assignment02

September 27, 2018

1 Mathematical Foundations for Computer Vision and Machine Learning

2 Assignment02 - Python Programming

2.1 Taylor Approximation

2.2 Name: Jinwoo Jeon

2.3 ## Student ID: 20143954

2.4 Setup

To make a python program for this assignment, you have to import 2 modules.

One is for making arrays that store the coordinates of x and ys, known as **numpy**, and the other is **matplotlib** that helps plotting the function to the screen. In this code,

```
In [1]: '''
        Subject: Human Interface Computer Vision
        Title: Assignment02 - Taylor Approximation
        Author: Jinwoo Jeon
        Student ID: 20143954
        '''

        # Import modules
        import matplotlib.pyplot as plt
        import numpy as np
```

2.5 Define a two dimensional differentiable function:

$$y = f(x) = x^3 - 5x^2 - 2x - 7 + e^{\frac{x+2}{2}}$$

2.6 Define the derivation of the given function:

$$f'(x) = 3x^2 - 10x - 2 + \frac{1}{2}e^{\frac{x+2}{2}}$$

The code for this part is below.

```
In [2]: # Define a two dimensional differentiable function
        def func(x):   # y = f(x)
            return x**3 -5*x**2 -2*x - 7 + np.exp(0.5*x+1)

        # Define the derivation of the given function
        def deri(x):
            return 3*x**2 -10*x -2 + 0.5*np.exp(0.5*x+1)
```

2.7 First Order Taylor Approximation at x = z:

$$\hat{f}(x) = f(z) + \frac{f'(z)}{1!}(x - z)$$

```
In [3]: # First Order Taylor Approximation
        def taylor_apporx(x,z):
            return func(z) + deri(z)*(x-z)
```

2.8 Define the domain of the function: x = [-5:0.05:5]

```
In [4]: diff = 0.05

        # Define the domain of the function
        t1 = np.arange(-5, 5, diff)
```

2.9 Pick 3 points in the domain:

$$P1 = (-2.0, f(-2.0))$$
$$P2 = (0.0, f(0.0))$$
$$P3 = (4.0, f(4.0))$$

In the code below, t2, t3, t4 are defined to plot approximated function around given points.

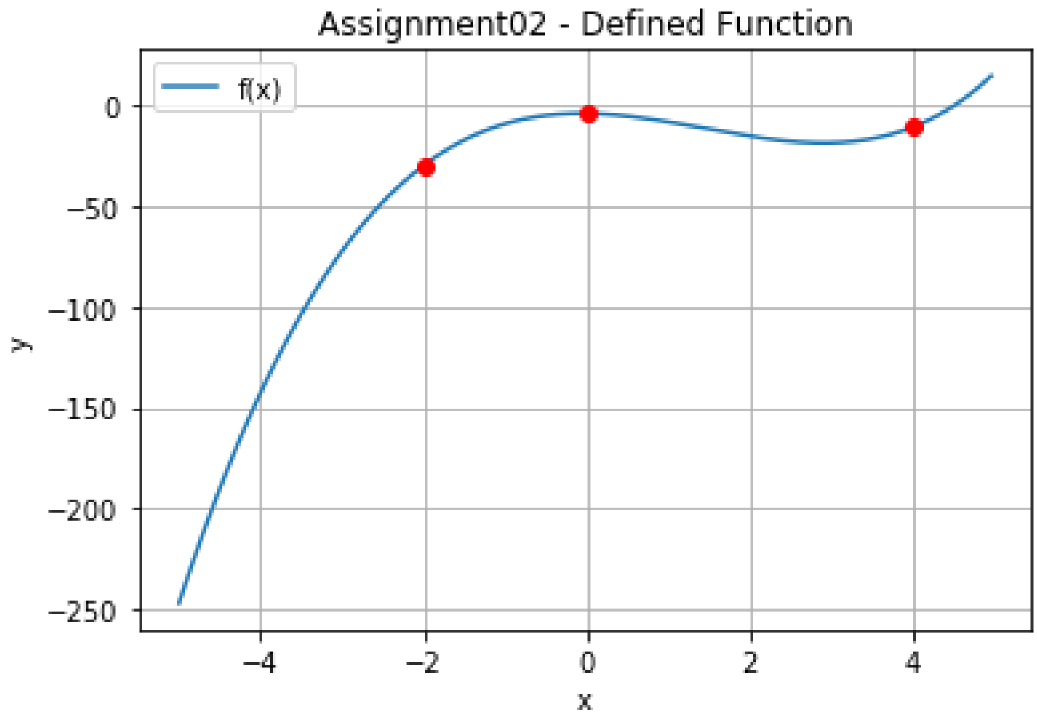
```
In [5]: # Pick 3 points in the domain
        # tap means taylor approximation point
        tap = [-2.0, 0.0, 4.0]

        #domain of the approximated function
        t2 = np.arange(tap[0] - diff, tap[0] + diff, diff/10)
        t3 = np.arange(tap[1] - diff, tap[1] + diff, diff/10)
        t4 = np.arange(tap[2] - diff, tap[2] + diff, diff/10)
```

2.10 Plot the graph of the defined function

The function is plotted below. There are three **red dots** that represent 3 points picked above.

