ECE 351 - 52

Signals and Systems 1

Lab 2

Submitted By :
Owen Blair

Contents

1 Important Notes

It is important to note that plt.show() is needed at the end of the python code to properly show the plots of each function. It was not included in every section of code that plots a function(s) because it is assumed that its included at the end the code.

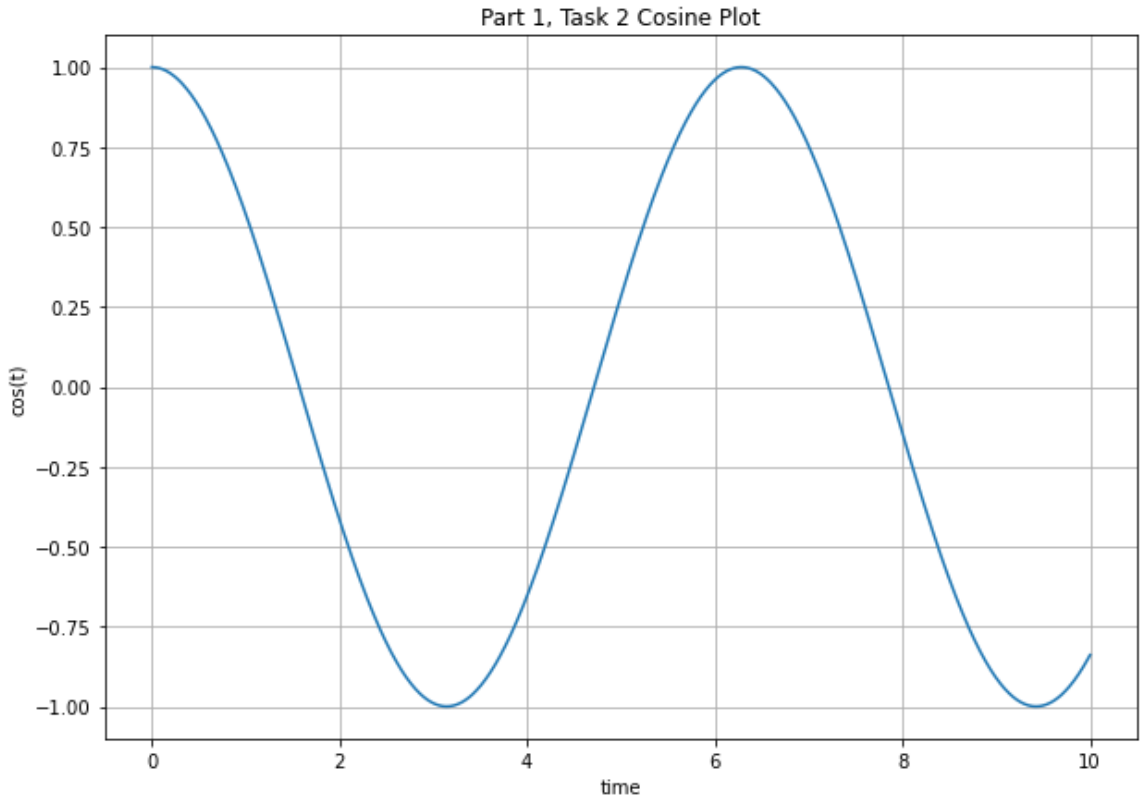
The web address to the GitHub where LaTeX code is stored is here:
https://github.com/Blairis123/ECE351_Reports

The web address to the GitHub where the Python code is here:
https://github.com/Blairis123/ECE351_Code

2 Part 1 Deliverables

Part 1 was all about how to make a good looking plot. For this task a simple cosine plot was to be made using a user defined function that used numpy.cos(). This plot can be seen in Figure 1 and the code used to do so can be seen in Listing 1.

