Assignment02

March 20, 2019

20164245 Hong Jin

1 Import packages numpy and matplotlib

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
```

2 Define a differentiable function that maps from real number to real number.

2.1 Define $f(x) = x * sinx + x * cosx $ as func(x)

```
In [2]: def func(x):
            f =  x * np.sin(x) + x * np.cos(x)
            return f
```

3 Define a domain of the function.

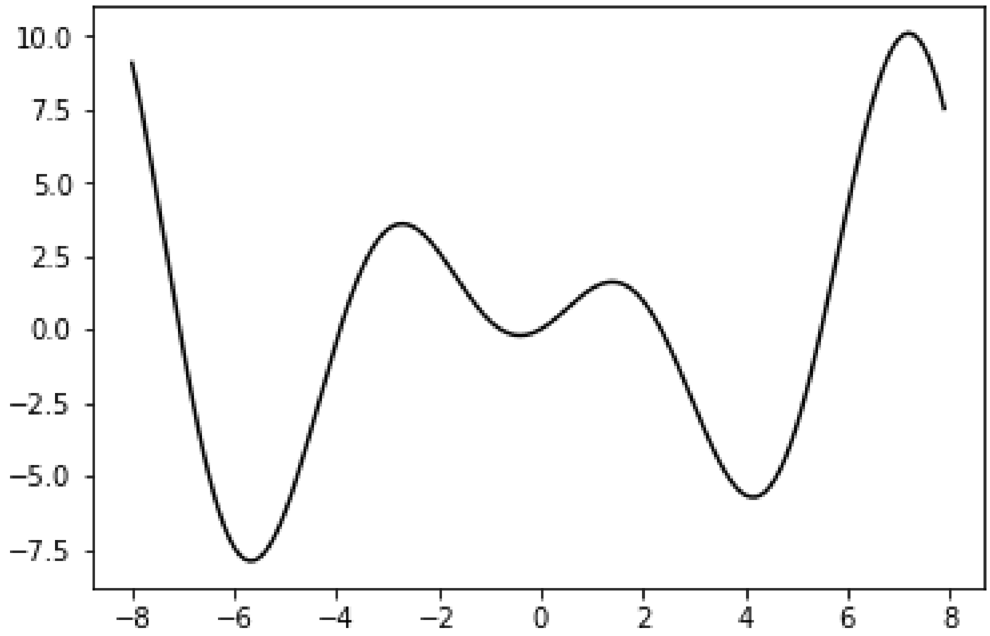
3.1 Domain : $-8 < x < 8$

```
In [3]: x = np.arange(-8,8,0.1)
```

4 Plot the function.

```
In [4]: f = func(x)
        plt.figure(1)
        plt.plot(x, f, 'k', label="function")

Out[4]: [<matplotlib.lines.Line2D at 0x1f29bacb3c8>]
```



5 Select a point within the domain.

5.1 Point x : -3, 4

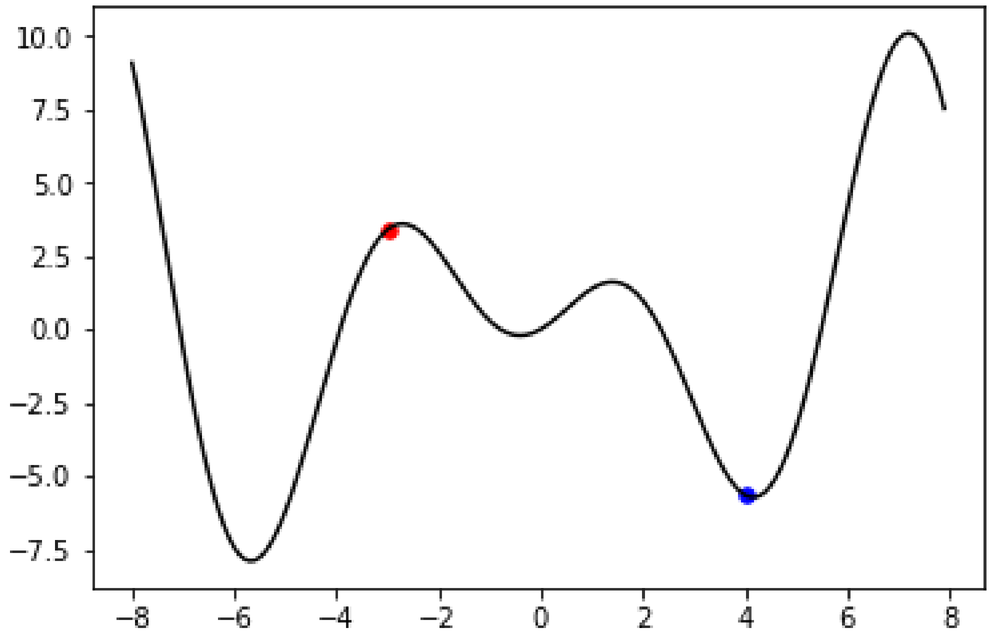
```
In [5]: p1 = -3
        p2 = 4
```

5.2 Mark the selected point on the function.