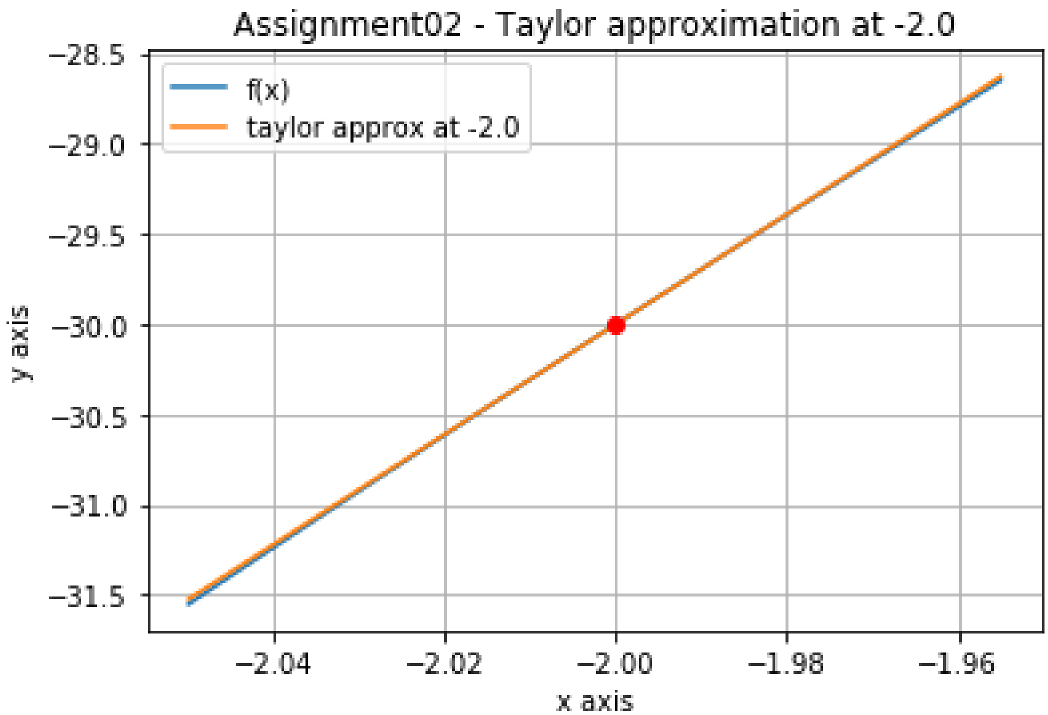
```
In [6]: # Plot the graph of the defined function
        plt.figure(1)
        plt.plot(t1, func(t1), label='f(x)')
        plt.plot(tap, [func(tap[0]), func(tap[1]), func(tap[2])] , 'ro') # 3 points
```

```
# plot informations of the graph
plt.xlabel('x')
plt.ylabel('y')
plt.title("Assignment02 - Defined Function")
plt.legend()
plt.grid(True)
```



