Project Report for ECE 351

Lab 2: User Defined Functions

Isaias Munoz

September 6, 2022

University of Idaho

Contents

1 Introduction

The purpose of this lab is to introduce user defined functions and then utilize these functions to time shift, scale , reverse and discrete differentiation. This lab uses Python to accomplish it. One of the main key points in this lab is to use numpy and not lists. I first began with inserting the necessary packages or a lot actually.

2 Methodology

2.1 Part 1

The first task that was assigned was to write a user defined function in order to plot:

$$y(t) = cos(t)$$

Following code that was provided as an example of user define functions and using numpy.cos() I created the following user define function for $y(t) = cos(t)$. making sure the axis were all visible and within range Figure 1 below shoes the code. It is important to notice that I did not need the if statement and could have achieved it by just writing a for loop without the "if". The plot is under results as Fig 7.

```
steps = .2e-2 # Define step size
#creating and plotting cosine(t)
################################################################################
t = np . arange (0 , 10 + steps , steps )

def func1 ( 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
         if i < ( len( t ) + 1) : #works both ways with an if or withouth an if
            y [ i ] = np . cos ( t [ i ])
    return y # send back the output stored in an array


y = func1 ( t ) # call the function we just created


#plotting the function I just created
plt . figure ( figsize = (10 , 7) )
plt . plot (t , y )
plt . grid ()
plt . ylabel ('cos(t)' )
plt . xlabel ('time')
plt . title ('Plotting cosine task 1')
################################################################################
```

Figure 1: Cosine user define function

2.2 Part 2

For part 2 the task was to create user define functions for a figure that was given and to plot the results. The derivation of the equation was pretty tough to understand but I finally manage to understand how to make equations from a graph using step and ramp functions.

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

The first equation created was a step and ramp function shown in Figure 2.

```
#creating a stepfunction and ramp function and using it to plot fig2
##########################################################################
# >>> np.arange(start=1, stop=10, step=3)
# array([1, 4, 7])

t = np . arange (-5 , 10 + steps , steps )
# q = np . arange (-1 , 14 + steps , steps )

def stepfunc ( t ) :  #creating step function
    y = np.zeros ( t.shape  ) #initializing y as all zeroes
    for i in range (len ( t ) ): #startingforloop
        if  t [ i ]<0:
            y[i]=0
        else:
            y[i]=1
    return y

def rampfunc ( 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

tryone=stepfunc(t)
try2=rampfunc(t)

plt . figure ( figsize = (10 , 7) )
plt . plot (t , tryone )
plt . grid ()
plt . ylabel ('Y output')
plt . xlabel ('t')
plt . title ('plotting step func to make sure it works')

plt . figure ( figsize = (10 , 7) )
plt . plot (t , try2 )
plt . grid ()
plt . ylabel ('Y output')
plt . xlabel ('t')
plt . title ('plotting ramp func to make sure it works')

fig2=rampfunc(t)-rampfunc(t-3)+5*stepfunc(t-3)-2*stepfunc(t-6)-2*rampfunc(t-6)


plt . figure ( figsize = (10 , 7) )
plt . plot (t , fig2 )
plt . grid ()
plt . ylabel ('Y output')
plt . xlabel ('t')
plt . title ('plotting FIgure 2 using step and ramp functions')

##########################################################################
```

Figure 2: Step and Ramp user functions Code

I began with initializing a range of my x-axis (t) and that would be from (-5,10) incremented specific steps I declared earlier. I defined the function "stepfunc" and passed it the range of my x-axis. I began a loop for the length of my x-axis range and if the value of the position in which the range landed on was less then zero then I would make the $y[i]$ equal to zero, if it wasn't then I would make that position in my y array as one. My 'y' array was initialized to zero earlier and is my 'output'. This gives me a step function. Figure 8 shows the plot that it works as a step function under the results section. I did something very similar with the ramp function, except that if $t[i]$ is not less then the value zero then I would make $y[i] = t[i]$ meaning I would make them equal with a slope of 1 which is the definition of a natural ramp function. Figure 9 shows the ramp function working. Finally it was time to put these functions to create the figure given and match it. After the functions were created then it was simply to write the function that was made from steps and ramp functions with appropriate shifts. Figure 10 can be found under the results section it shows the identical match of the figure provided using step and ramp functions.

