**Name: Sai Anish Garapati**
**UIN: 650208577**

**Assignment 4:**

**1.**
**(a)**

(1) $f(x,y) = -\log(1-x-y) - \log x - \log y$

(a) gradient of $f(x,y)$

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial}{\partial x} (-\log(1-x-y) - \log x - \log y \\ \frac{\partial}{\partial y} (-\log(1-x-y) - \log x - \log y \end{bmatrix}$$

$$= \begin{bmatrix} \frac{-1}{(1-x-y)} \cdot (0-1-0) - \frac{1}{x} - 0 \\ \frac{-1}{(1-x-y)} \cdot (0-0-1) - 0 - \frac{1}{y} \end{bmatrix}$$

$$\nabla f(x,y) = \begin{bmatrix} \frac{1}{(1-x-y)} - \frac{1}{x} \\ \frac{1}{(1-x-y)} - \frac{1}{y} \end{bmatrix}$$

Hessian of $f(x,y)$

$$\nabla^2 f(x,y) = \begin{bmatrix} \frac{\partial^2}{\partial x^2} f(x,y) & \frac{\partial^2}{\partial x \partial y} f(x,y) \\ \frac{\partial^2}{\partial y \partial x} f(x,y) & \frac{\partial^2}{\partial y^2} f(x,y) \end{bmatrix}$$

$$\frac{\partial^2}{\partial x^2} f(x,y) = \frac{\partial^2}{\partial x^2}\left(-\log(1-x-y) - \log x - \log y\right)$$

$$= \frac{\partial}{\partial x}\left(\frac{-1}{(1-x-y)}\cdot(0-1-0) - \frac{1}{x} - 0\right)$$

$$= \frac{\partial}{\partial x}\left(\frac{1}{(1-x-y)} - \frac{1}{x}\right)$$

$$= \frac{-1}{(1-x-y)^2}\cdot(0-1-0) + \frac{1}{x^2}$$

$$= \frac{1}{(1-x-y)^2} + \frac{1}{x^2}$$

$$\frac{\partial^2}{\partial x \partial y} f(x,y) = \frac{\partial^2}{\partial x \partial y}\left(-\log(1-x-y) - \log x - \log y\right)$$

$$= \frac{\partial}{\partial x}\left(\frac{-1}{(1-x-y)}\cdot(0-0-1) - 0 - \frac{1}{y}\right)$$

$$= \frac{\partial}{\partial x}\left(\frac{1}{(1-x-y)} - \frac{1}{y}\right)$$

$$= \frac{-1}{(1-x-y)^2}\cdot(0-1-0) - 0$$

$$= \frac{1}{(1-x-y)^2} = \frac{\partial^2}{\partial y \partial x} f(x,y)$$

$$\frac{\partial^2}{\partial y^2} f(x,y) = \frac{\partial^2}{\partial y^2}\left(-\log(1-x-y) - \log x - \log y\right)$$

$$= \frac{\partial}{\partial y}\left(\frac{-1}{(1-x-y)}\cdot(0-0-1) - 0 - \frac{1}{y}\right)$$