2.11 Plot the graph of Taylor approximation for the given function at the given points(P1)

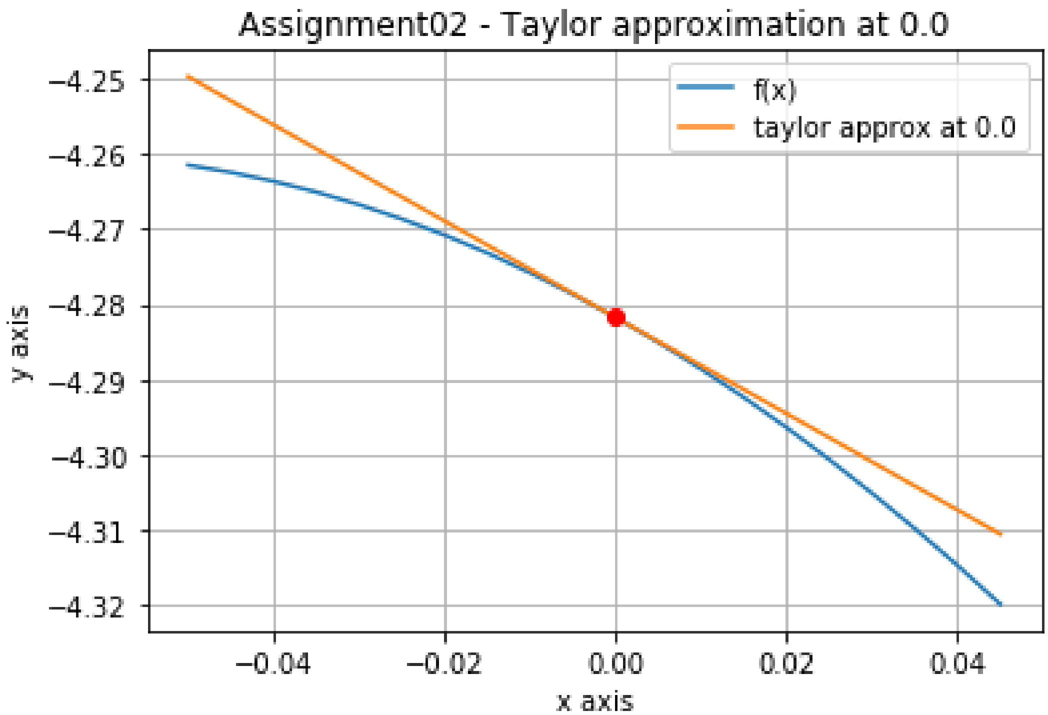
```
In [7]: # Plot the graph of Taylor approximation for the given function at the given points
        plt.figure(2)
        plt.plot(t2, func(t2), label='f(x)')
        plt.plot(t2, taylor_apporx(t2,tap[0]), label="taylor approx at {0}".format(tap[0]))
        plt.plot(tap[0], func(tap[0]), 'ro')

        # plot informations of the graph
        plt.xlabel('x axis')
        plt.ylabel('y axis')
        plt.title("Assignment02 - Taylor approximation at {0}".format(tap[0]))
        plt.legend()
        plt.grid(True)
```



You can check that **real function f(blue)** and **approximated function(orange)** is almost identical around `x = -2.0` .

2.12 Plot the graph of Taylor approximation for the given function at the given points(P2)

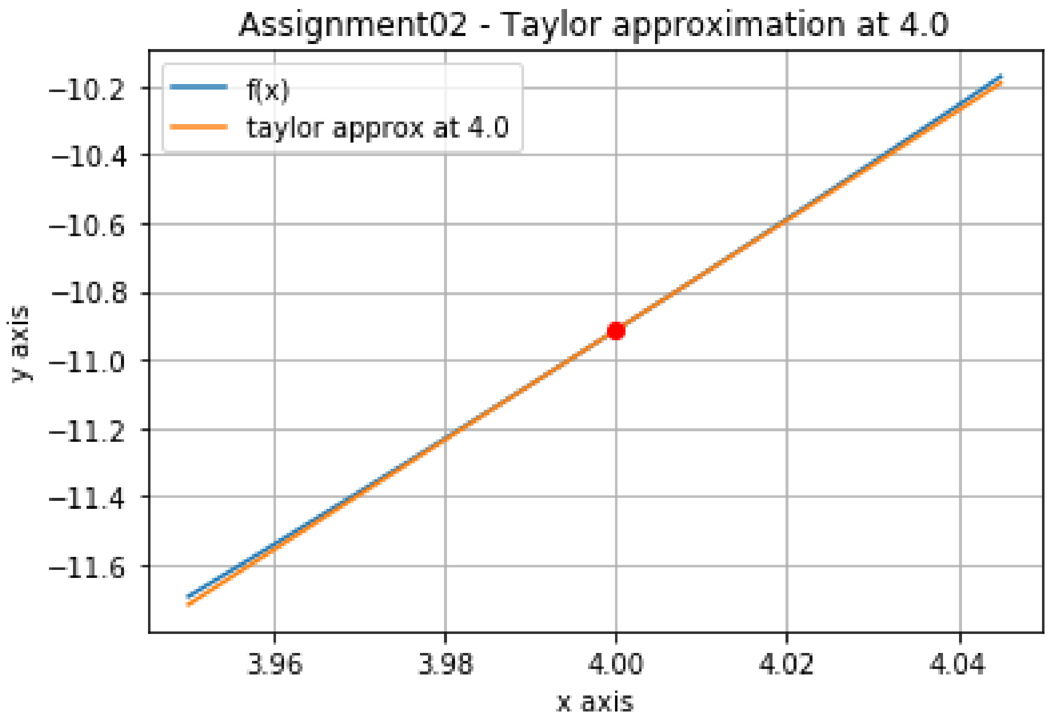
In [8]:
```
# Plot the graph of Taylor approximation for the given function at the given points
plt.figure(3)
plt.plot(t3, func(t3), label='f(x)')
plt.plot(t3, taylor_apporx(t3,tap[1]), label="taylor approx at {0}".format(tap[1]))
plt.plot(tap[1], func(tap[1]), 'ro')

# plot informations of the graph
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title("Assignment02 - Taylor approximation at {0}".format(tap[1]))
plt.legend()
plt.grid(True)
```



You can check that **real function f(blue)** and **approximated function(orange)** is almost identical around `x = 0.0`.