Figure 1: A Cosine Plot



Listing 1: User defined Function & Plotting Code

```
#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

    #Define step size
steps = 1e-2

#Part 1, Task 2-------------------------------------
    #Define a range of t. Start @ 0 go to 10 (+a step) w/ a
   stepsize of step
t = np.arange(0, 10+ steps, steps)

    #Call function
func1 = cosine(t)

```

```
    #Make plot and then show it
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')
```

3 Part 2 Deliverables

Part 2 is where the step and ramp functions are made and are used to create the function shown in Figure 2: Function to Plot of the lab 2 handout. The derived equation for the Function to Plot is as follows:

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

The user defined step function's code is shown in Listing 2 and the ramp function's code is shown in Listing 3. The stepFunc() takes three inputs. The inputs are t, the time array of the function, startTime, the time the step takes place, and stepHeight, the amount the step function moves up or down. The rampFunc() function takes three inputs as well. The inputs are t, the time array of the function, startTime, the time the ramp starts, and slope, the slope of the ramp function. The plot for the step and ramp user defined functions is in Figure 2. Both functions return an array the size of t with values that correspond to the values in t. The code used to create and plot the function from the lab 2 handout can be seen in Listing 4 and a screen capture of this plot can be seen in Figure 3.

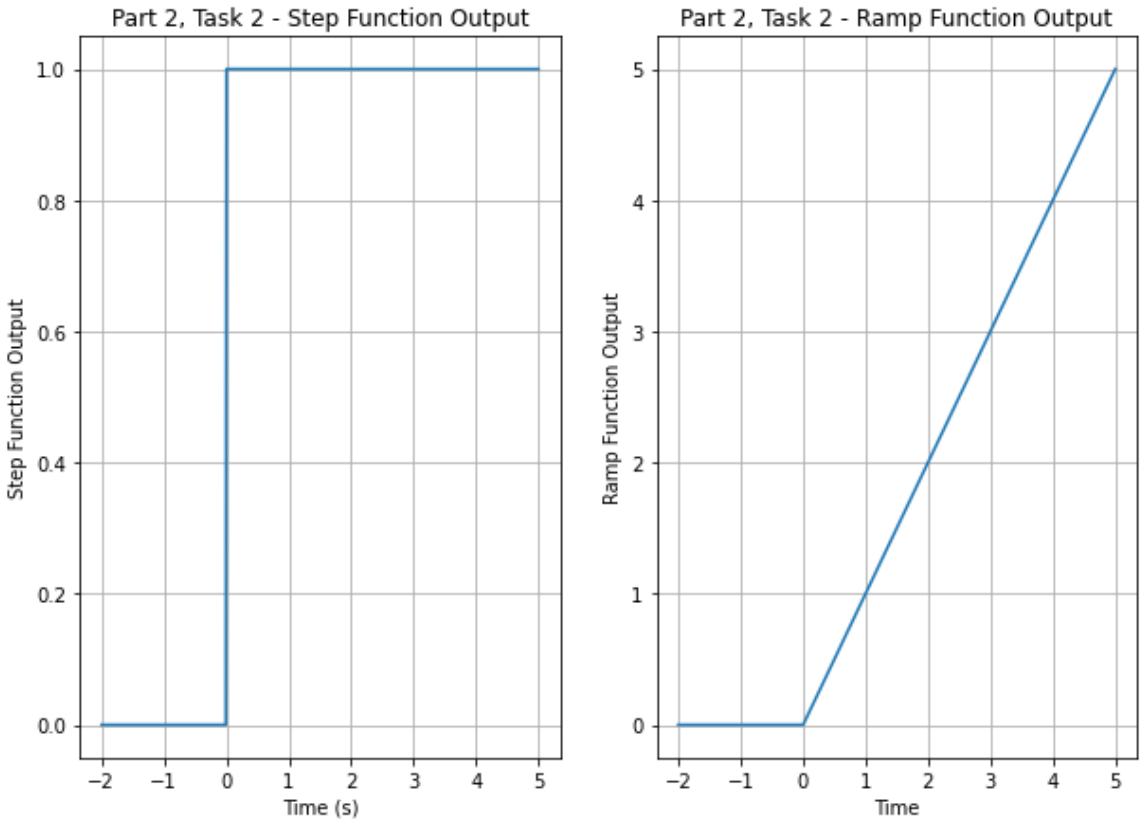
Listing 2

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

Listing 3

```
#Make a ramp function using an array t, startTime, and slope
def rampFunc(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
```