2.3 Part 3

In this task the function created was to be used for time-shifting and scaling operations. Figure 3 shows the time reversal code.

```
#################################################################################
#Apply a time reversal and plot the result.
def timereversalfunc(temp): #Much easier when making a function!!!!!!! took forever

    timereverse=np.zeros(temp.shape)

    for i in range (0,len(temp)-1):

        timereverse[i]=temp[(len(temp)-1)-i] #imputting values into array reversed

    return timereverse

reversalz=timereversalfunc(fig2)  #calling timereversal and passing it original fig

plt . figure ( figsize = (10 , 7) )
plt . plot (t , reversalz)
plt . grid ()
plt . ylabel ('Y output' )
plt . xlabel ('time')
plt . title ('Plotting Fig 2 with timereversal')



#################################################################################
```

Figure 3: Time reversal function Code

The time reversal took me a while and made me realize making functions is much easier then what I originally was thinking. I created 'timereversalfunc' which took in a value 'temp'. I initialized 'timereverse' to zero and began a loop to from 0 to the length of temp which in this case $temp = fig2$ or the function that was used to make the figure in part 2. I then proceeded to insert all the values into 'timereverse' but increments down or the opposite way to fill it in backwards. I return and called the function and the plot is in Figure 11 under the results sections.

The next sub-task was to time shift the figure created $f(t-4)$ and $f(-t-4)$. Below in Figure 4 is the code.

```
##############################################################################
#  apply f(t-4) and f(-t-4)
#this is a long way of shifitiing the t axis
#Apply time-shift operations f (t -4) and f (-t -4) and plot the results.
# steps = .2e-2 # Define step size
# q = np . arange (-1 , 14 + steps , steps ) #this also works by just chaning
# your range of numbers because your chaing your t shifting no need
# to write a functuon well maybe, not quiet a function is better tbh
#other way doesnt work because steeps to 14 way to long

def functminus4(temp2,shifter):

        temp2= temp2 +shifter#shifting your time scale instaed of the actual fi
     # temp2.rotate(4)
        return temp2

tminus4=functminus4(t,4)
# fig2shift=rampfunc(q)-rampfunc(q-3)+5*stepfunc(q-3)-2*stepfunc(q-6)-2*rampfun

plt . figure ( figsize = (10 , 7) )
plt . plot (tminus4 , fig2)
plt . grid ()
plt . ylabel ('Y output' )
plt . xlabel ('time shifted')
plt . title ('Plotting Fig 2 with f(t-4)')



# negtminusneg4=functminus4(-t,-4)#doesnt work cause your reverse both

plt . figure ( figsize = (10 , 7) )
plt . plot (tminus4 , reversalz)
plt . grid ()
plt . ylabel ('Y output' )
plt . xlabel ('time shifted')
plt . title ('Plotting Fig 2 with f(-t-4)')

# plt . figure ( figsize = (10 , 7) )
# plt . plot (q, fig2shift)
# plt . grid ()
# plt . ylabel ('Y output' )
# plt . xlabel ('time')
# plt . title ('Plotting Fig 2 with timereversal')


##############################################################################
```

Figure 4: Time Shifting function Code

In the time-shift code in Figure 4 I define 'functminus4' and passed it two values; 'temp2' and 'shifter'. One would hold the range shifted and the other was how many moves shifted. I then called 'functminus4' and passed t and 4 't' would be my range since all I am doing is shifting my range declared earlier. I then plot that with the original figure and obtain the plot which can be found under the results section under Figure 12. The other sub-task was $f(-t-4)$ and to plot the results. Figure 4 has the code for this shift as well. I used the reversed shift used in the previous task and the reversed figure and plotted both. No function needed to be created and this actually worked. The result is in Figure 13.

