Name: **Salman Irfan**

Roll no: **2021-MSCS-511**

Course: **Artificial Neural Networks**

Submitted To: **Dr. Muhammad Awais Hassan**

Assignment 4: Gradient Vector: -

■ Write a python program that calculate the gradient vector of a function at any given point. Multiply it with 2, and calculate new point in direction of the gradient vector)

■ Draw the Function on Graph and Highlight the starting point and new calculated point

Source Code: -

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d

# Step 1: Define the function


def quadratic_function(x, y):
    return x**2 - y**2

# Step 2: Define the partial derivatives


def partial_deriv_x(x, y):
    return 2*x


def partial_deriv_y(x, y):
    return -2*y

# Step 3: Define the gradient vector function


def gradient_vector(x, y):
    grad_x = partial_deriv_x(x, y)
    grad_y = partial_deriv_y(x, y)
    return (grad_x, grad_y)


# Step 4: Visualization
# Set up a 2x2 plot layout
```

```
# The plt.subplots() function returns a tuple containing a Figure instance and a
NumPy array of Axes instances, respectively.
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Plot the function
X, Y = np.meshgrid(np.linspace(-10, 10, 100), np.linspace(-10, 10, 100))
Z = quadratic_function(X, Y)
axs[0, 0].contour(X, Y, Z, levels=np.linspace(-100, 100, 50))
axs[0, 0].set_title("Graph of the Function")

# Plot the point at which we want to find the gradient vector
x, y = 1, 1
axs[0, 1].contour(X, Y, Z, levels=np.linspace(-100, 100, 50))
axs[0, 1].scatter(1, 1, color='blue')
axs[0, 1].set_title("Graph at point (1,1)")

# Plot the gradient vector at the point (1, 1)
grad_vec = gradient_vector(x, y)
axs[1, 0].arrow(x, y, grad_vec[0]*2, grad_vec[1]*2)  # multiplied with 2
axs[1, 0].set_title("Gradient vector at point (1,1)")

# Show the point and gradient vector on the function plot
axs[1, 1].contour(X, Y, Z, levels=np.linspace(-100, 100, 50))
axs[1, 1].set_xlabel('B0')
axs[1, 1].set_ylabel('B1')
axs[1, 1].plot(x, y, marker=".", label='Starting Point')
# The quiver() method in matplotlib is used to plot a vector field, which
consists of a collection of arrows or vectors that represent the magnitude and
direction of some physical quantity (e.g., velocity, electric field, or in this
case, the gradient of a function).
axs[1, 1].quiver(x, y, grad_vec[0]*2, grad_vec[1]*2, angles='xy',
                    scale_units='xy', scale=1,  label='Gradient Vector')
axs[1, 1].set_title("Function with Point and Gradient Vector")
axs[1, 1].legend()

# Show the plot
plt.show()

# Step 5: Visualization in 3D
fig = plt.figure(figsize=(12, 6))

# Plot the function in 3D
ax = fig.add_subplot(1, 2, 1, projection='3d')
X, Y = np.meshgrid(np.linspace(-10, 10, 100), np.linspace(-10, 10, 100))
Z = quadratic_function(X, Y)
```

```
ax.plot_surface(X, Y, Z, cmap='cool')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title("Graph of the Function")

# Plot the gradient vector in 3D
ax = fig.add_subplot(1, 2, 2, projection='3d')
x, y = 1, 1
grad_vec = gradient_vector(x, y)
ax.plot_surface(X, Y, Z, cmap='cool')
ax.quiver(x, y, quadratic_function(x, y),
            grad_vec[0]*2, grad_vec[1]*2, 0, length=25, normalize=True)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title("Gradient vector at point (1,1)")


# Show the plot
plt.show()
```

Step 01: Define a function

■ First you need to define the function with two argument for that you need to find the gradient vector.

■ This function take two arguments and return the value of f(B o ,B1) according to the passed parameters.

■ Let the function is ■ f(B o ,B1) = x 2 - y2

```
# Step 1: Define the function
def quadratic_function(x, y):
    return x**2 - y**2
```

Step 02: Functions to calculate

■ Define two separate functions that calculates the partial derivative of the given functions with respect to B0 and B1.

■ Each of these functions will take one argument and return the value of the partial derivative at given point.

