Math 6019 Assignment 2.

Problem 1:

Implement Powell method for minimizing a multivariate objective function.

The function is defined in the python file named *PowellMethod.py.* This script contains three functions viz ‘powell_method’, ‘powell_surfaceplot’ and ‘powell_contourplot’ to minimize a multivariate objective function given an initial guess of size Nx1 array and to plot the multivariate objective function along with the local minimum respectively.

The function ‘powell_method’ takes the following input values:

INPUT:

F : It is the objective function

x0 : It is an array of size Nx1 initial guesses required for Powell's method

Lower: It is an array of size Nx1 lower bound values

Upper: It is an array of size Nx1 upper bound values

tol : This is the tolerance accepted for the minimized objective function (by default tol = 1e-8)

maxiter : Maximum iterations to stop the optimization (by default maxiter = 1e3)

OUTPUT:

1. Returns the values of minimum 'X_k' and total number of iterations ‘k’

The ‘powell_surfaceplot’ and ‘powell_contourplot’ returns a figure each in a new window showing the plot of the objective function and the local minimum for the given objective function.

Running the script:

1. At the end of the script, I am defining an equation in ‘x’ using python’s anonymous function and storing it in the variable ‘Func’.
2. Inputting the value for initial guess, lower bound and upper bound values and assigning them to the variables ‘X0’, ‘L’ and ‘U’ respectively (**The input values are Nx1 array**).
3. Calling the function ‘powell_method’ and passing the values, ‘Func’ and ‘X0’ to scan the values containing the local minimum of the given objective function.
4. Calling the function ‘powell_surfaceplot’ and ‘powell_contourplot’ and passing the values ‘Func’ and 'X_k' to show the figure containing objective function plot and the local minimum of the objective function.
5. Displaying the minimum values ‘X_k’ and the number of iterations ‘k’ to the console.

The python code is appropriately commented defining the process.

Some of the compiled results:

i. F(x) = -4x +x^2-y+xy+y^2 and X0 = ([-1.0, 6.8]), L = ([0, 0.]), U = ([10., 10.])

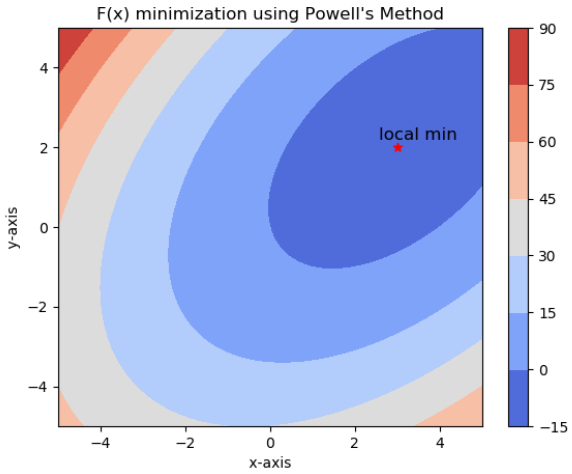

Figure 1: Contour Plot of the multivariate objective function with the local minimum.

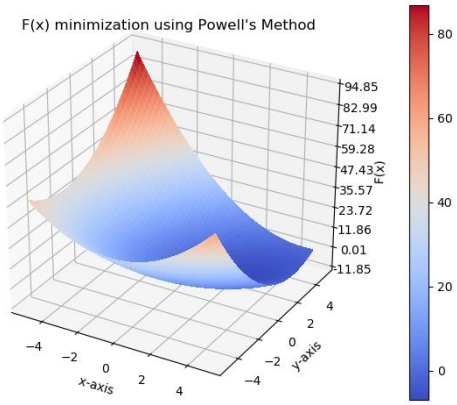

Figure 2: Surface Plot of the multivariate objective function with the local minimum.

Output values:

minimum exists at [3. 2.]

Fmin -7.0

total no. of iterations = 144

Problem 2:

Implement a gradient based method for minimizing a multivariate objective function.

