Programes

1. Find the derivate of using limit of the difference coeffficient method at x = 1.

In [ ]:

```
import math
def function(x):
    x2=pow(x,2)
    value=pow(math.e,x2)+math.sin(x)-math.tan(x)+math.log(x)
    return value

def derivative(f,x,h=0.00001):
    return (f(x + h) - f(x))/h


print('the derivative of given function at x=1 is ',derivative(function,1))
```

```
the derivative of given function at x=1 is  3.5513661338359976
```

2. Find the gradient of Rosenbrock function using limit of the difference coeffficient method at the point (1,2). Rosenbrock function is defined below.

In [ ]:

```
def function(x,y):
    '''Rosenbrock function'''
    return pow((1-x),2)+100*(pow((y-x*x),2))

def p_derivative_wrt_x(f,x,y,h=0.0001):
    return (f(x + h,y) - f(x,y))/h

def p_derivative_wrt_y(f,x,y,h=0.0001):
    return (f(x,y+h) - f(x,y))/h

x,y=(1,2)
print(p_derivative_wrt_x(function,x,y))
print(p_derivative_wrt_y(function,x,y))
```

```
-399.9798959998202
200.0100000003613
```

3. Find the point of minima of function using Gradient Descent method taking initial solution x0 = 2.

In [ ]:

```
def function(x):
    return x*x+math.sin(x)

def slope_at_x(x):
    return 2*x+math.cos(x)

def get_minima(x,num_iterations=100000):
    '''new = old - slope * learning_rate''' #by gradient descent
    minima=x
    learning_rate=0.001
    for i in range(num_iterations):
        minima=minima-slope_at_x(minima)*learning_rate
    return minima

minima=get_minima(2)
print(f'local minima is at x = {minima} and value is {function(minima)}')
```

```
local minima is at x = -0.4501836112948622 and value is -0.23246557515821561
```

In [ ]:

In [ ]:

From Book

In [ ]:
```python
def f(x:float)->float:
    '''this is equation: y=x^2+2x+5'''
    return x*x+2*x+5
```

In [ ]:
```python
from typing import Callable
def difference_quotient(f: Callable[[float], float], x: float, h: float) -> float:
    return (f(x + h) - f(x)) / h
```

In [ ]:
```python
difference_quotient(f,3,0.00001)
```

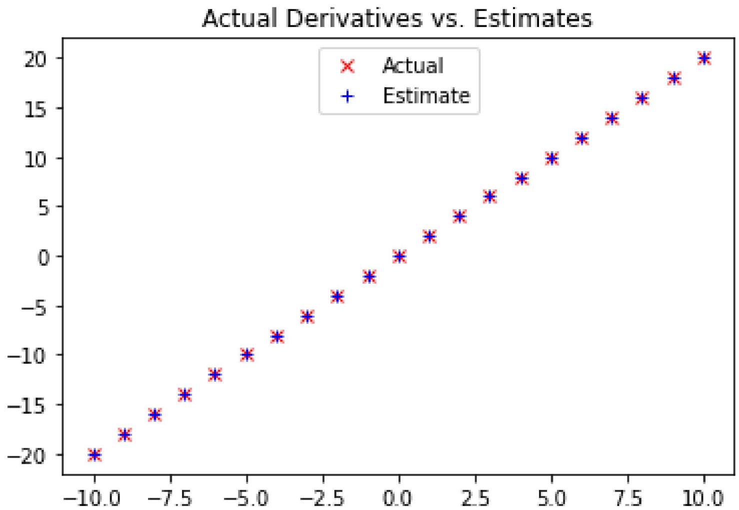
Out[ ]:
```
8.000009999875601
```

In [ ]:
```python
def square(x:float)->float:
    return x*x
def derivative(x:float)->float:
    return 2*x
```

In [ ]:
```python
xs = range(-10, 11)
actuals = [derivative(x) for x in xs]
estimates = [difference_quotient(square, x, h=0.001) for x in xs]
print(actuals)
print(estimates)
```

```
[-20, -18, -16, -14, -12, -10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
[-19.998999999984335, -17.998999999988996, -15.999000000007868, -13.999000000005424, -1
1.99900000000298, -9.999000000004088, -7.998999999999867, -5.998999999999199, -3.9989999
999994197, -1.998999999999973, 0.001, 2.0009999999996975, 4.000999999999699, 6.000999999
999479, 8.0010000000037, 10.001000000002591, 12.001000000005035, 14.00100000000748, 16.0
00999999988608, 18.000999999983947, 20.000999999993496]
```

In [ ]:
```python
# plot to show they're basically the same
import matplotlib.pyplot as plt
plt.title("Actual Derivatives vs. Estimates")
plt.plot(xs, actuals, 'rx', label='Actual') # red x
plt.plot(xs, estimates, 'b+', label='Estimate') # blue +
plt.legend(loc=9)
plt.show()
```



In [ ]:

```
def partial_difference_quotient(f: Callable[[Vector], float], v: Vector, i: int, h: flo
    """Returns the i-th partial difference quotient of f at v"""
    w = [v_j + (h if j == i else 0) for j, v_j in enumerate(v)] # add h to just the ith
    return (f(w) - f(v)) / h
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Testing

In [ ]:

```
l1 = ["eat", "sleep", "repeat"]
s1 = "geek"

# creating enumerate objects
obj1 = enumerate(l1)
obj2 = enumerate(s1)

print ("Return type:", type(obj1))
print (list(enumerate(l1)))

# changing start index to 2 from 0
print (list(enumerate(s1, 2)))
```

```
for i,j in obj1:
    print(i,j)
```

```
Return type: <class 'enumerate'>
[(0, 'eat'), (1, 'sleep'), (2, 'repeat')]
[(2, 'g'), (3, 'e'), (4, 'e'), (5, 'k')]
0 eat
1 sleep
2 repeat
```

In [ ]:

```
import math
math.e
```

Out[ ]:

```
2.718281828459045
```

In [ ]:

```
# !pip install pyppeteer
```

In [ ]:

```
# !jupyter nbconvert --execute --to pdf assignment3.ipynb
```

In [ ]:

```
def gradientDescent(X, y, theta, alpha, num_iters):
    """
       Performs gradient descent to learn theta
    """
    m = y.size  # number of training examples
    for i in range(num_iters):
        y_hat = np.dot(X, theta)
        theta = theta - alpha * (1.0/m) * np.dot(X.T, y_hat-y)
    return theta
```