CS 559 - Fall 2021
Homework #4

Krishna Garg
kgarg8@uic.edu

(1) **Comparing Gradient Descent and Newton's method**

The objective function is:

$$f(x,y) = -log(1-x-y) - logx - logy$$

with domain D = $(x,y) : x+y, x+y < 1, x > 0, y > 0$

GRADIENT CALCULATIONS:

$$\begin{aligned} d_x f = \frac{\partial f(x,y)}{\partial x} &= -\frac{1}{1-x-y} * (-1) - \frac{1}{x} \\ &= \frac{1}{1-x-y} - \frac{1}{x} \\ d_y f = \frac{\partial f(x,y)}{\partial y} &= -\frac{1}{1-x-y} * (-1) - \frac{1}{y} \\ &= \frac{1}{1-x-y} - \frac{1}{y} \end{aligned}$$

$$G = (d_x f \quad d_y f) = \left(\tfrac{1}{1-x-y} - \tfrac{1}{x} \quad \tfrac{1}{1-x-y} - \tfrac{1}{y}\right)$$

HESSIAN CALCULATIONS

$$H = \begin{pmatrix} d_{xx} f & d_{xy} f \\ d_{yx} f & d_{yy} f \end{pmatrix}$$

$$\begin{aligned} d_{xx} f = \frac{\partial (d_x f)}{\partial x} &= -\frac{1}{(1-x-y)^2} * (-1) + \frac{1}{x^2} = \frac{1}{(1-x-y)^2} + \frac{1}{x^2} \\ d_{xy} f = \frac{\partial (d_x f)}{\partial y} &= -\frac{1}{(1-x-y)^2} * (-1) + 0 = \frac{1}{(1-x-y)^2} \\ d_{yx} f = \frac{\partial (d_y f)}{\partial x} &= -\frac{1}{(1-x-y)^2} * (-1) + 0 = \frac{1}{(1-x-y)^2} \\ d_{yy} f = \frac{\partial (d_y f)}{\partial y} &= -\frac{1}{(1-x-y)^2} * (-1) + \frac{1}{y^2} = frac1(1-x-y)^2 + \frac{1}{y^2} \end{aligned}$$

$$H = \begin{pmatrix} \frac{1}{(1-x-y)^2} + \frac{1}{x^2} & \frac{1}{(1-x-y)^2} \\ \frac{1}{(1-x-y)^2} & \frac{1}{(1-x-y)^2} + \frac{1}{y^2} \end{pmatrix}$$

The initial point w_0 is (0.19, 0.02) and the learning rate η is 0.02.

Results: Refer Fig. (1, 2)

Generally, the notion is that Gradient Descent can go zigzag in the trajectory before it converges to the global minimum and may take a longer time to converge than the Newton method.

But for the given objective function, we can observe that Gradient Descent (GD) does not depict the zigzag behavior and rather it converges much much faster than Newton Method. For GD, it takes around 10 iterations to converge whereas it takes around 400 iterations for the Newton's method to converge. We observe the same phenomenon with a different learning rate also.

(2) **Linear Least Squares Fit**

CLOSED FORM SOLUTION

For the equation $y = w_1 x + w_0$, we can find w_1 and w_0 using the following equations:

$$w_1 = \frac{N \sum(xy) - \sum x . \sum y}{N \sum x^2 - (\sum x)^2}$$
$$w_0 = \frac{\sum y - N \sum x}{N}$$

GRADIENT DESCENT METHOD

Energy function is given by:

$$f(w_1, w_0) = \sum_{i=1}^{50} (y_i - (w_1 x_i + w_0))^2$$

The partial derivatives are given by:

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

The weights are updated according to the following equations:

$$w_1 = w_1 - \eta * \frac{\partial f}{\partial w_1}$$
$$w_0 = w_0 - \eta * \frac{\partial f}{\partial w_0}$$

Results: Refer Fig. (3)

$w_1 = 1.0124815602408974, w_0 = -0.2820802313809418$

$w_1 = 0.9946671535014963, w_0 = 0.3175021482094951$

From the plot [3], we can observe that both the above methods could almost fit the datapoints and the line curves obtained using the two methods are almost indistinguishable. The weights mentioned above are almost equal in their w_1 parameter (which is the slope of the line) and there is only a slight difference in the w_0 (which is the intercept) and therefore, the line curves almost overlap.

In conclusion, both methods work fine in the given case where the objective function is bit simple and the number of datapoints are very less and the problem is univariate regression. In more complex cases, particularly when the objective function is non-convex, the closed form solution may not be available and in some other cases where the problem is multivariate with large number of datapoints, the closed form solution again may require tremendous mathematical calculations. In all those cases, gradient descent proves a better strategy due to its simple iterative way of approximating the weights.