Figure 2: User Defined Step and Ramp Functions

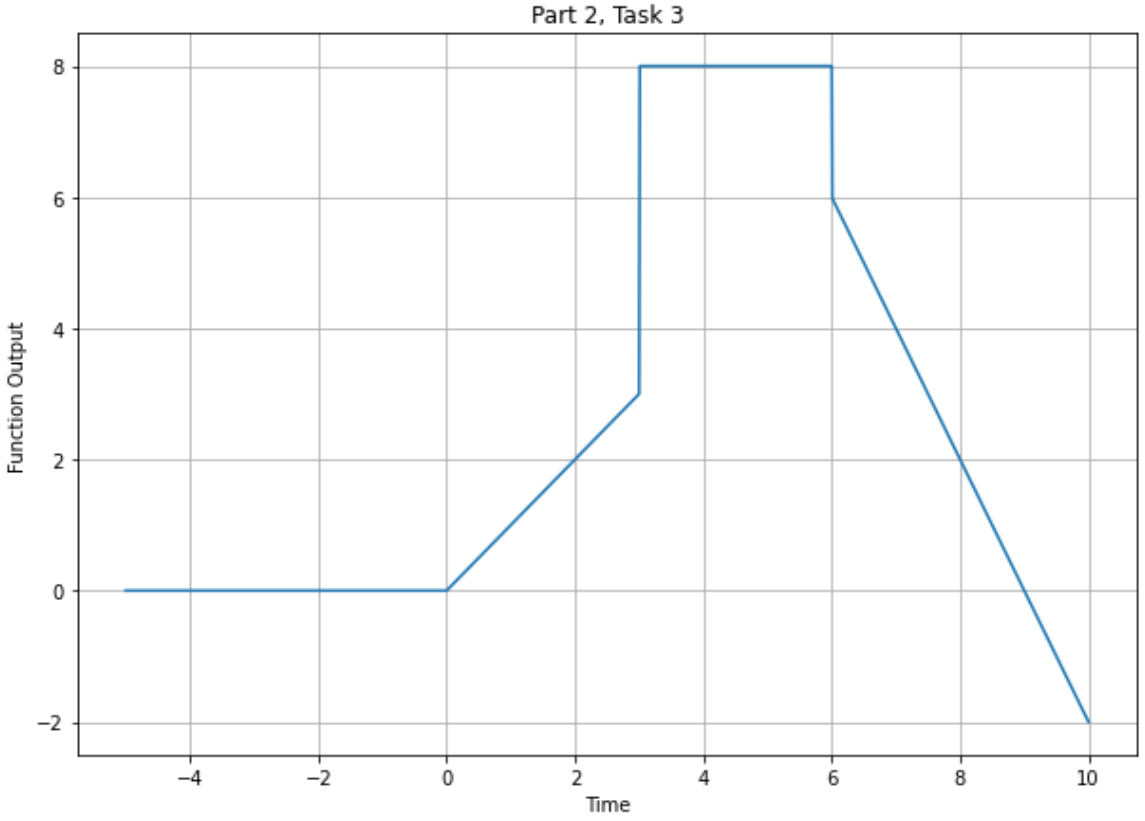


Listing 4: Function to Plot Code

```
#Part 2, Task 3-----------------------------------
    #Define a range of t. Start @ -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 = rampFunc(t, 0, 1)
negRamp = rampFunc(t, 3, -1)
fiveStep = stepFunc(t, 3, 5)
negTwoStep = stepFunc(t, 6, -2)
negTwoRamp = rampFunc(t, 6, -2)

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


plt.figure(figsize=(10,7))
```

```
18 plt.plot(t, func2Plot)
19 plt.title('Part 2, Task 3')
20 plt.ylabel('Function Output')
21 plt.xlabel('Time')
22 plt.grid()
```