The gradient based method implemented in this case is Quasi-Newton method and function is defined in the python file named ***QuasiNewton.py.*** This script contains three functions viz 'quasi_newton', 'quasi_newton _surfaceplot' and 'quasi_newton_contourplot' to minimize a multivariate objective function given an initial guess of size Nx1 array and to plot the multivariate objective function along with the local minimum respectively.

The function 'quasi_newton' takes the following input values:

INPUT:

F : It is the objective function

x0 : It is an array of size Nx1 initial guesses required for Powell's method

tol : This is the tolerance accepted for the minimized objective function (by default tol = 1e-8)

maxiter : Maximum iterations to stop the optimization (by default maxiter = 1e3)

OUTPUT:

1. Returns the values of minimum 'X_k' and total number of iterations 'k'

The 'quasi_newton _surfaceplot' and 'quasi_newton_contourplot' returns a figure each in a new window showing the plot of the objective function and the local minimum for the given objective function.

Running the script:

1. At the end of the script, I am defining an equation in 'x' using python's anonymous function and storing it in the variable 'Func'.
2. Inputting the value for initial guess values and assigning it to the variable 'X0' (**The input values are Nx1 array**).
3. Calling the function 'quasi_newton' and passing the values, 'Func' and 'X0' to scan the values containing the local minimum of the given objective function.
4. Calling the function 'quasi_newton _surfaceplot' and 'quasi_newton_contourplot' and passing the values 'Func' and 'X_k' to show the figure containing objective function plot and the local minimum of the objective function.
5. Displaying the minimum values 'X_k' and the number of iterations 'k' to the console.

The python code is appropriately commented defining the process.

Some of the compiled results:

ii. $F(x) = -4x + x^2 - y + xy + y^2$ and X0 = ([2.55, 1.75])

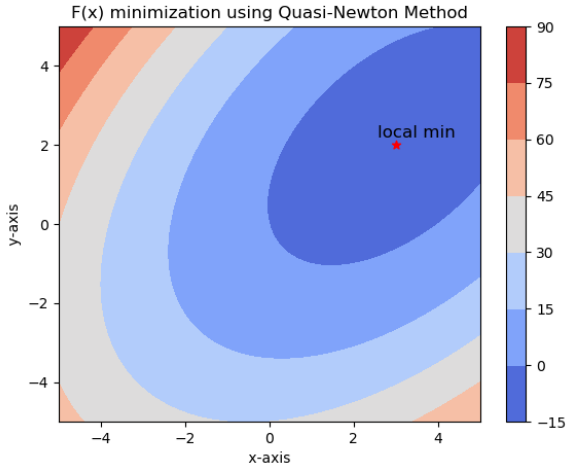

Figure 3: Contour Plot of the multivariate objective function with the local minimum.

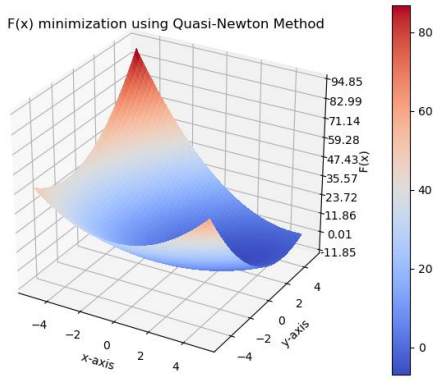

Figure 4: Contour Plot of the multivariate objective function with the local minimum.

Output values:

minimum exists at [3., 2.]

Fmin -7.0

total no. of iterations = 12

Problem 3:

Write a function that implements N-dimensional Rosenbrock Function and test it using Powell and Quasi-Newton method.

The function is defined in the python file named *Rosenbrock.py.* This script contains a function named 'rosenbrock' which takes 'x' as input and return a N-dimensional Rosenbrock Function, given that 'x' has N variables. to find the local minimum of an objective function given an initial guess and to plot the objective function along with the local minimum value respectively.