The other sub-task was to to apply a timescale operation. $f(t/2)$ and $f(2t)$ and plot the results. I felt like these were the easiest but took a while to think about and once figured out they were simple.

```
###############################################################################

# Apply time scale operations f (t/2) and f (2t) and plot the results.

# def scalefunc(temp3,scale):
#     # temp3/=scale
#     # return temp3

#     for i in range(0,len(temp3)-1):
#         temp3[i]=temp3[i]*scale

#     return temp3
t = np . arange (-5 , 10 + steps , steps )
#much simpler thank making a function since you can divide the timescale and
#then simply graph your original function
tdivby2=t/2
ttimes2=t*2

# funcscale=scalefunc(t,2)


plt . figure ( figsize = (10 , 7) )
plt . plot (tdivby2 , fig2)
plt . grid ()
plt . ylabel ('Y output' )
plt . xlabel ('time divided by 2')
plt . title ('Plotting Fig 2 with f(t/2)')


plt . figure ( figsize = (10 , 7) )
plt . plot (ttimes2 , fig2)
plt . grid ()
plt . ylabel ('Y output' )
plt . xlabel ('time multiplied by 2')
plt . title ('Plotting Fig 2 with f(2t)')

###############################################################################
```
1

Figure 5: Time Shifting function Code

I was thinking of making a function but realized if I divided my x-range time-frame I could achieve the same thing. That is what I did I divided my time-frame by 2 and plotted it with the original function 'fig2'. The same scenario happen for multiplication.

I multiplied my time scale by 2. It is important to note that I rarely am making changes to the original timescale. I am making it equal to another value and that way I can mess with it and not change the original. Like in this task 'tdivby2' and 'ttimes2' are the scale changes and its those I am plotting with my original function 'fig2'. The plot results are in Figure 14 and 15 for both shifts respectively.

The last task was to differentiate the figure and plot it by hand and use python $numpy.diff()$ to differentiate it. Figure 6 shows the code and the plot can be found under the results section Figure 16.

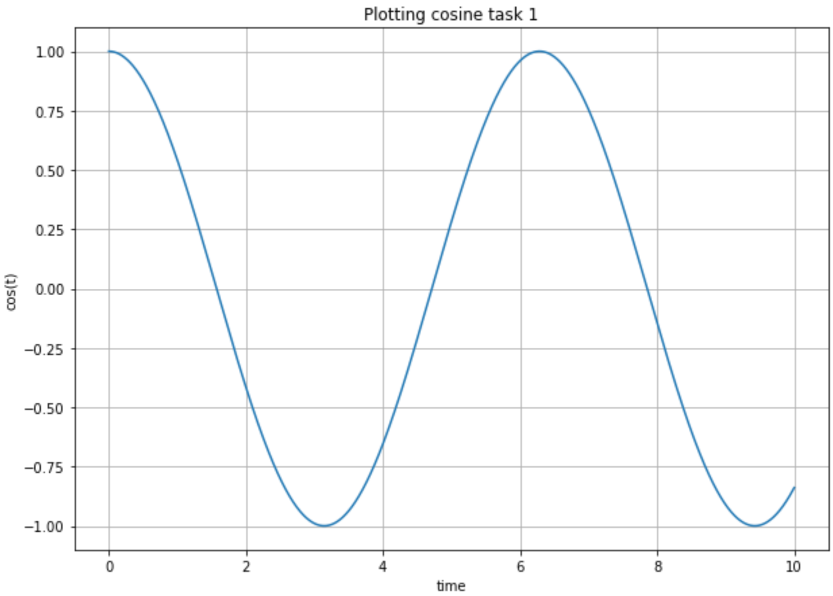
```
#############################################################################



#Calculate and plot the diritive of func2Plot
derivfig2 = np.diff(fig2)
xsize = np.arange(-5, 10, steps)
plt.figure(figsize=(10, 7))
plt.plot(xsize, derivfig2)
plt.ylabel('Y Output')
plt.xlabel('Time')
plt.title("Derivative of fig2")
plt.grid()




############################################################################


```

Figure 6: Time Shifting function Code

I called the differentiate function and was easier then what I thought, since I thought I needed a special package included in python. After calling the $np.diff()$ I proceeded to plot it which used the same structure as my previous plots.