Figure 3: User Made Function to Plot



4 Part 3 Deliverables

Part 3 consisted of creating time-shifting and time reversing user defined functions and plotting them. To time shift a function a timeShift() function was made to shift a time array that is associated with a function array. the timeShift() function takes two inputs, a time array to be shifted, t, and a shift amount, shift and returns the shifted array. This can be found in Listing 5. A similar function was made for a time reversal and the code can be found in Listing 6. The timeReversal() function takes one input, an array to be reversed, and returns a reversed array. The time reversal

function Function To Plot can be seen in Figure 5. The functions of $f(t-4)$ and $f(-t-4)$ can be seen in Figure 6. These plots were made by taking the previously defined Function To Plot and applying first the timeShift(t,4) function to the associated time values, this is $f(t-4)$, and then the timeReversal(func2Plot) function to make the time reversed function with a time shift, this is $f(-t-4)$

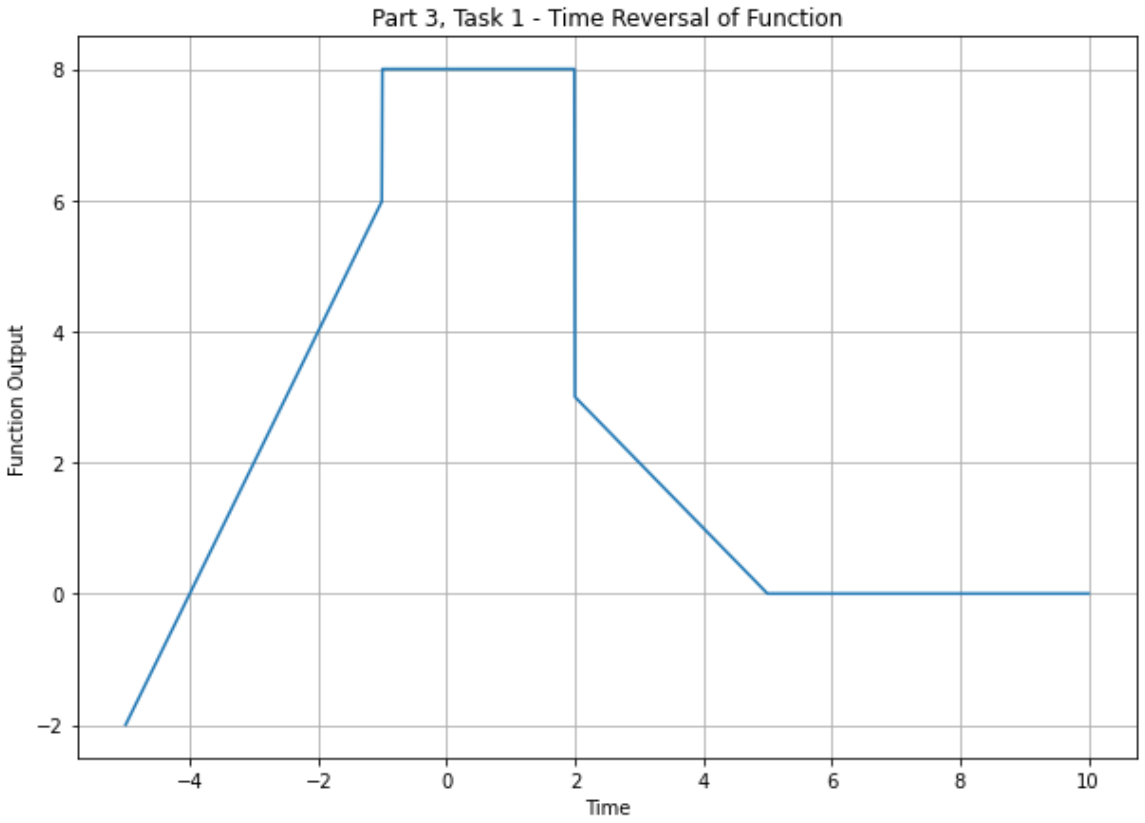
Listing 5: timeShift() User Defined Function

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

    timePlot += shift
    return timePlot
```

Listing 6: timeReversal() User Defined Function