To test the Rosenbrock function, I am importing functions from both *PowellMethod.py* and *QuasiNewton.py* to minimize the N-dimensional Rosenbrock function.

Some of the compiled results:

1. Minimization of Rosenbrock function of 2 dimensions using Powell's Method and inputs X0 = ([2.55, 1.75]), L = ([-2, -2]), U = ([2, 2])

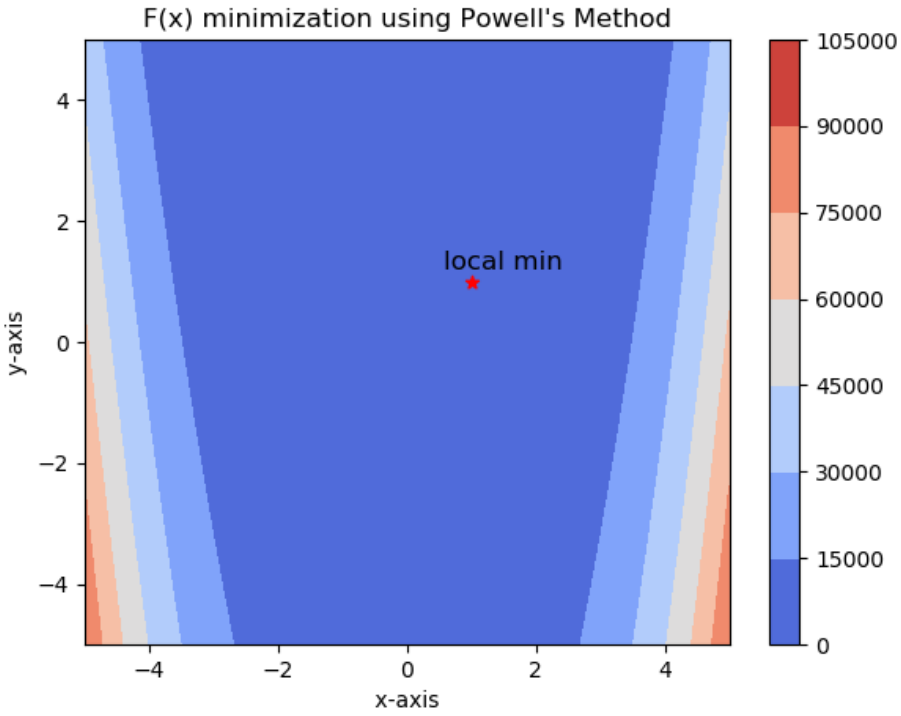

Figure 5: Contour plot of Rosenbrock function minimization using Powell's Method.

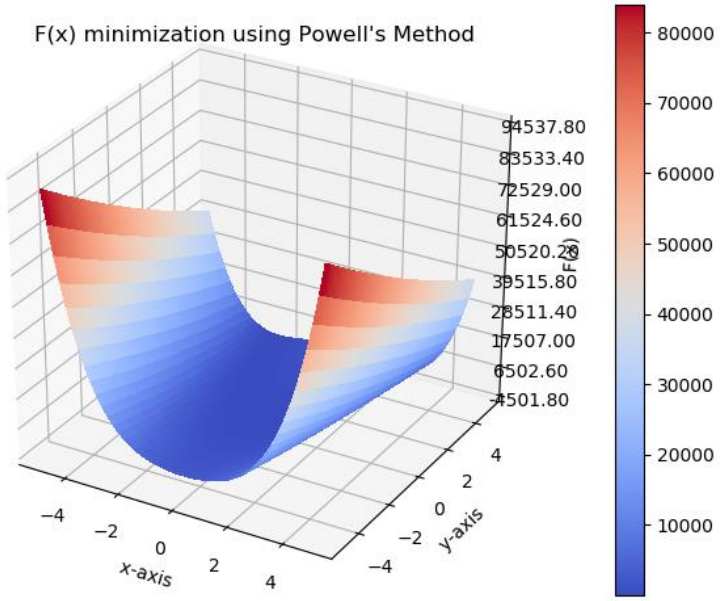

Figure 6: Surface plot of Rosenbrock function minimization using Powell's Method.