```
In [6]: def origin_plot():
            y1 = func(p1)
            y2 = func(p2)

            plt.scatter(p1,y1,c ='r',s = 30)
            plt.scatter(p2,y2, c ='b',s = 30)
            plt.plot(x,f,'k',label="function")

        origin_plot()
```



6 Define the first-order Taylor approximation at the selected point.

6.1 Define $f'(x) = sinx + x * cosx + cosx - x * sinx$ as derivate function d_func(x)

```
In [7]: def d_func(x):
            df = np.sin(x) + x*np.cos(x) + np.cos(x) - x*np.sin(x)
            return df
```

6.2 Define Tylor Approximation $f(p) + f'(p)(x - p)$

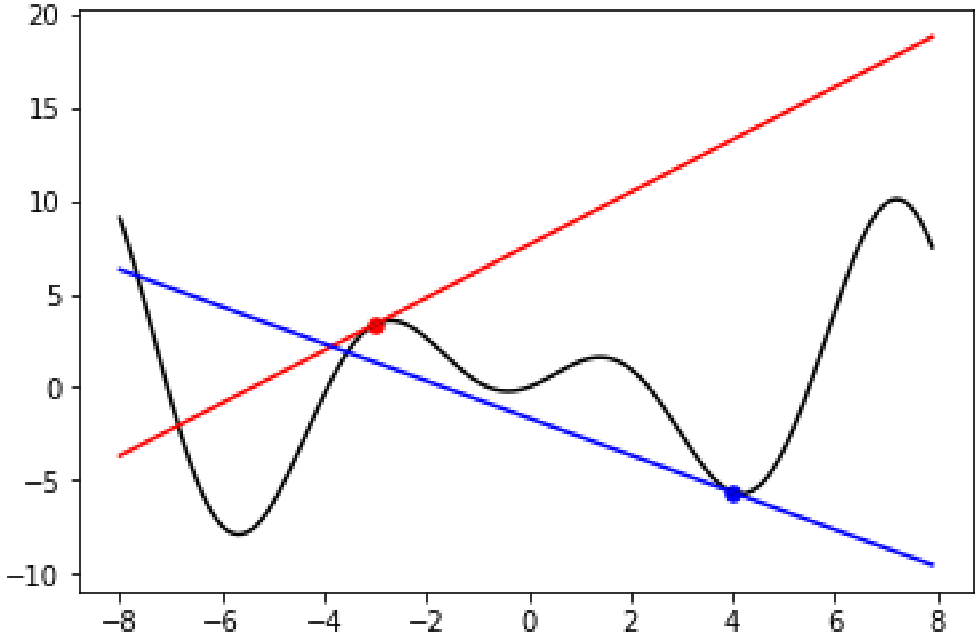
```
In [8]: def tylor(p,x):
            result = func(p) + d_func(p) * (x-p)
            return result
```

7 Plot the Tylor approximation with the same domain of the original function.

```
In [9]: origin_plot()

        df1 = tylor(p1,x)
        df2 = tylor(p2,x)
        plt.plot(x, df1, 'r')
        plt.plot(x, df2, 'b')
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x1f29bbfae10>]
```


20
15
10
5
0
−5
−10
−8
−6
−4
−2
0
2
4
6
8