$$= \frac{\partial}{\partial y}\left(\frac{1}{(1-x-y)} - \frac{1}{y}\right)$$

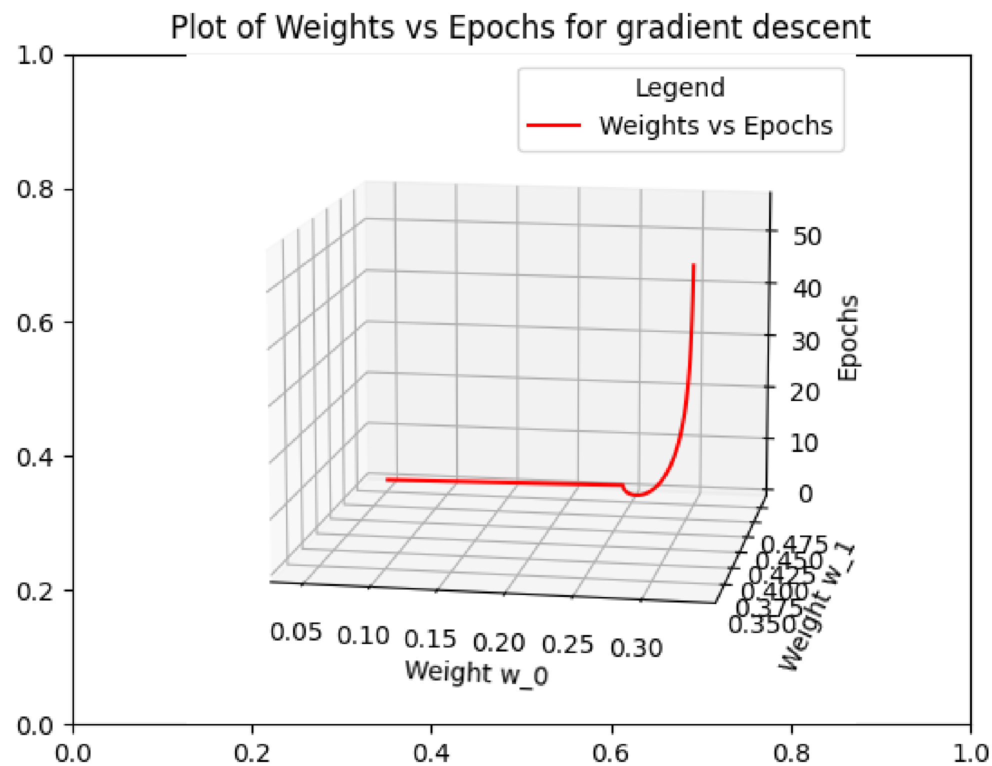
$$= \frac{-1}{(1-x-y)^2}\cdot(0-0-1) + \frac{1}{y^2}$$

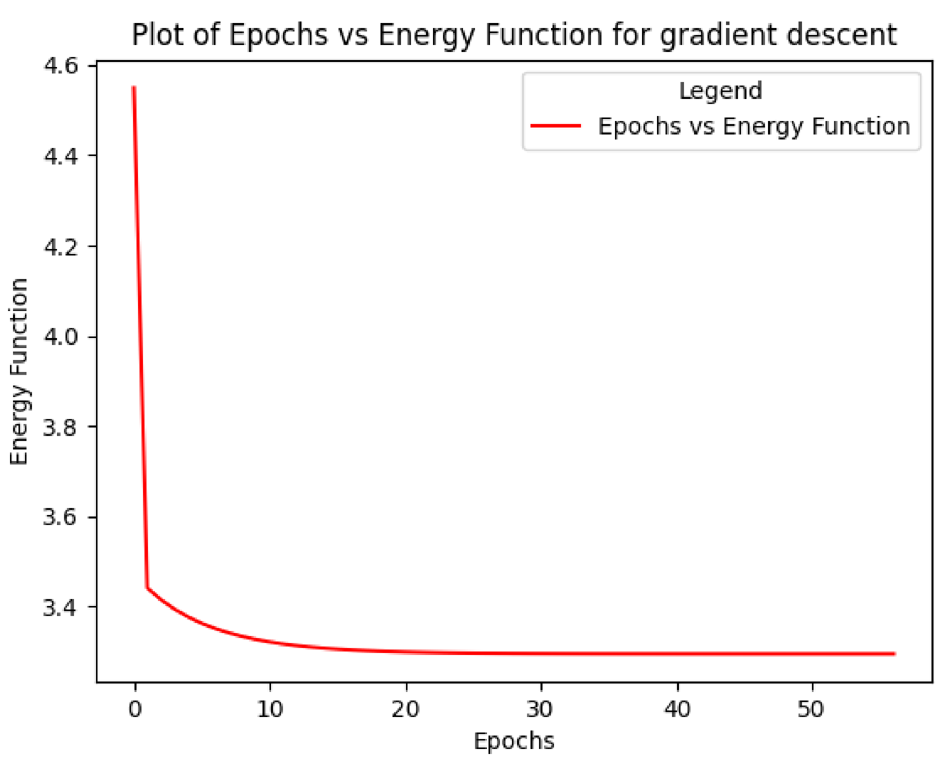
$$= \frac{1}{(1-x-y)^2} + \frac{1}{y^2}$$