Output values:

Powell Method Minimum [1. 1.]

Powell Method Fmin 0.0

Powell Method total no. of iterations = 38799

2. Minimization of Rosenbrock function of 2 dimensions using Quasi-Newton Method and input X0 = ([2.55, 1.75])

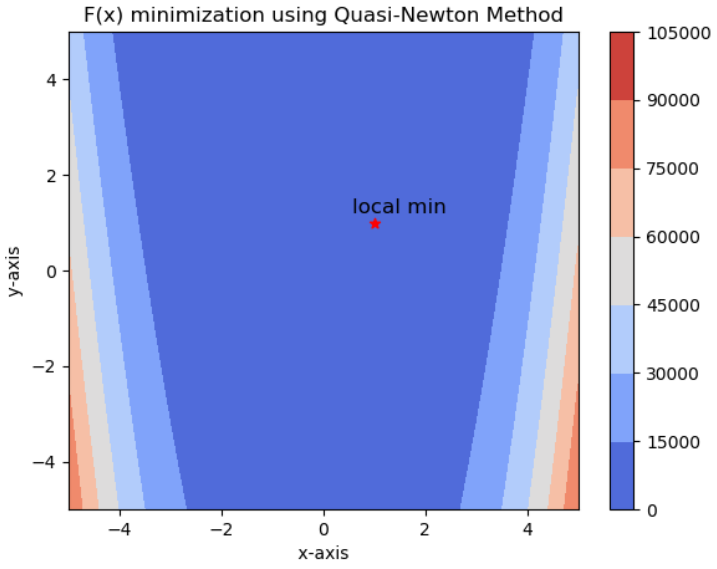

Figure 7: Contour plot of Rosenbrock function minimization using Quasi-Newton Method.

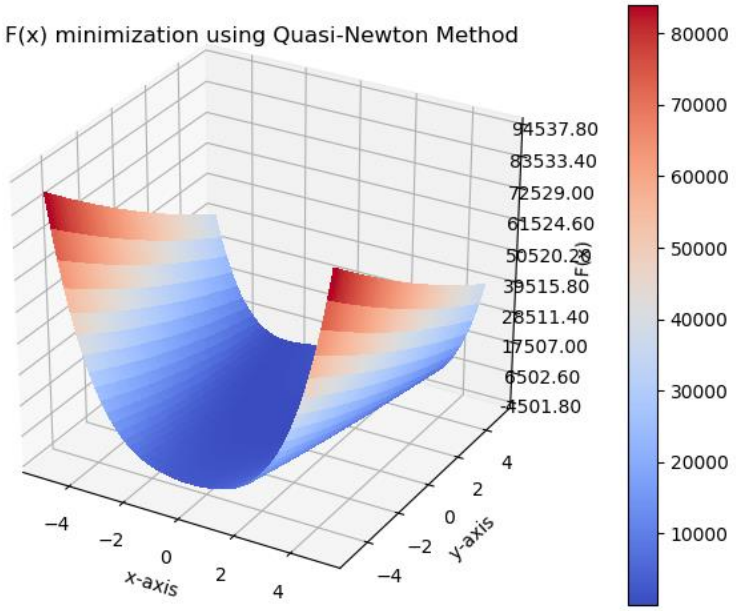

Figure 8: Surface plot of Rosenbrock function minimization using Quasi-Newton Method.

Output values:

Quasi Newton Minimum [1., 1.]

Quasi Newton Fmin 0.0

Quasi Newton total no. of iterations = 36

Problem 4:

Implement a given two-dimensional objective function and determine the number of local minimums and maximums and their locations.

$$F(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

The function is defined in the python file named *HW2_Problem4.py.* This script contains two functions viz 'f (x)'and 'main ()' to implement the given objective function and find local minimums and maximums of the objective function using Quasi-Newton method.