```
#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
```

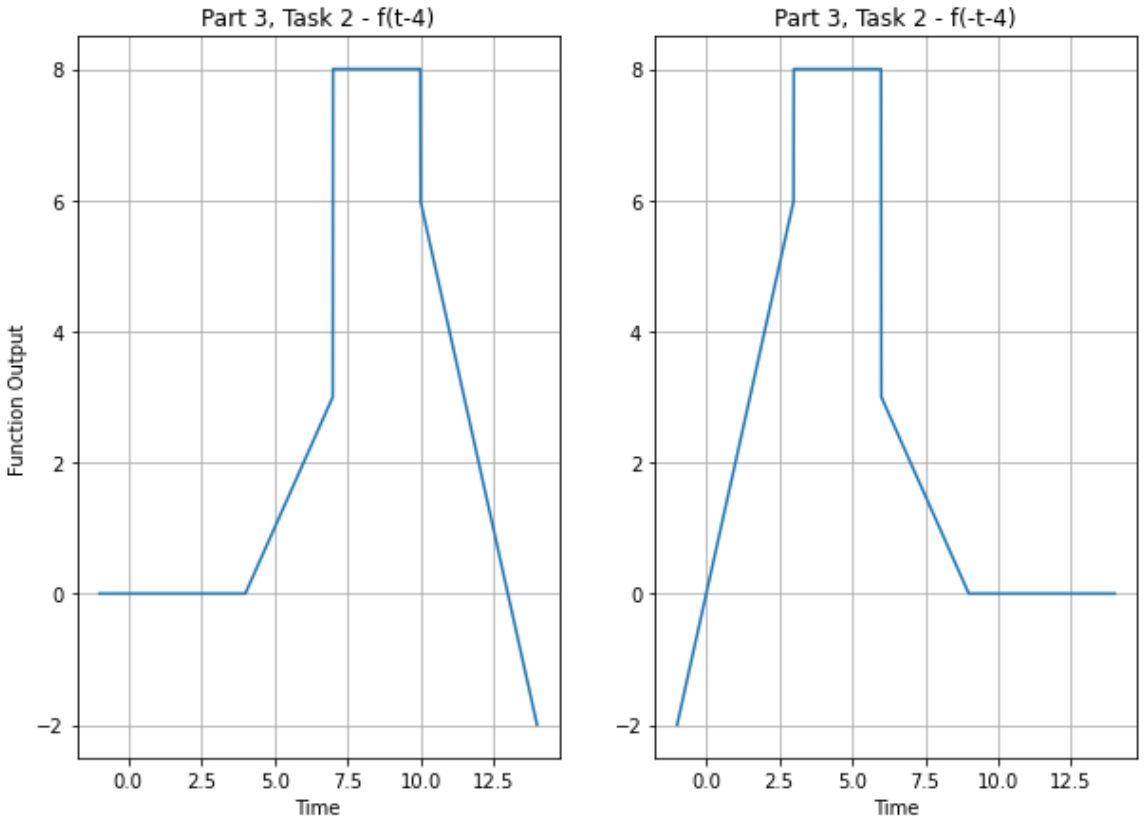
Figure 5: Time reversal of func2Plot



Listing 7: Code to Plot func2plot Time Reversal

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

    #Ploting reverseTimeFunction
plt.figure(figsize=(10, 7))
plt.plot(t, reverseTimeFunction)
plt.ylabel('Function Output')
plt.xlabel('Time')
plt.title('Part 3, Task 1 - Time Reversal of Function')
plt.grid()
```

Figure 6: f(t-4) and f(-t-4) Plots



Listing 8: Code to Plot f(t-4) and f(-t-4)

