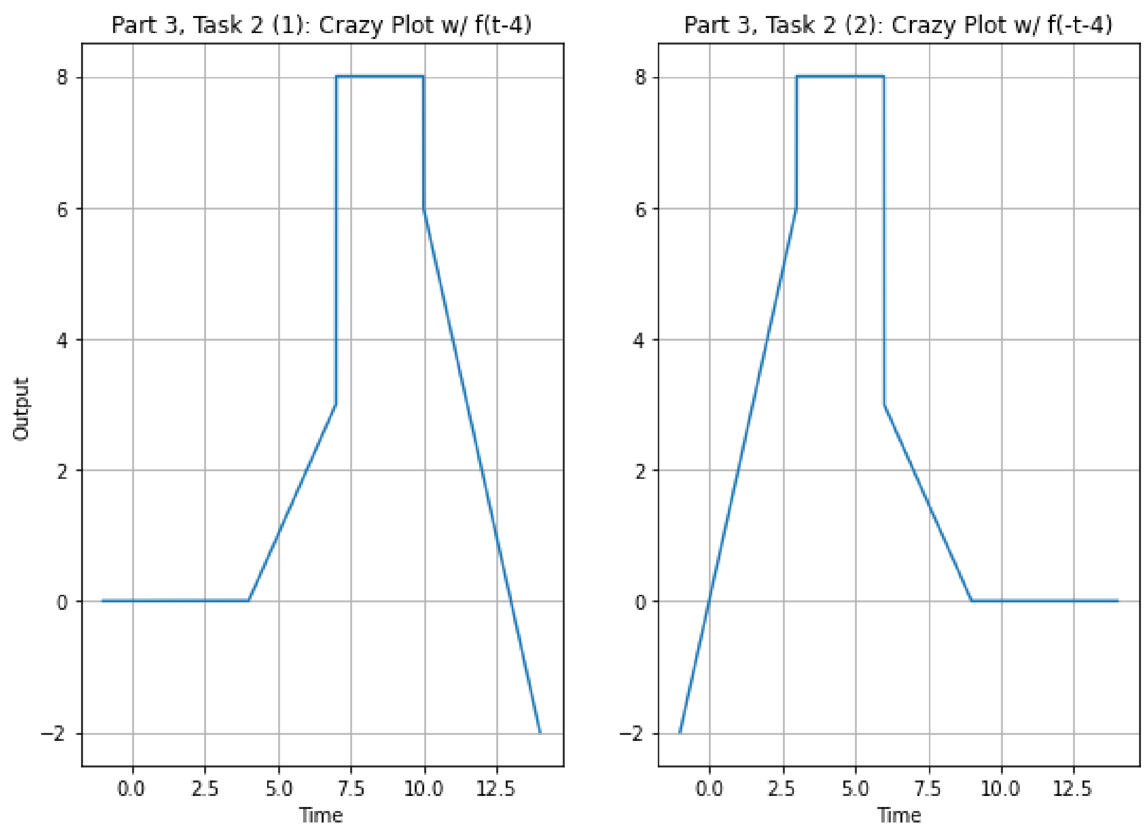


Figure 6

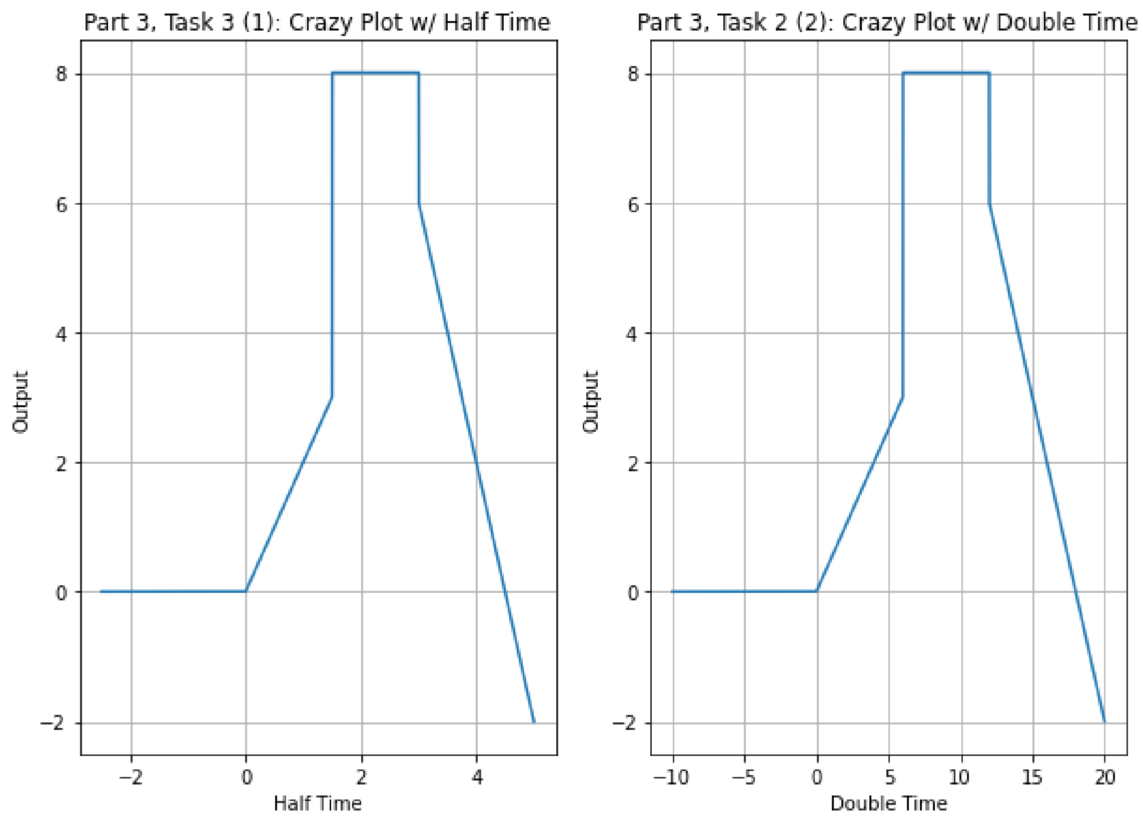


Figure 7

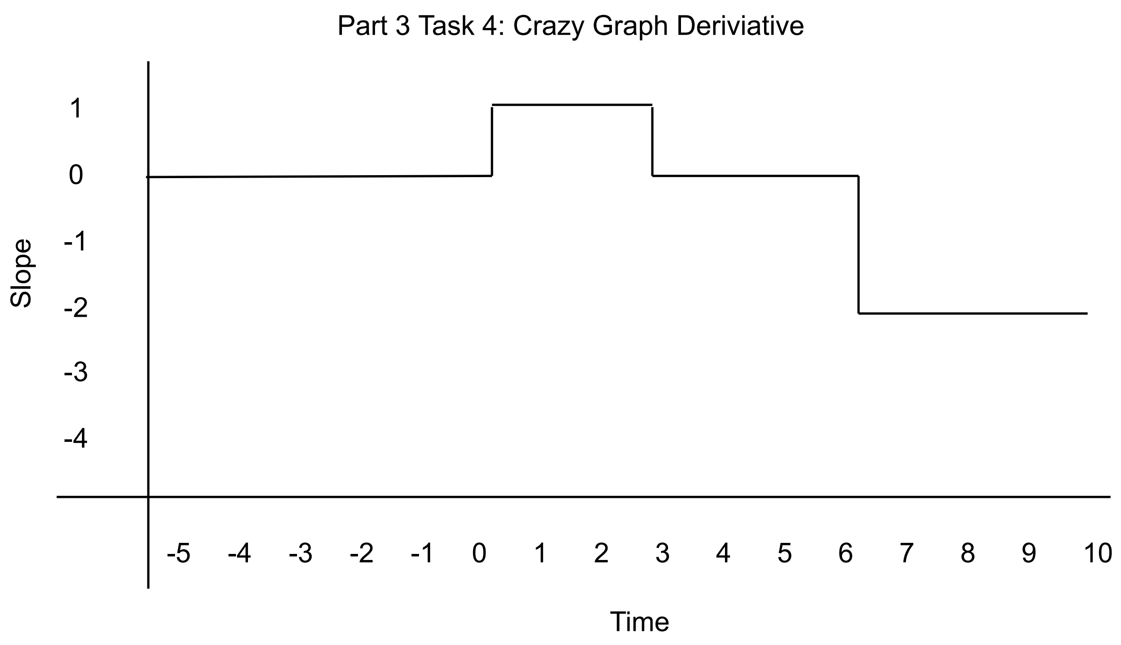
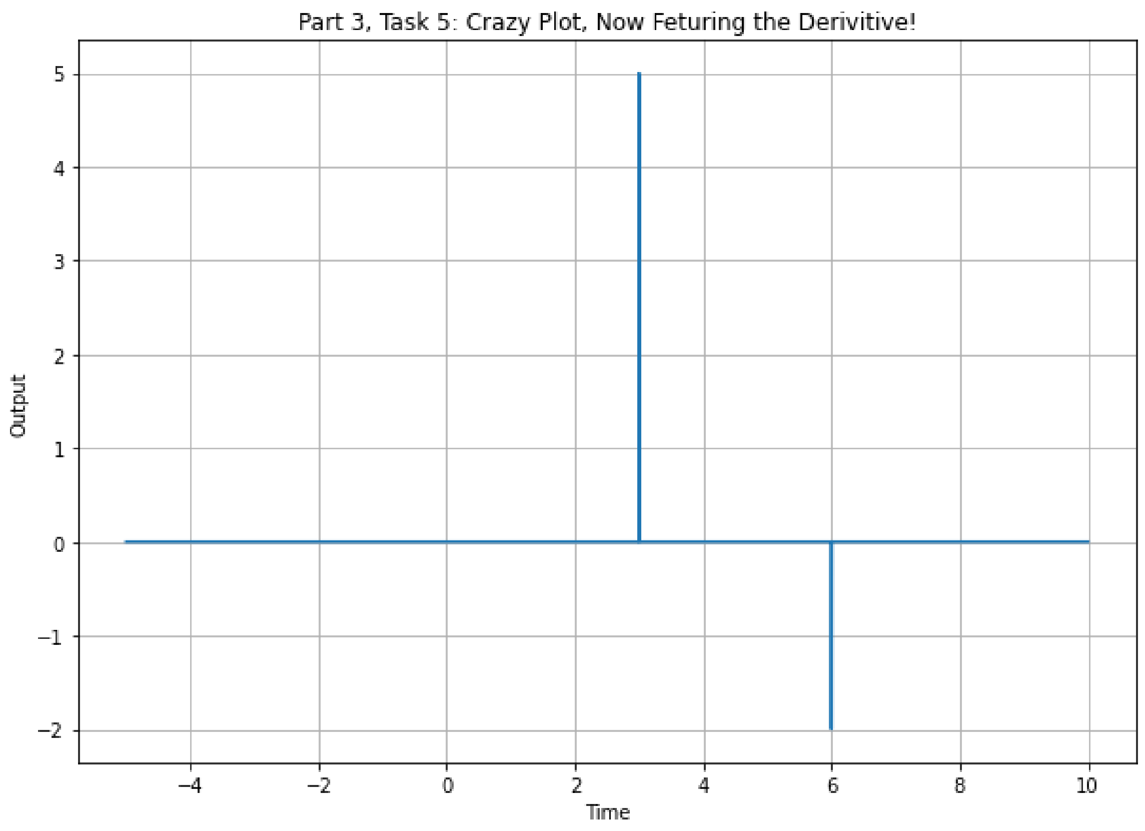

Part 3 Task 4: Crazy Graph Deriviative
Slope
1
0
-1
-2
-3
-4
-5
-4
-3
-2
-1
0
1
2
3
4
5
6
7
8
9
10
Time

Figure 5


Part 3, Task 5: Crazy Plot, Now Feturing the Derivitive!
Output
5
4
3
2
1
0
−1
−2
−4
−2
0
2
4
6
8
10
Time

4 Questions

4.1 Question 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.
They are not identical. It is not possible for them to be identical because Python is dumb.// Python does not take into account the time when the differential equation is used, only taking into account the difference between the two values.
The hand drawn plot takes into account the time as well, allowing for a more accurate graph.

4.2 Question 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.
No. The numpy.diff operator will still only assume that the time value is 1 instead of being infinitely small like the hand drawn graph.

4.3 Question 3:

Leave any feedback on the clarity of lab tasks, expectations, and deliverables.
There was a lot of deliverables in Part 3. Splitting it up into more parts would of made keeping track of everything easier and consequentially would of made keeping track of where I am in the lab easier as well.
There were multiple times where I completely lost my place in Part 3 and just had to go down the list to find where I was again.