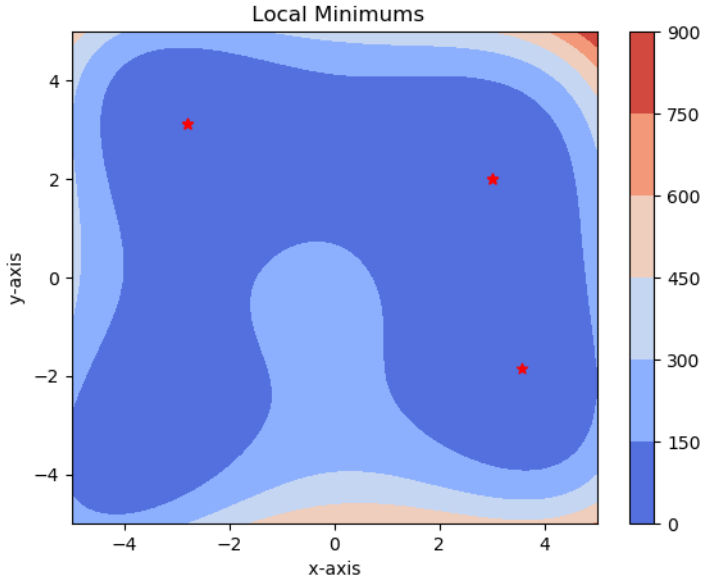

Figure 9: Contour plot of the local minimums of the given objective function

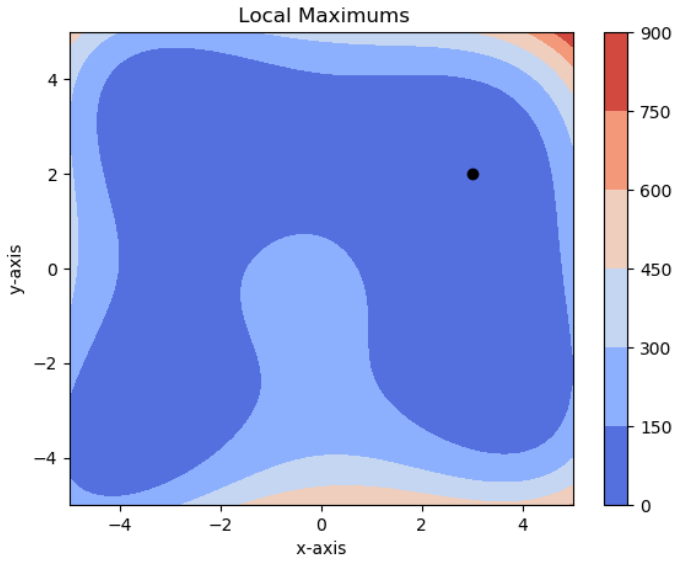

Figure 10: Contour plot of the local maximums of the given objective function

There are 3 local minimums and their locations are at (3, 2), (-2.81, 3.13) and (3.58, -1.85) with F(xmin) being 0, 0.00085 and 0.00113 respectively.

There is only 1 local maximum in the search space and its location is at (3, 2) and F(max) = 0

Problem 5:

Create a function that fits given data to a line through the origin using total least squares.

The function is defined in the python file named ***HW2_Problem5.py.*** This script contains two functions viz 'total_least_squares' and 'prob5_plot' to find the local minimum of an objective function given an initial guess and to plot the objective function along with the local minimum value respectively.