```
# Step 2: Define the partial derivatives
def partial_deriv_x(x, y):
    return 2*x
def partial_deriv_y(x, y):
    return -2*y
```

Step 03: Function to find the Gradient: -

■ Write function to find the Gradient vector at any given point. Take two values as input and return two value in form of pair (gradient vector)

```
# Step 3: Define the gradient vector function
def gradient_vector(x, y):
    grad_x = partial_deriv_x(x, y)
    grad_y = partial_deriv_y(x, y)
    return (grad_x, grad_y)
```

Step 04: Visualization: -

■ Draw the graph of the function with some values of B0 and B1 let within the interval [-10,10]

■ Draw the graph the point at which we want to find the gradient vector

■ Draw the gradient vector point on the graph (you can multiply gradient vector with 2)

```
# Step 4: Visualization
```

```
# Set up a 2x2 plot layout
# The plt.subplots() function returns a tuple containing a Figure instance and a
NumPy array of Axes instances, respectively.
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Plot the function
X, Y = np.meshgrid(np.linspace(-10, 10, 100), np.linspace(-10, 10, 100))
Z = quadratic_function(X, Y)
axs[0, 0].contour(X, Y, Z, levels=np.linspace(-100, 100, 50))
axs[0, 0].set_title("Graph of the Function")

# Plot the point at which we want to find the gradient vector
x, y = 1, 1
axs[0, 1].contour(X, Y, Z, levels=np.linspace(-100, 100, 50))
axs[0, 1].scatter(1, 1, color='blue')
axs[0, 1].set_title("Graph at point (1,1)")

# Plot the gradient vector at the point (1, 1)
grad_vec = gradient_vector(x, y)
axs[1, 0].arrow(x, y, grad_vec[0]*2, grad_vec[1]*2)  # multiplied with 2
axs[1, 0].set_title("Gradient vector at point (1,1)")

# Show the point and gradient vector on the function plot
axs[1, 1].contour(X, Y, Z, levels=np.linspace(-100, 100, 50))
axs[1, 1].set_xlabel('B0')
axs[1, 1].set_ylabel('B1')
axs[1, 1].plot(x, y, marker=".", label='Starting Point')
# The quiver() method in matplotlib is used to plot a vector field, which
consists of a collection of arrows or vectors that represent the magnitude and
direction of some physical quantity (e.g., velocity, electric field, or in this
case, the gradient of a function).
axs[1, 1].quiver(x, y, grad_vec[0]*2, grad_vec[1]*2, angles='xy',
                    scale_units='xy', scale=1,  label='Gradient Vector')
axs[1, 1].set_title("Function with Point and Gradient Vector")
axs[1, 1].legend()

# Show the plot
plt.show()
```

Optional: 3D Visualization: -

```
# Step 5: Visualization in 3D
```

```
fig = plt.figure(figsize=(12, 6))

# Plot the function in 3D
ax = fig.add_subplot(1, 2, 1, projection='3d')
X, Y = np.meshgrid(np.linspace(-10, 10, 100), np.linspace(-10, 10, 100))
Z = quadratic_function(X, Y)
ax.plot_surface(X, Y, Z, cmap='cool')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title("Graph of the Function")

# Plot the gradient vector in 3D
ax = fig.add_subplot(1, 2, 2, projection='3d')
x, y = 1, 1
grad_vec = gradient_vector(x, y)
ax.plot_surface(X, Y, Z, cmap='cool')
ax.quiver(x, y, quadratic_function(x, y),
            grad_vec[0]*2, grad_vec[1]*2, 0, length=25, normalize=True)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title("Gradient vector at point (1,1)")


