Tazmeen Afroz

Roll No : 22P-9252

LAB-08

Implementation of Gradient Descent

cost function

In [1]:
```
import numpy as np

#Function to calculate the cost
def compute_cost(x, y, w, b):

    m = x.shape[0]
    cost = 0

    for i in range(m):
        f_wb = w * x[i] + b
        cost = cost + (f_wb - y[i])**2
    total_cost = 1 / (2 * m) * cost

    return total_cost
```

minimize cost function for single iteration

In [2]:
```
# first iteration
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

w = 100
b = 20
alpha = 0.1
m = x.shape[0]

f_wb = w * x + b
print(f_wb)


temp_w = w - alpha * (1 / m) * np.sum((f_wb - y) * x)

temp_b = b - alpha * (1 / m) * np.sum(f_wb - y)


print("Cost before update: ", compute_cost(x, y, w, b))

print("Cost after update: ", compute_cost(x, y, temp_w, temp_b))
```

```
print("Updated w: ", temp_w)

print("Updated b: ", temp_b)
```

```
[120 220 320 420 520]
Cost before update:  58902.0
Cost after update:  1978.3600000000058
Updated w:  -13.800000000000026
Updated b:  -11.400000000000006
```

for n iterations

In [3]:
```
# for many iterations

for i in range(m):
    w = temp_w
    b = temp_b
    f_wb = w * x + b
    temp_w = w - alpha * (1 / m) * np.sum((f_wb - y) * x)
    temp_b = b - alpha * (1 / m) * np.sum(f_wb - y)
    print("Cost after update: ", compute_cost(x, y, temp_w, temp_b))
    if compute_cost(x, y, temp_w, temp_b) == 0:
        break
print("Updated w: ", temp_w)

print("Updated b: ", temp_b)

print("Cost after update: ", compute_cost(x, y, temp_w, temp_b))

print("Number of iterations: ", i)
```

```
Cost after update:  69.93520000000046
Cost after update:  5.836336000000027
Cost after update:  3.57043552
Cost after update:  3.3813382335999993
Cost after update:  3.2656561227519996
Updated w:  3.6628
Updated b:  -5.989152
Cost after update:  3.2656561227519996
Number of iterations:  4
```

finding optimal value of w and b over n iterations

In [5]:
```
import numpy as np


cost_threshold =  4
min_cost = float('inf')

for i in range(m):
    w = temp_w
    b = temp_b
    f_wb = w * x + b
    temp_w = w - alpha * (1 / m) * np.sum((f_wb - y) * x)
    temp_b = b - alpha * (1 / m) * np.sum(f_wb - y)
```

```
    current_cost = compute_cost(x, y, temp_w, temp_b)
    print("Cost after update: ", current_cost)

    if current_cost < min_cost:
        min_cost = current_cost
        optimal_w = temp_w
        optimal_b = temp_b



    if current_cost == 0 or current_cost > cost_threshold:
        break

print("Updated w: ", temp_w)
print("Updated b: ", temp_b)
print("Cost after update: ", compute_cost(x, y, temp_w, temp_b))
print("Number of iterations: ", i)
print("Minimum cost: ", min_cost)
print("Optimal w: ", optimal_w)
print("Optimal b: ",optimal_b)
```

```
Cost after update:  3.050293505606195
Cost after update:  2.948035840853135
Cost after update:  2.849206331271493
Cost after update:  2.753689970027852
Cost after update:  2.6613756848946557
Updated w:  3.497842642048
Updated b:  -5.40768667008
Cost after update:  2.6613756848946557
Number of iterations:  4
Minimum cost:  2.6613756848946557
Optimal w:  3.497842642048
Optimal b:  -5.40768667008
```

In [ ]: