**Assignment No.: ML 4**

**Title:** Implement Gradient Descent Algorithm to find the local minima of a function.

For example, find the local minima of the function y=(x+3)² starting from the point x=2.

**Objective:**

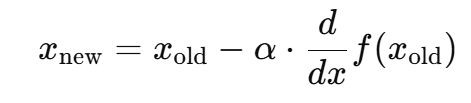
The objective is to implement the **Gradient Descent Algorithm** to find the local minima of the function y=(x+3)2, starting from an initial point x=2. The goal is to iteratively move toward the local minimum of the function by updating the value of x in the direction of the negative gradient.

**Theory of Gradient Descent:**

**Gradient Descent** is an optimization algorithm used to minimize a function by iteratively moving in the direction of the steepest descent, as defined by the negative of the gradient (derivative). It is commonly used in machine learning and optimization problems.

* **Gradient**: The gradient of a function at a particular point gives the direction and rate of the steepest increase in the function’s value. In one-dimensional functions, the gradient is simply the derivative.
* **Learning rate (α\alphaα)**: This is a small positive number that controls the size of the steps taken to reach the minimum. If the learning rate is too large, the algorithm might overshoot the minimum, and if it's too small, the convergence will be slow.
* **Stopping condition**: Gradient descent is an iterative process, and it stops when the gradient becomes sufficiently small (close to zero), meaning the algorithm has reached a local minimum.

For a function ***y=f(x)***, the gradient descent update rule is:



**Problem:**

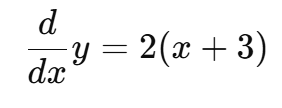
We are tasked with finding the local minimum of the function:

y=(x+3)2

Starting at x=2, we will implement the gradient descent algorithm to minimize this function.

**Steps:**

1. **Compute the gradient (derivative)**: The derivative of ***y=(x+3)2***with respect to ***x*** is:



1. **Update the value of *x*** based on the gradient.
2. **Iterate** until the gradient is close to zero, indicating convergence to a local minimum. The algorithm stops either when the number of iterations is reached or the gradient becomes very small (less than ***1e−6***).

**Python Code:**

|  |
| --- |
| import matplotlib.pyplot as plt  # Define the function and its derivative  def function(x):      return (x + 3)\*\*2  def gradient(x):      return 2 \* (x + 3)  # Gradient Descent Algorithm  def gradient\_descent(starting\_x, learning\_rate, num\_iterations):      x = starting\_x      x\_values = [x]  # Track x values for visualization      for i in range(num\_iterations):          grad = gradient(x)          x = x - learning\_rate \* grad          x\_values.append(x)          if abs(grad) < 1e-6:  # Stopping condition when gradient is near 0              break      return x, x\_values  # Parameters for Gradient Descent  starting\_x = 2  # Starting point  learning\_rate = 0.1  # Learning rate (alpha)  num\_iterations = 100  # Maximum number of iterations  # Perform Gradient Descent  final\_x, x\_values = gradient\_descent(starting\_x, learning\_rate, num\_iterations)  # Print the final result  print(f"The local minimum occurs at x = {final\_x:.6f}")  # Plotting the function and the gradient descent path  x\_range = range(-10, 5)  y\_values = [function(x) for x in x\_range]  plt.plot(x\_range, y\_values, label='y = (x + 3)^2')  plt.scatter(x\_values, [function(x) for x in x\_values], color='red', label='Gradient Descent Path')  plt.xlabel('x')  plt.ylabel('y')  plt.title('Gradient Descent to find local minima')  plt.legend()  plt.grid(True)  plt.show() |

**Conclusion:**

* **Gradient Descent Algorithm** successfully finds the local minimum of the function ***y=(x+3)2*** starting from ***x=2.***
* The algorithm iteratively reduces the value of xxx until it converges to the local minimum at ***x=−3x*** , which is the minimum value of the function ***y***.
* The visualization shows how the algorithm updates the values of ***x*** to approach the minimum.

By adjusting the learning rate and the number of iterations, the convergence speed can be controlled, and this algorithm can be applied to other convex functions to find their minima.