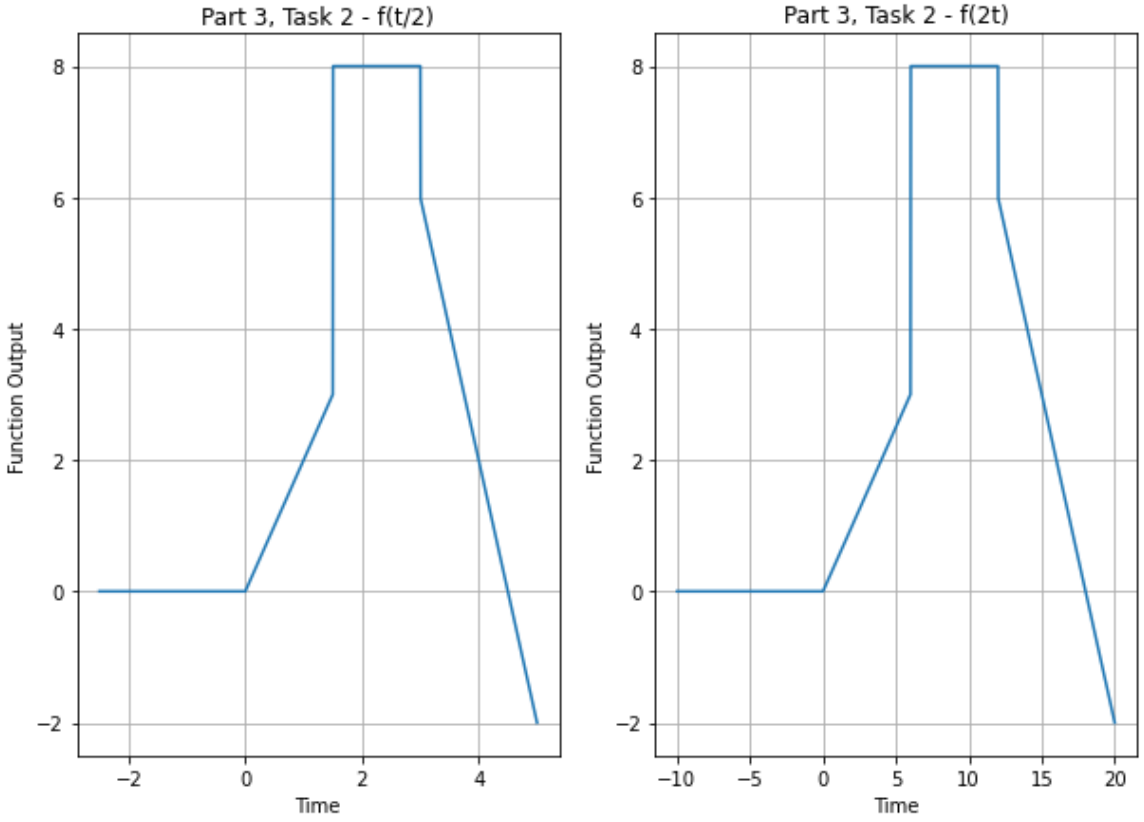
```
#Part3, Task 2-------------------------------------
#tScale = np.arange(-5 + 4, 10 + steps + 4, steps)
tScale = timeShift(t,4)

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

plt.subplot(1,2,2)
plt.plot(tScale,reverseTimeFunction)
plt.xlabel("Time")
plt.title("Part 3, Task 2 - f(-t-4)")
plt.grid()
```

The second part of this section was applying time scales to the Function To Plot. This was done by manipulating the associated time array with each function to be plotted. The user defined function created to do this is called timeScale(). This function takes two inputs, an array, t, and a scaling factor, scale and returns a scaled time array. The code for this can be seen in Listing 8. Figure 7 contains the plots of f(t/2) and f(2t) that were created using the timeScale() function.

Listing 9: timeScale() User Defined Function

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

Figure 7: Plots of f(t/2) and f(2t)



Listing 10: Code to plot f(t/2) and f2t)

```
#Part3, Task 2-------------------------------------
#tScale = np.arange(-5 + 4, 10 + steps + 4, steps)
tScale = timeShift(t,4)

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

plt.subplot(1,2,2)
plt.plot(tScale,reverseTimeFunction)
plt.xlabel("Time")
plt.title("Part 3, Task 2 - f(-t-4)")
plt.grid()
```

Within this section the derivative of the Function To Plot was taken using the numpy.diff() library defined function and compared to a hand drawn derivative function that was expected. The lab handout says that the hand drawn derivative function will look different than the one created by the numpy.diff() function. The hand drawn derivative function can be seen in Figure 8 and the numpy.diff() function can be found in Figure 9. Each plot is followed by a listing that contains the code used to create each plot.

Figure 8: Hand Drawn Function To Plot Derivative Plot

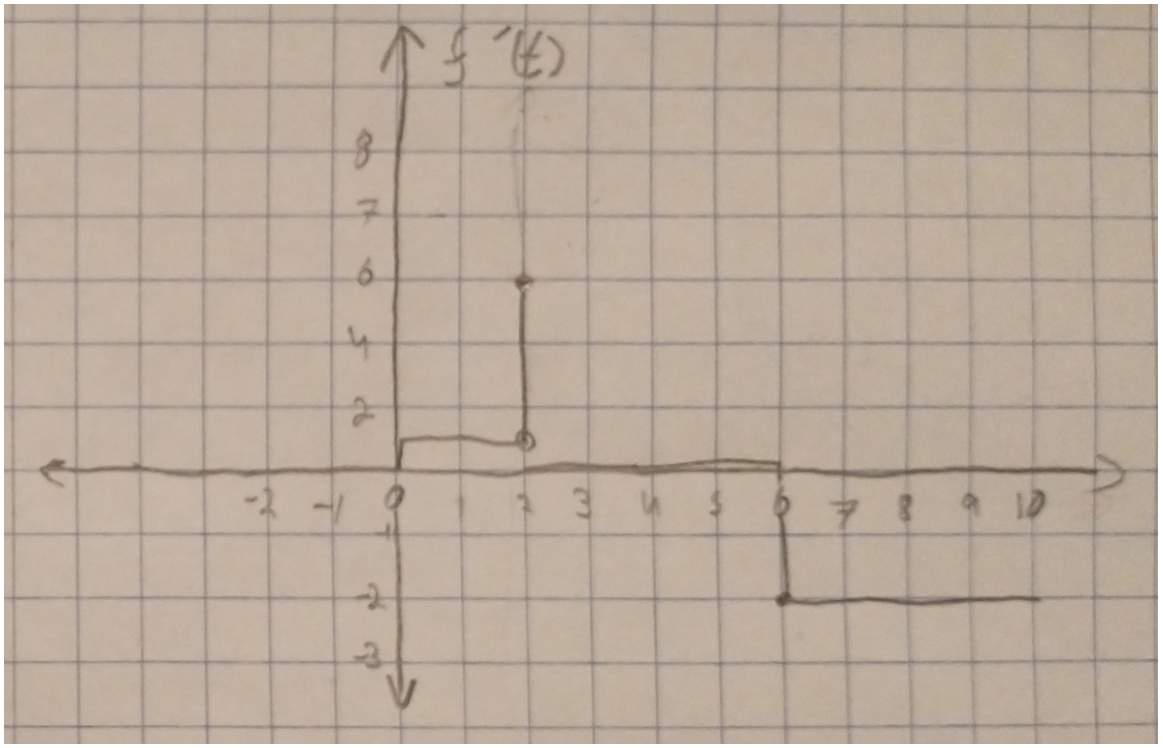
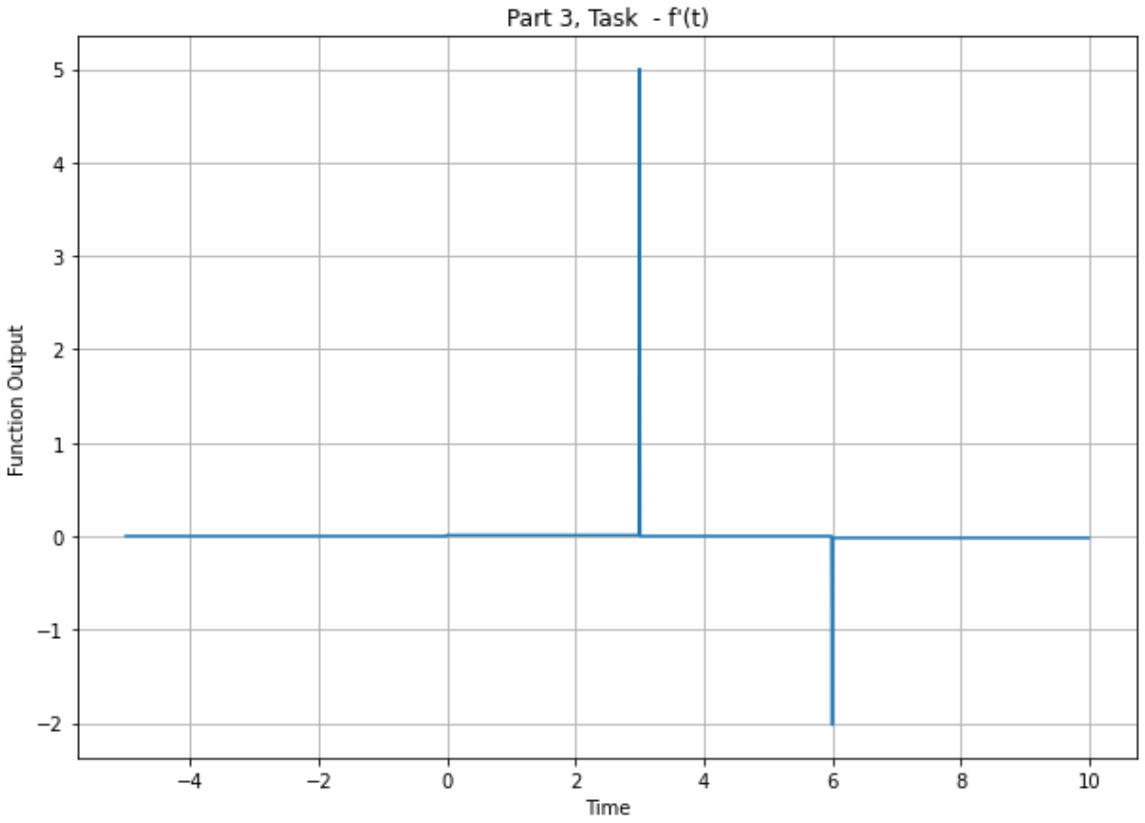