3 Results

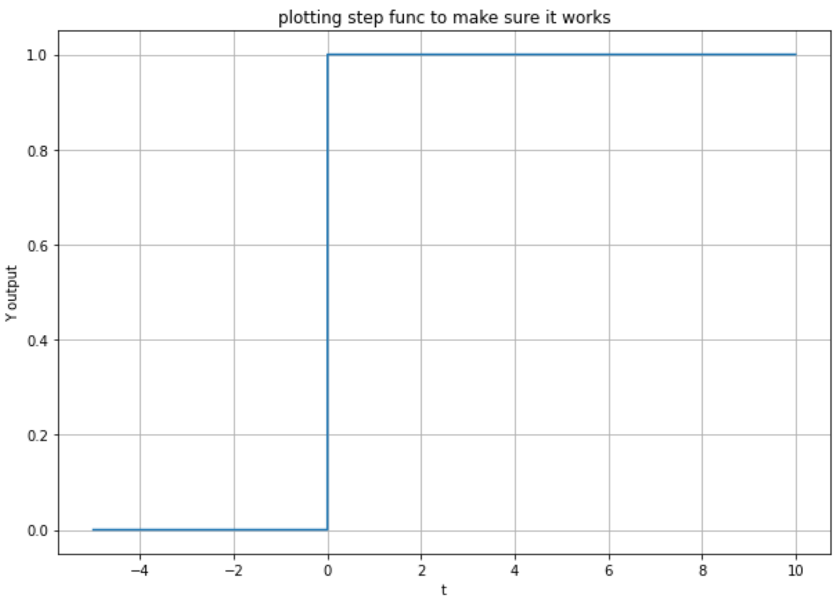
3.1 Part 1



1

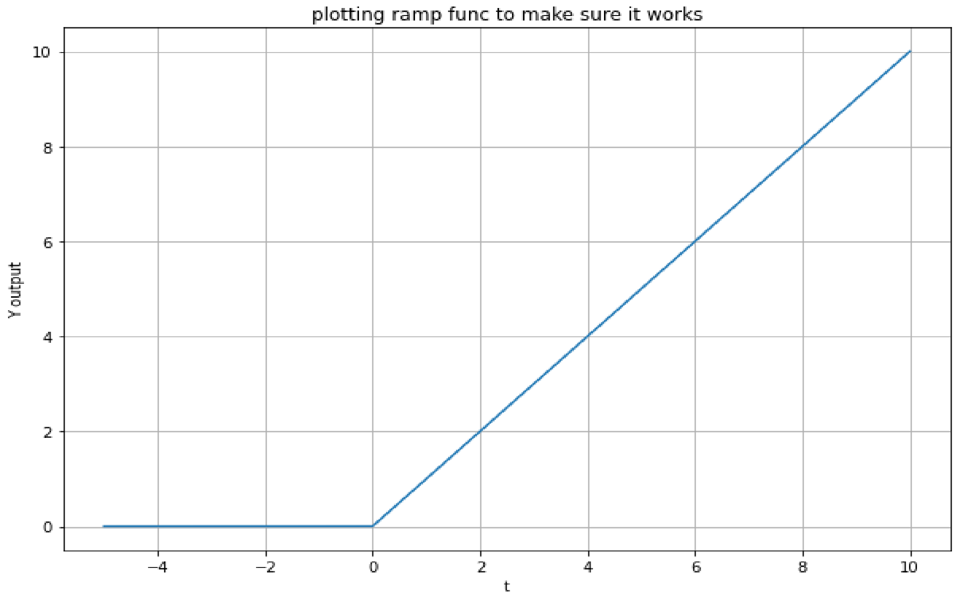
Figure 7: Cosine(x)