$$H = \nabla^2 f(x,y) = \begin{bmatrix} \frac{1}{(1-x-y)^2} + \frac{1}{x^2} & \frac{1}{(1-x-y)^2} \\ \frac{1}{(1-x-y)^2} & \frac{1}{(1-x-y)^2} + \frac{1}{y^2} \end{bmatrix}$$

**(b) Gradient descent**
Initial weights w0:
[[0.04655414]
 [0.48582796]]

Eta = 1.0



With eta = 1.0, weights have jumped out of the domain where the function is defined and eta was divided by 10 when this happens. Final eta after the training is done = 0.01.

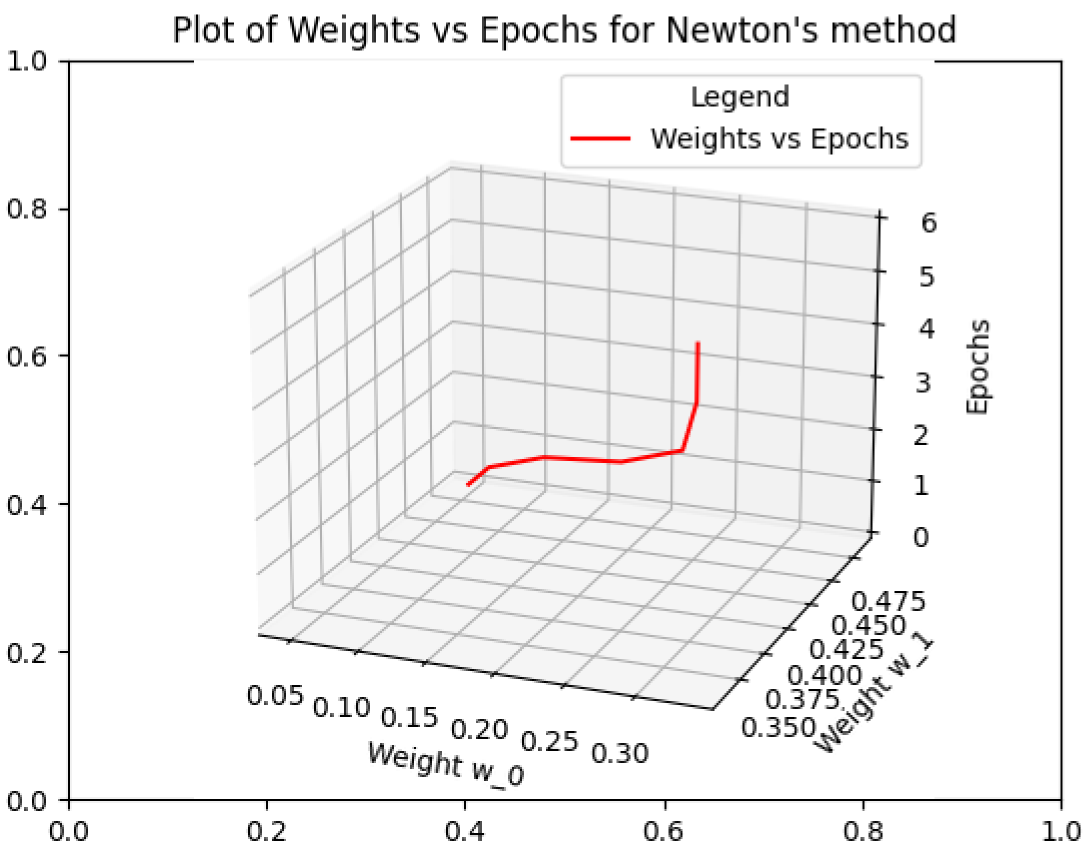
Epochs taken for convergence: 56

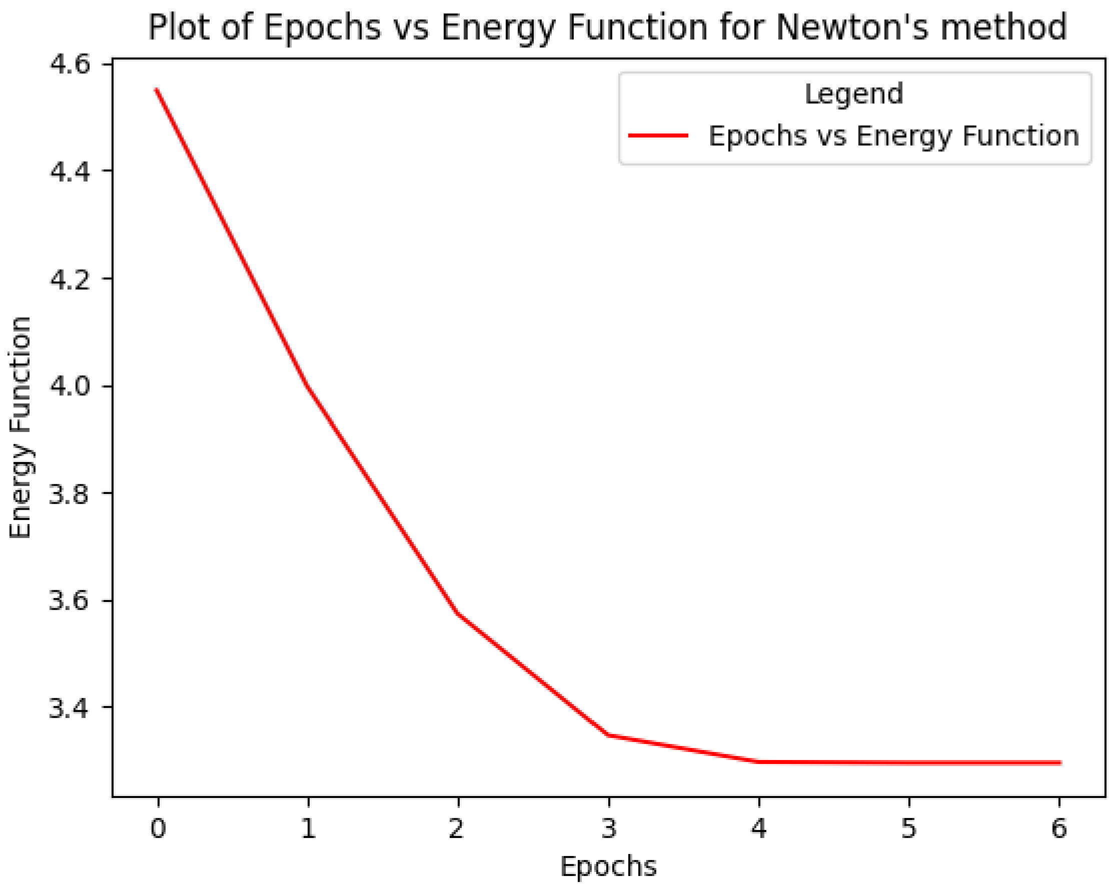
Final weights:
[[0.33262939]
 [0.33404001]]

Final energy function: 3.295841343235882

**(c) Newton's method:**
Initial weights w0:
[[0.04655414]
 [0.48582796]]

Eta = 1.0






Epochs taken for convergence:  6

Final weights:
[[0.33333313]
 [0.33333343]]

Final energy function:  3.2958368660045982

**(d)**
When trained with Newton’s method, the model converges faster with 6 iterations when compared to the one trained with gradient descent(56 iterations). This is because, though both models began training with eta = 1.0, weights in gradient descent jumped out of the domain due to which eta had to be changed to 0.01. But with Newton's method the model converged fine with eta = 1.0 because of the additional Hessian matrix.