# Show the plot
plt.show()
```

Output: -

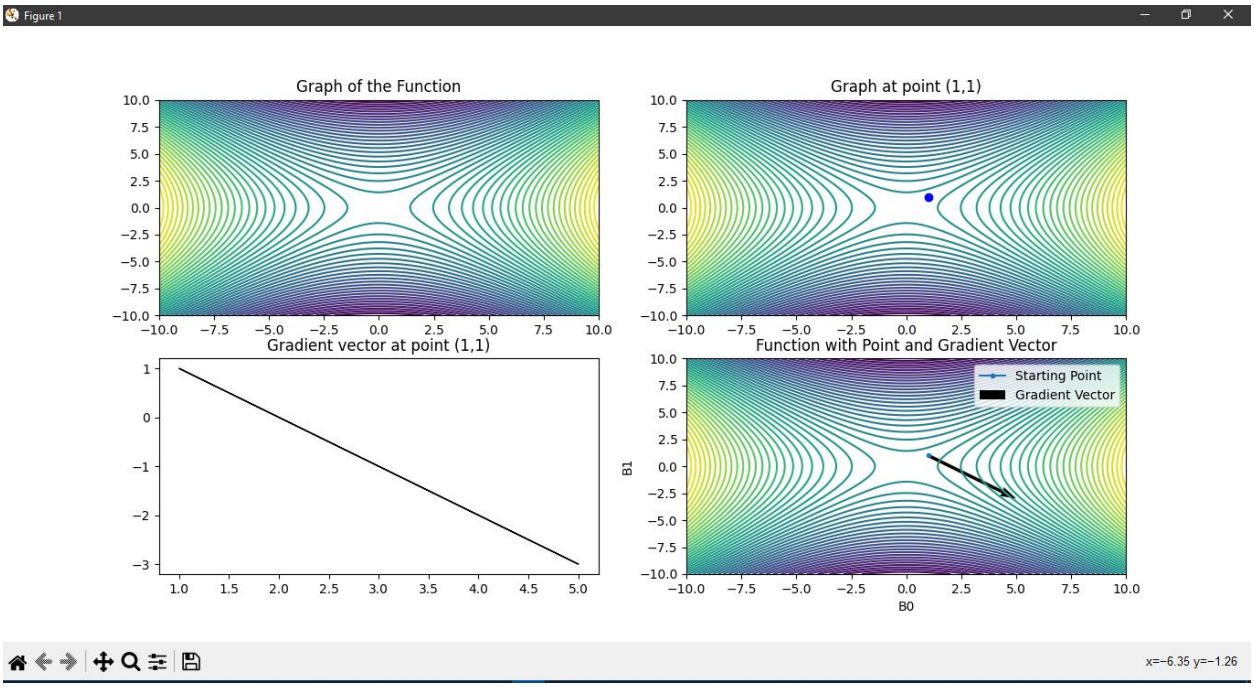
Figure 1
Graph of the Function
Graph at point (1,1)
Gradient vector at point (1,1)
Function with Point and Gradient Vector
Starting Point
Gradient Vector
B0
B1
x=−6.35 y=−1.26

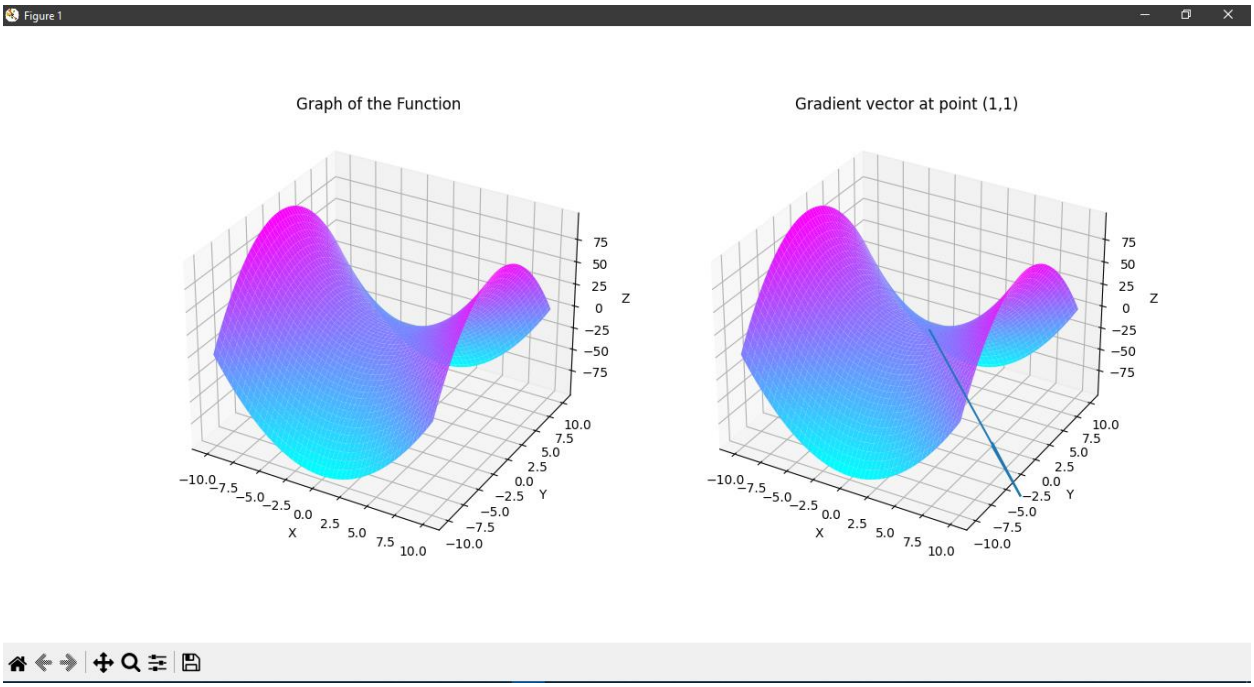
Figure 1
Graph of the Function
Gradient vector at point (1,1)
X
Y
Z