2.13 Plot the graph of Taylor approximation for the given function at the given points(P3)

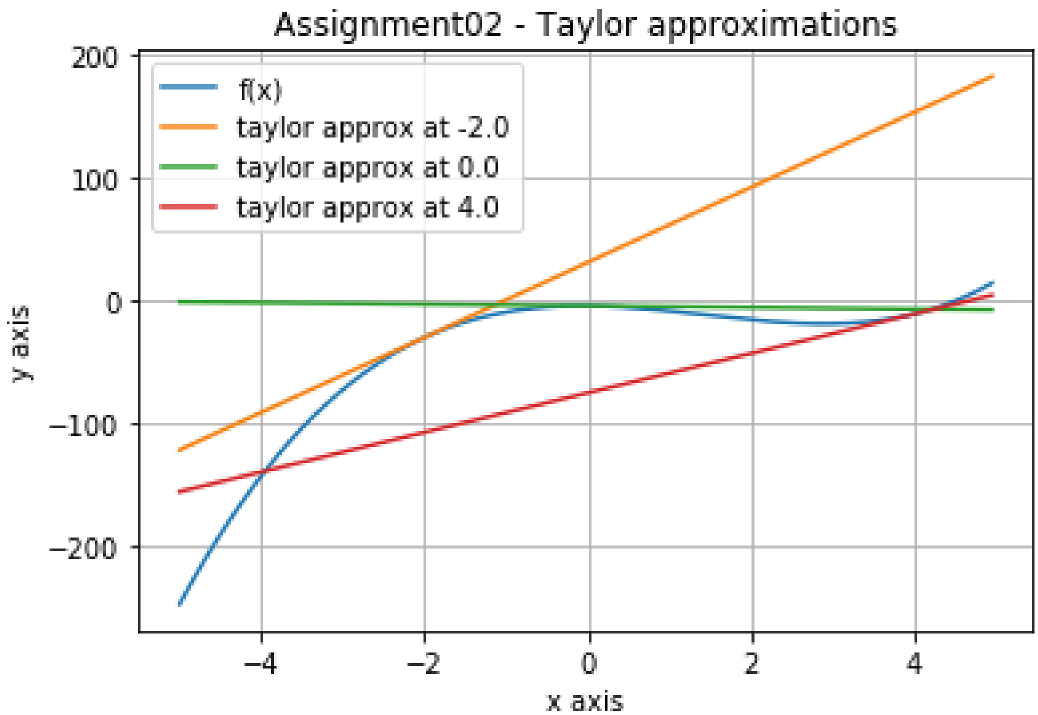
In [9]:
```
# Plot the graph of Taylor approximation for the given function at the given points
plt.figure(4)
plt.plot(t4, func(t4), label='f(x)')
plt.plot(t4, taylor_apporx(t4,tap[2]), label="taylor approx at {0}".format(tap[2]))
plt.plot(tap[2], func(tap[2]), 'ro')

# plot informations of the graph
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title("Assignment02 - Taylor approximation at {0}".format(tap[2]))
plt.legend()
plt.grid(True)
```



You can check that **real function f(blue)** and **approximated function(orange)** is almost identical around `x = 4.0`.

2.14 Plot all the taylor appoximation at the given points to check that appoximation is not accurate in whole domain

In [10]:
```
# Plot all the taylor appoximation at the given points to check that appoximation is 
plt.figure(5)
plt.plot(t1, func(t1), label='f(x)')
plt.plot(t1, taylor_apporx(t1, tap[0]), label="taylor approx at {0}".format(tap[0]))
plt.plot(t1, taylor_apporx(t1, tap[1]), label="taylor approx at {0}".format(tap[1]))
plt.plot(t1, taylor_apporx(t1, tap[2]), label="taylor approx at {0}".format(tap[2]))
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title("Assignment02 - Taylor approximations")
plt.legend()
plt.grid(True)
```



You can see that approximated functions are not identical when it comes to original domain **t1**. They are only useful around given points and just a tangent line of given function.

2.15 Show all the graphs on the screen:

```
In [11]: # plot all figures
         plt.show()
```