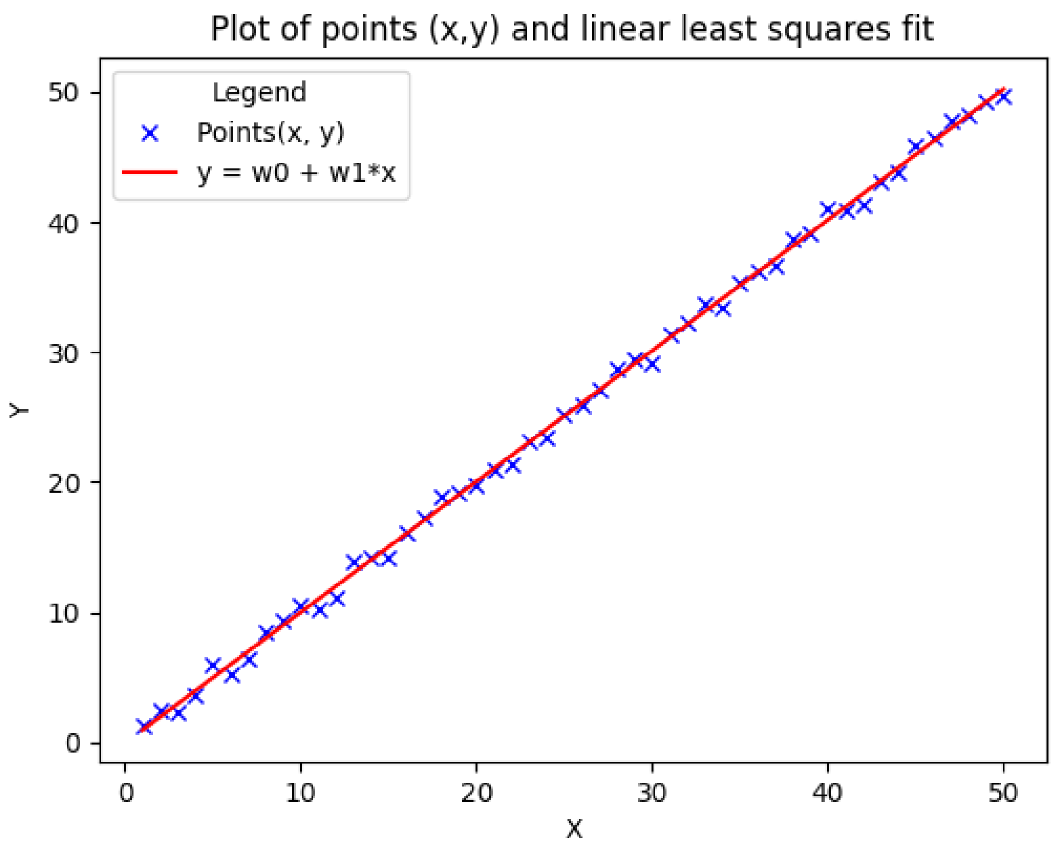
**2)**
**(c)**
Linear least squares fit:
Final weights obtained:
[[-0.07554311]
 [ 1.00569777]]

Final energy function:  3.771377183705698

Linear least squares fit is:
y = -0.07554311 + 1.00569777*x

**(d)**

Plot of points (x, y) and line y = -0.07554311 + 1.00569777*x




**(e)**

2)

(e) gradient of $\sum_{i=1}^{50} (y_i - (w_0 + w_1 x_i))^2$

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial w_0} \left( \sum_{i=1}^{50} (y_i - (w_0 + w_1 x_i))^2 \right) \\ \frac{\partial}{\partial w_1} \left( \sum_{i=1}^{50} (y_i - (w_0 + w_1 x_i))^2 \right) \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{50} \frac{\partial}{\partial w_0} (y_i - (w_0 + w_1 x_i))^2 \\ \sum_{i=1}^{50} \frac{\partial}{\partial w_1} (y_i - (w_0 + w_1 x_i))^2 \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{50} 2 \cdot (y_i - (w_0 + w_1 x_i)) \cdot (0 - 1 - 0) \\ \sum_{i=1}^{50} 2 \cdot (y_i - (w_0 + w_1 x_i)) \cdot (0 - 0 - x_i) \end{bmatrix}$$

$$= -2 \cdot \sum_{i=1}^{50} \begin{bmatrix} (y_i - (w_0 + w_1 x_i)) \\ (y_i - (w_0 + w_1 x_i)) x_i \end{bmatrix}$$

$$\nabla = -2 \cdot \sum_{i=1}^{50} (y_i - (w_0 + w_1 x_i)) \cdot \begin{bmatrix} 1 \\ x_i \end{bmatrix}$$