Figure 9: numpy.diff() Generated Function Plot



Listing 11: Code to Plot numpy.diff() Generated Derivative

```
#Calculate and plot the diritive of func2Plot
func2PlotDir = np.diff(func2Plot)
tMod = np.arange(-5, 10, steps)
plt.figure(figsize=(10, 7))
plt.plot(tMod, func2PlotDir)
plt.ylabel('Function Output')
plt.xlabel('Time')
plt.title("Part 3, Task  - f'(t)")
plt.grid()
```

5 Questions

5.1 Question 1

The plot of the Function To Plot derivative drawn by hand and the one generated by numpy.diff() are not identical. This is because the numpy.diff() function takes the difference between each array value and does not account for the time between the

two points. In short the numpy.diff() function assumes the time between each index value is 1 where the hand drawn derivative takes the time difference into account. It is possible or the two methods of derivative generation to make identical plots and that is by making the resolution of the Function To Plot equal to 1.

5.2 Question 2

The correlation between the plots of the derivative by hand and the one generated by the numpy.diff() function is that the closer to 1 the step size of func2plot gets the closer the plot of the derivatives will be. This is because the derivative made by hand can account for an infinitely small time step where numpy.diff() assumes that the time difference between each array value is 1. In essence the hand drawn derivative assumes

$$df(t) = \frac{f(t + \Delta t) - f(t)}{\Delta t}$$

where numpy.diff() assumes

$$df(t) = \frac{f(t_{i+1}) - f(t_i)}{1}$$

5.3 Question 3

The lab was fairly clear. The only thing I found cumbersome was that part 3 had a lot of deliverables and I think it would be better if the time scale operations and derivatives were their own sections.