3.2 Part 2



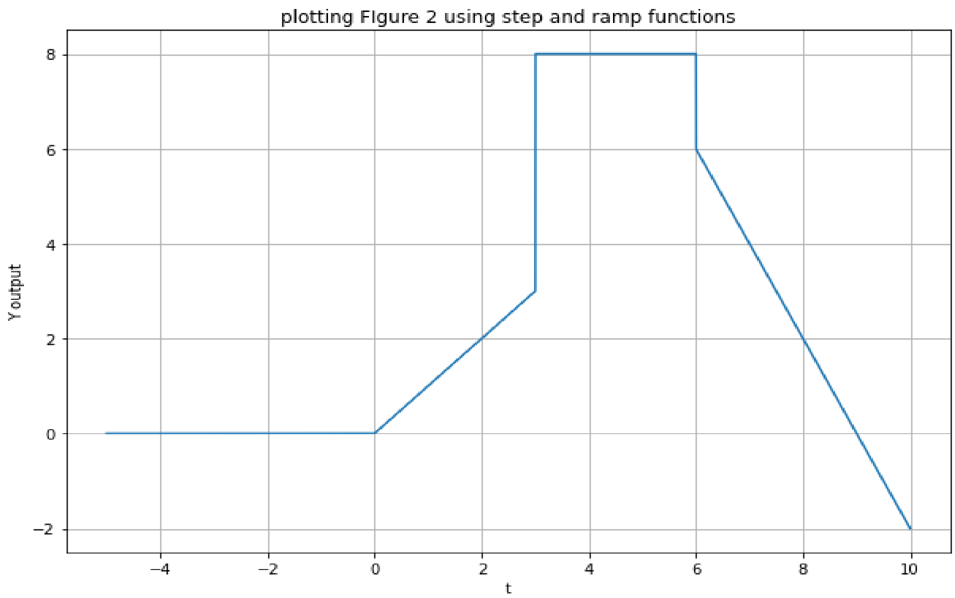
1

Figure 8: Step function



1

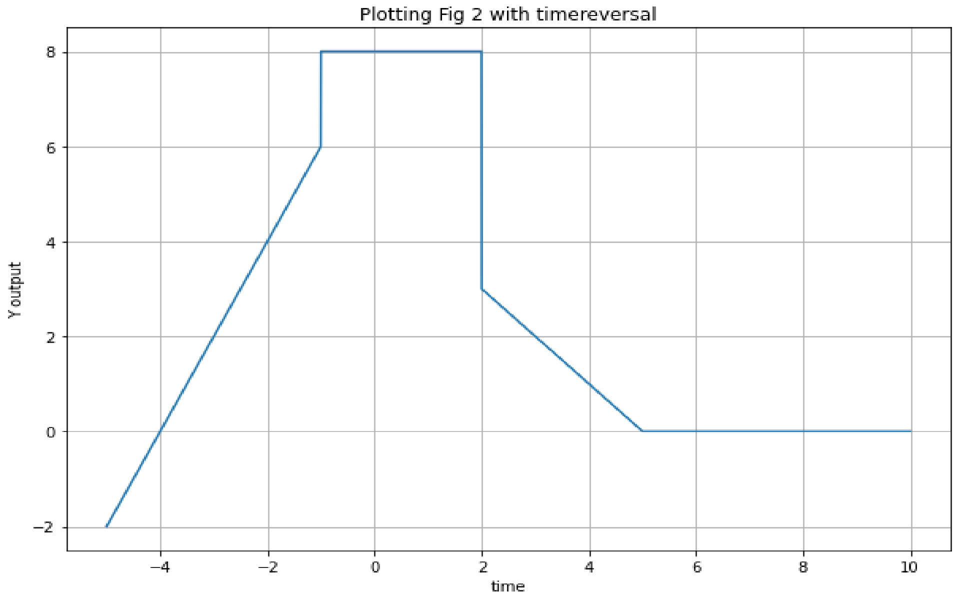
Figure 9: Ramp function



1

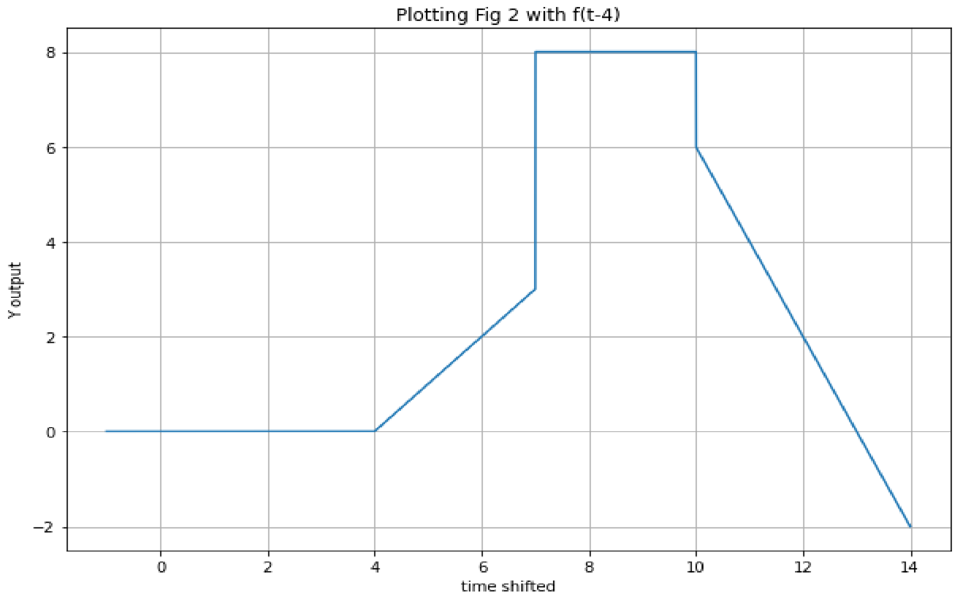
Figure 10: Imitated figure

3.3 Part 3



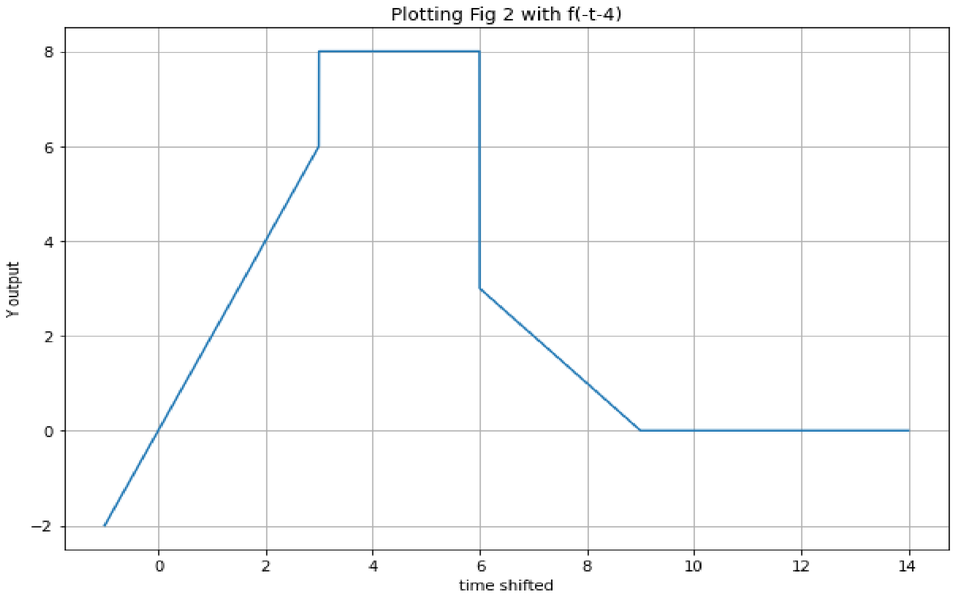
1

Figure 11: Time reversal figure



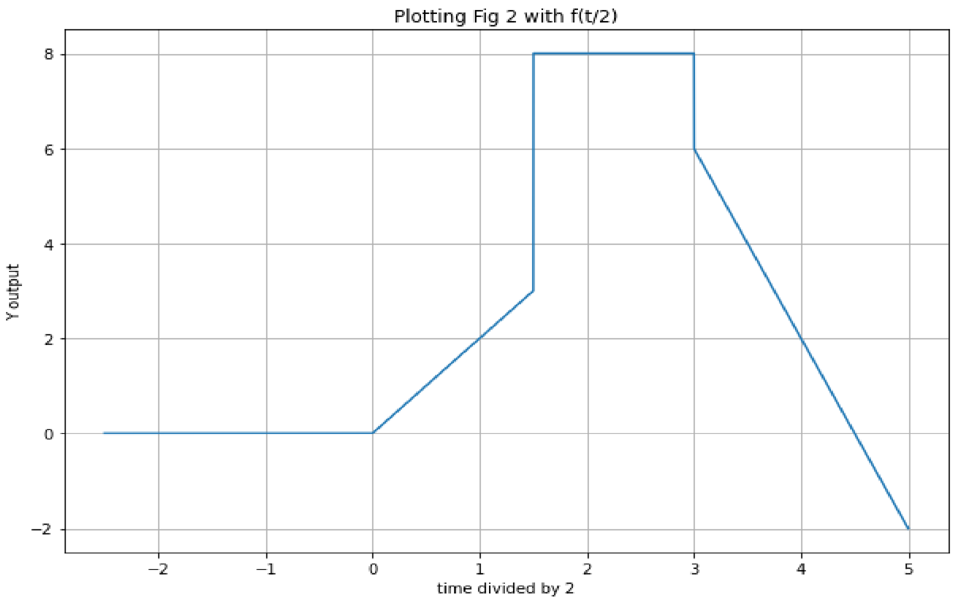
1

Figure 12: f(t-4) function



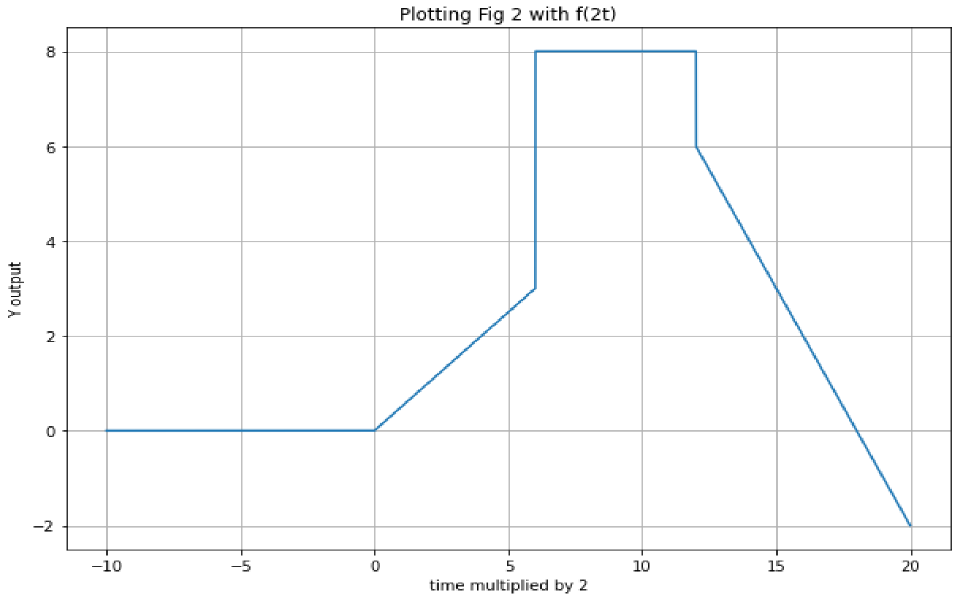
1