**(f)**

Linear least Squares fit using Gradient descent with eta = 0.0001 (since the weights are overflowing with higher eta values):

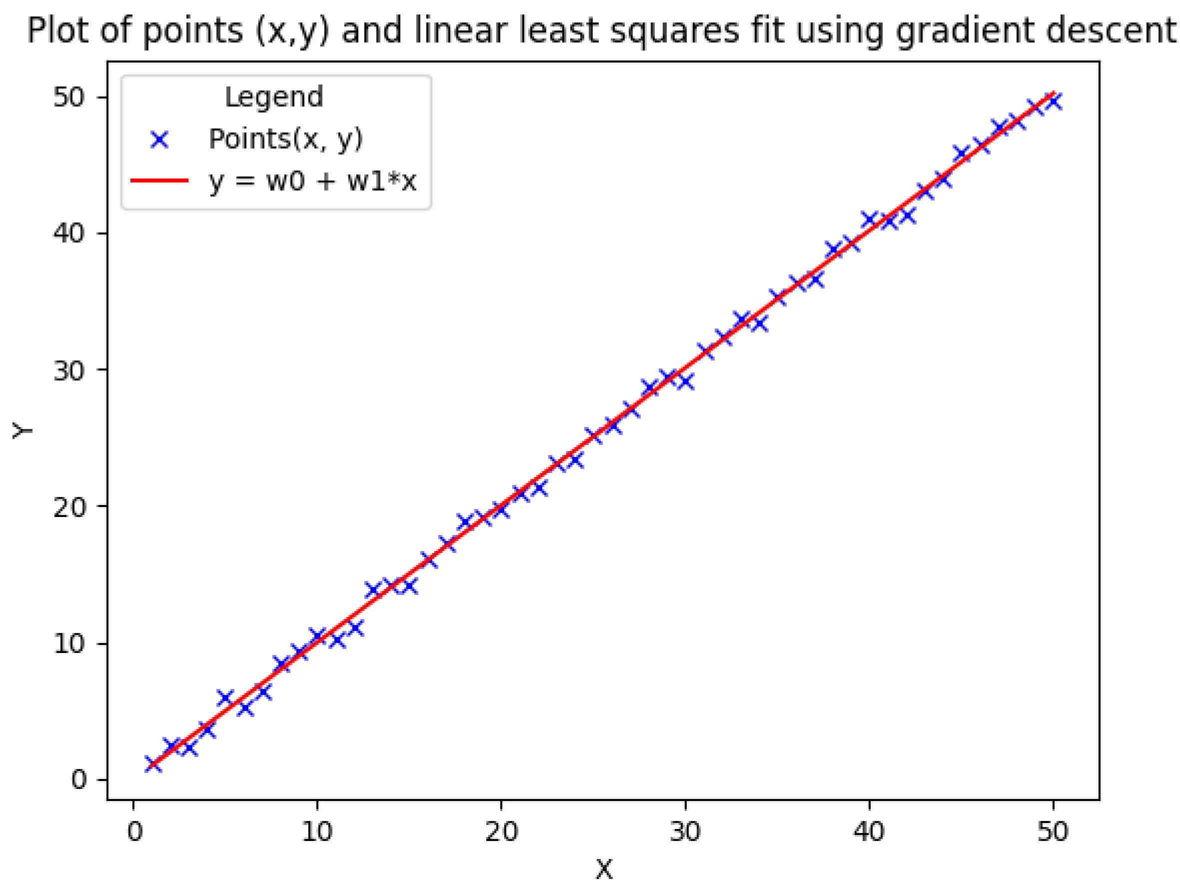
Gradient descent converged in 2804 epochs

Weights obtained for linear least squares using gradient descent:
[[-0.06842625]
 [ 1.00475665]]

Final Energy function:  3.774487730323436

Linear least squares fit is:
y = -0.06842625 + 1.00475665*x

Plot of points (x, y) and line y = -0.06842625 + 1.00475665*x




The weights and the final cost function obtained in both cases (f) and (c) are almost the same. But gradient descent took about 2804 iterations for convergence.