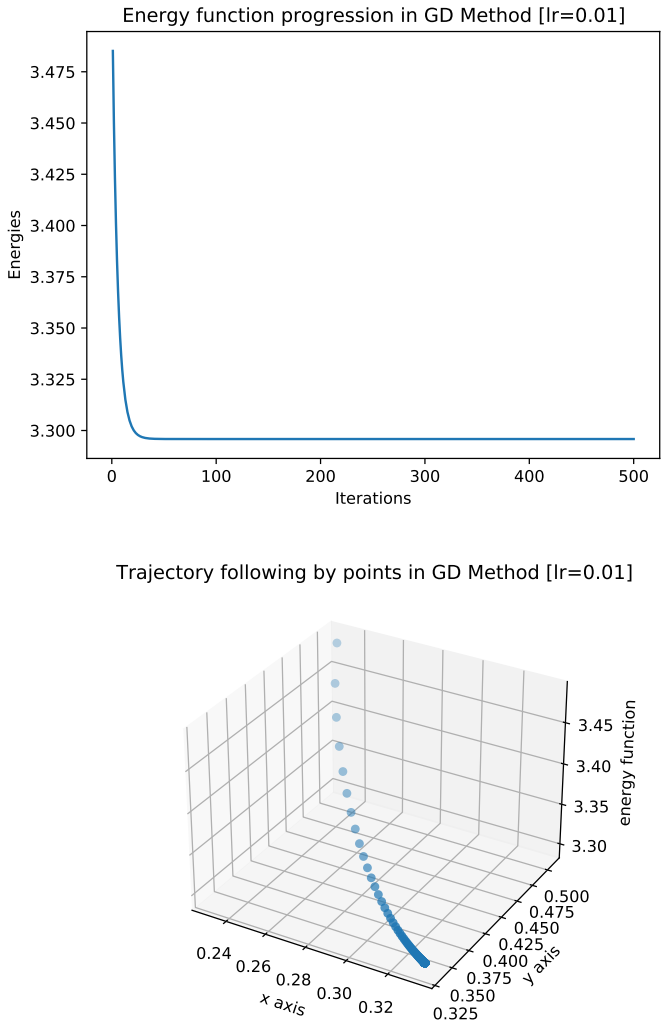
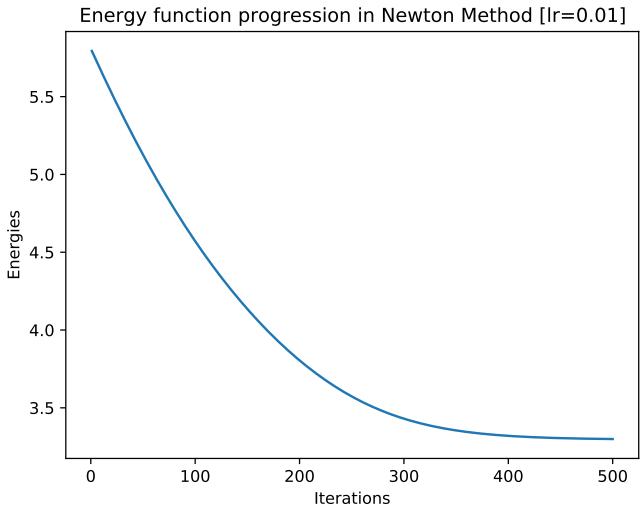