Figure 13: f(-t-4) function



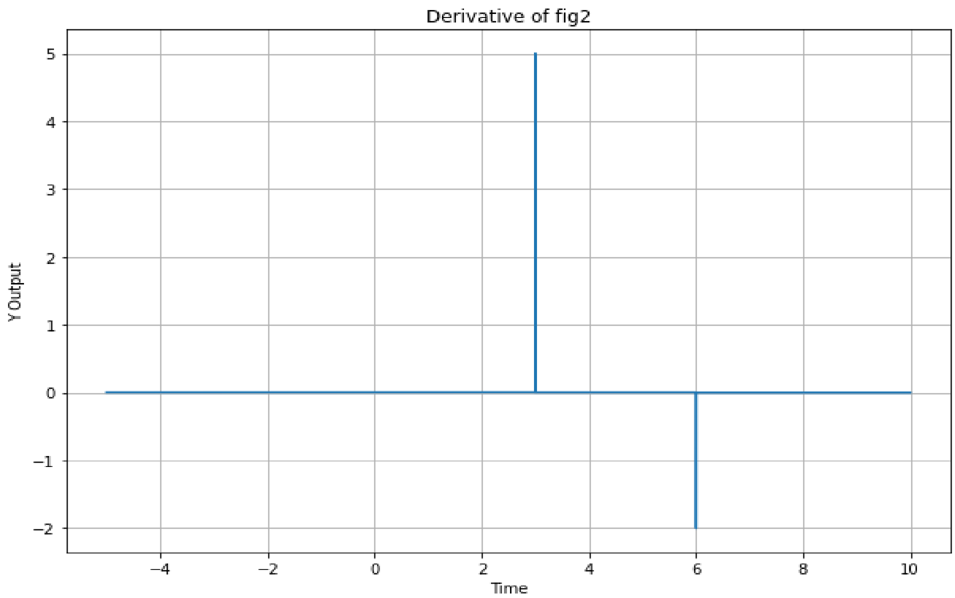
1

Figure 14: f(t/2) function



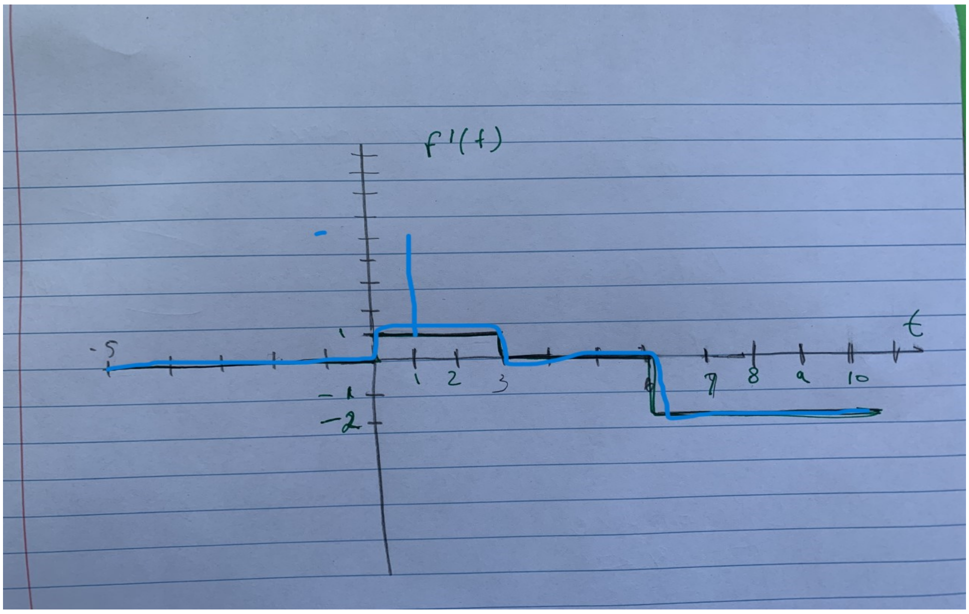
1

Figure 15: f(2t) function



1

Figure 16: Derivative function



1

Figure 17: Derivative function

4 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.

They are not quiet identical because pythons plot shows a spike at time 3 where mine doesn't match. I think it depends on how you are taking the derivative. Maybe in python since its array values it is taking derivatives more precisely with small intervals on the function so there is a derivative at 3 seconds. Therefore I think it is possible to match maybe by messing with how you spread your time range and steps. Maybe by making a step-size of 1.

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.

By shifting your step-size towards 1 it makes both graphs more identical.

3. Leave any feedback on the clarity of lab tasks, expectations, and deliverables.

I was just confuse if we could have made a couple subplots for Task 3 and not have a bunch of graphs. Another thing is on the time reverse it took me a while to know get it.Maybe an example shown would be helpful but that was just me. Aside from that it was straightforward but very long, hopefully because it was the first lab and figuring out latex.