**Python code (Q1):**

```
# Name: Sai Anish Garapati
# UIN: 650208577

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(55)

# E = -log(1 - x - y) - logx - logy
def energy_function(w):
    return float(-np.log(1 - w[0] - w[1]) - np.log(w[0]) - np.log(w[1]))

def gradient(w):
    return 1/(1 - w[0] - w[1]) + np.array([-1/(w[0]), -1/(w[1])])

def hessian(w):
    return 1/((1 - w[0] - w[1])**2) + np.array([[float(1/((w[0])**2)), 0.0], [0.0,
float(1/((w[1])**2))]])

def gradient_descent(W, eta, epsilon):
    print('Gradient Descent with eta = {}, epsilon = {}: \n'.format(eta, epsilon))
    E = []

    while (True):
        E.append(energy_function(W[-1]))

        # Converging criteria: When Energy function change is less than 1e-6
        if (len(E) > 1 and abs(E[-1] - E[-2]) <= epsilon):
            break

        g = gradient(W[-1])

        w_new = W[-1] - (eta * g)

        # Taking care of the model when weights go beyond the domain
        while (w_new[0] <= 0 or w_new[1] <= 0 or (w_new[0] + w_new[1] >= 1)):
            eta /= 10.0
            print('eta value changed to {}\n'.format(eta))
            w_new = W[-1] - (eta * g)

        W.append(w_new)

    print('Epochs taken for convergence: ', len(W) - 1, '\n')
    print('Initial weights: ', W[0],'\n')
    print('Final weights: ', W[-1], '\n')
    print('Final energy function: ', E[-1], '\n')

    plt.title('Plot of Weights vs Epochs for gradient descent')
    ax = plt.axes(projection="3d")
    ax.set_xlabel('Weight w_0')
    ax.set_ylabel('Weight w_1')
```

```
    ax.set_zlabel('Epochs')
    ax.plot3D([float(x[0]) for x in W], [float(x[1]) for x in W], [i for i in range(0,
len(W))], 'r', label='Weights vs Epochs')
    plt.legend(title='Legend')
    plt.show()

    plt.title('Plot of Epochs vs Energy Function for gradient descent')
    plt.xlabel('Epochs')
    plt.ylabel('Energy Function')
    plt.plot([x for x in range(0, len(W))], E, 'r', label='Epochs vs Energy Function')
    plt.legend(title='Legend')
    plt.show()

def newton_method(W, eta, epsilon):
    print('Newton\'s method with eta = {}, epsilon = {}: \n'.format(eta, epsilon))
    E = []

    while (True):
        E.append(energy_function(W[-1]))

        # Converging criteria: When Energy function change is less than 1e-6
        if (len(E) > 1 and abs(E[-1] - E[-2]) <= epsilon):
            break

        g = gradient(W[-1])
        H = hessian(W[-1])

        w_new = W[-1] - (eta * np.dot(np.linalg.inv(H), g))

        # Taking care of the case when weights go beyond the defined domain
        while (w_new[0] <= 0 or w_new[1] <= 0 or (w_new[0] + w_new[1] >= 1)):
            eta /= 10.0
            print('eta value changed to {}'.format(eta))
            w_new = W[-1] - (eta * np.dot(np.linalg.inv(H), g))

        W.append(w_new)

    print('Epochs taken for convergence: ', len(W) - 1, '\n')
    print('Initial weights: ', W[0], '\n')
    print('Final weights: ', W[-1], '\n')
    print('Final energy function: ', E[-1], '\n')

    plt.title('Plot of Weights vs Epochs for Newton\'s method')
    ax = plt.axes(projection="3d")
    ax.set_xlabel('Weight w_0')
    ax.set_ylabel('Weight w_1')
    ax.set_zlabel('Epochs')
    ax.plot3D([float(x[0]) for x in W], [float(x[1]) for x in W], [i for i in range(0,
len(W))], 'r', label='Weights vs Epochs')
    plt.legend(title='Legend')
    plt.show()

    plt.title('Plot of Epochs vs Energy Function for Newton\'s method')
```

```
    plt.xlabel('Epochs')
    plt.ylabel('Energy Function')
    plt.plot([x for x in range(0, len(W))], E, 'r', label='Epochs vs Energy Function')
    plt.legend(title='Legend')
    plt.show()

if __name__ == '__main__':
    W1 = []
    W1.append(np.random.rand(2, 1)/2)
    eta = 1.0
    epsilon = 1e-6
    gradient_descent(W1, eta, epsilon)

    # Training with Newton's method with the same initial weights as gradient descent
    W2 = []
    W2.append(W1[0])
    newton_method(W2, eta, epsilon)
```