Figure 1: Gradient Descent Method



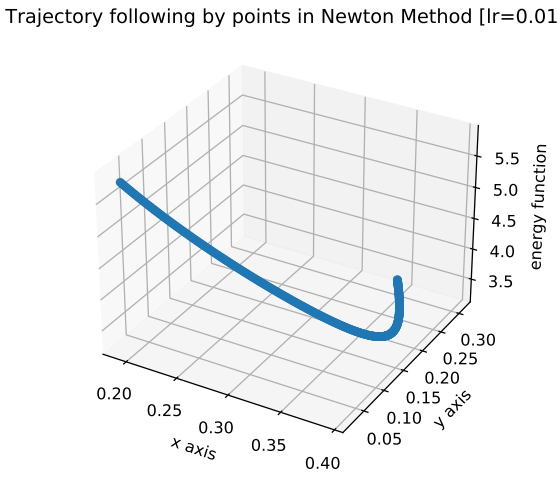


Figure 2: Newton's Method

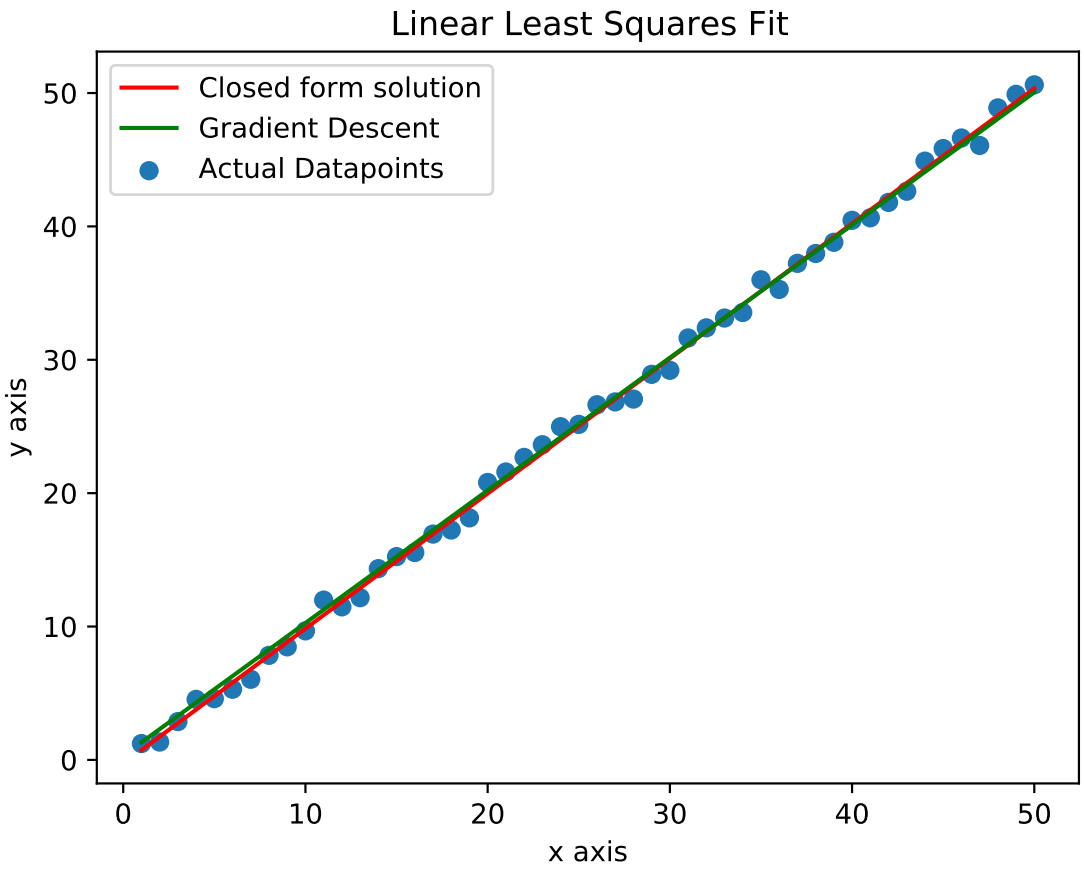


Figure 3: Least Squares Fit using usingusing (a) Closed form solution, (b) Gradient Descent

Code for (Q1)

```
import math, numpy as np, matplotlib.pyplot as plt

def objective_function(x, y):
   return -math.log(1 - x - y) - math.log(x) - math.log(y)

def gradient_function(x, y):
   return [(1/ (1 - x - y)) - 1/x, (1/ (1 - x - y)) - 1/y]

def hessian_function(x, y):
   return [[1/(x*x) + 1/(1-x-y)**2, 1/(1-x-y)**2], [1/(1-x-y)**2,
       1/(1-x-y)**2 + 1/(y*y)]]

x_0, y_0 = 0.19, 0.02 # Initial point
method = 'GD' # or 'Newton'

for eta in [1e-2]:
   x, y, E = [], [], []
   x_i = x_0; y_i = y_0

   iterations = 0
   while iterations < 500:
      g = gradient_function(x_i, y_i)

      if method == 'GD': # Gradient Descent
         delta = g
      else:           # Newton's Method
         H = hessian_function(x_i, y_i)
         H_inv = np.linalg.inv(np.array(H))
         delta = H_inv @ np.array(g)

      # update
      x_i = x_i - eta * delta[0]
      y_i = y_i - eta * delta[1]

      try:
         E_i = objective_function(x_i, y_i)
      except ValueError:
         E_i = 0

      E.append(E_i)
      x.append(x_i)
      y.append(y_i)

      iterations +=1

   # Plot Energy function
   epoch_arr = list(range(1, iterations+1))
   plt.figure()
   plt.xlabel('Iterations')
   plt.ylabel('Energies')
   plt.plot(epoch_arr, E)
   plt.title('Energy function progression in {} Method
       [lr={}]'.format(method, eta))
   plt.savefig('{}_Energy_x_{}_y_{}_eta_{}.pdf'.format(method, x_0, y_0,
       eta))

   # Plot Trajectory
   fig = plt.figure()
   ax = fig.add_subplot(111, projection='3d')
   ax.set_ylabel('y axis')
   ax.set_xlabel('x axis')
   ax.set_zlabel('energy function')
```

```
    plt.title('Trajectory following by points in {} Method
        [lr={}]'.format(method, eta))
    ax.scatter(x, y, E)
    plt.savefig('{}_Trajectory_x_{}_y_{}_eta_{}.pdf'.format(method, x_0,
        y_0, eta))
```