**Python code (Q2):**

```
# Name: Sai Anish Garapati
# UIN: 650208577

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2021)

# Using transpose(w) = D.transpose(X).inv(X.transpose(X)) since transpose(X).X is a singular
matrix
def linear_least_square(D, X):
    return np.transpose(np.dot(D, np.dot(np.transpose(X), np.linalg.inv(np.dot(X,
np.transpose(X))))))

def energy_function(D, X, W):
    return np.linalg.norm(D - np.dot(X, W), 2)

def gradient_descent(D, X, eta, epsilon):
    W = np.random.randn(2, 1)
    E = []
    epoch = 0
    while(True):
        E.append(energy_function(D, X, W))

        # Convergence criteria: When the change in cost function is less than 1e-6
        if (len(E) > 1):
            if (abs(E[-1] - E[-2]) <= epsilon):
                break
        for x, d in zip(X, D):
            W = W + eta * np.dot(x.reshape(2, 1), (d - np.dot(np.transpose(W), x.reshape(2,
1))))
        epoch += 1
```

```
    print('Gradient descent converged in {} epochs\n'.format(epoch))
    print('Final Energy function: ', E[-1], '\n')
    return W

if __name__ == '__main__':
    X = np.array([list(range(1, 51))]).reshape(50, 1)
    X = np.concatenate((np.ones(shape=(50, 1)), X), axis=1)
    Y = np.array([x[1] + np.random.uniform(-1, 1) for x in X]).reshape(50, 1)

    # Linear least squares fit
    print('Linear Least Squares fit: \n')
    W_o = linear_least_square(np.transpose(Y), np.transpose(X))
    print('Weights obtained for linear least squares fit: ', W_o, '\n')
    print('Final energy function: ', energy_function(Y, X, W_o), '\n')

    plt.title('Plot of points (x,y) and linear least squares fit')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.plot([x[1] for x in X], Y, 'bx', label='Points(x, y)')
    x_plot = np.linspace(1, 50)
    y_plot = W_o[0] + W_o[1] * x_plot
    plt.plot(x_plot, y_plot, 'r', label='y = w0 + w1*x')
    plt.legend(title='Legend')
    plt.show()

    # Linear least squares fit using gradient descent
    eta = 0.0001
    epsilon = 1e-6
    print('Linear least Squares fit using Gradient descent with eta = {}, epsilon = 
{}:\n'.format(eta, epsilon))
    W_go = gradient_descent(Y, X, eta, epsilon)
    print('Weights obtained for linear least squares using gradient descent: ', W_go, '\n')

    plt.title('Plot of points (x,y) and linear least squares fit using gradient descent')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.plot([x[1] for x in X], Y, 'bx', label='Points(x, y)')
    x_plot = np.linspace(1, 50)
    y_plot = W_go[0] + W_go[1] * x_plot
    plt.plot(x_plot, y_plot, 'r', label='y = w0 + w1*x')
    plt.legend(title='Legend')
    plt.show()
```