The function 'total_least_squares' takes the following input values:

INPUTS:

1. X0 - It takes an array, which acts as an initial guess to Powell Method
2. m - It takes a float value and it represents the slope of the input line (by default m = 0.5)
3. n - It is the no. of points that are being mapped. It takes an integer value >0 (by default n = 100)
4. d_range - It is the range defined for the graph (by default d_range = (0, 5)
5. eps - represents the magnitude of the noise in the input data (by default eps = 1e-2)

OUTPUTS:

1. Returns the values of minimum 'xn' and total number of iterations 'k'
2. A new window with the plot showing objective function with the minimum point
3. Print the values of total number of iterations 'k', minimum 'x', minimum 'F(x)' to the console

Input values:

'X0' = ([1, 1]), 'slope' = 0.5; 'N' = 100; 'D_range' = (0, 5); = 0.0; 'noise' = 1e-3

Output values:

Minimized 'a' = 0.5; Minimized 'b' = 0.98, Minimized 'F(a)' = 0.0

Best Line Fit plot:

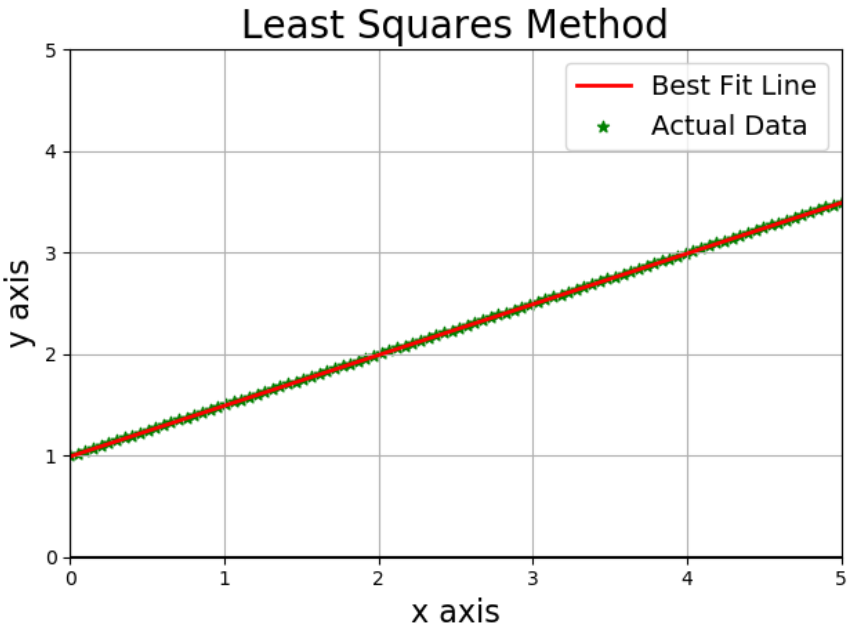

Figure 11: Best Line Fit plot for N=10

Problem 5:

The python script is named *HW2_Problem6.py.*

The collective compiled results for a, b, c for all 3 cases are shown in following figures.

1. P = (5, 3), C = (2, 2), r = 1

Optimal Path for Robot. The robot must travel to the intermediate point I = ([2.51, 1.13]) to avoid the circle while reaching the point P.

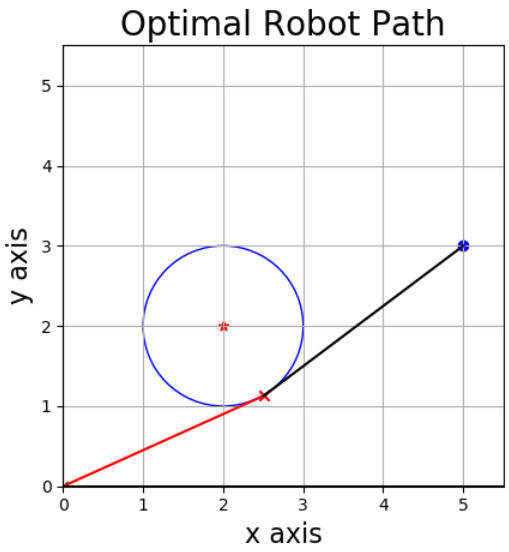

Figure 12: Plot showing the robot path to the given point while avoiding the circle.

2. P = (2, 4), C = (2, 2), r = 1

Optimal Path for Robot: The robot must travel to the intermediate point I = ([1.11, 2.46]) to avoid the circle while reaching the point P.

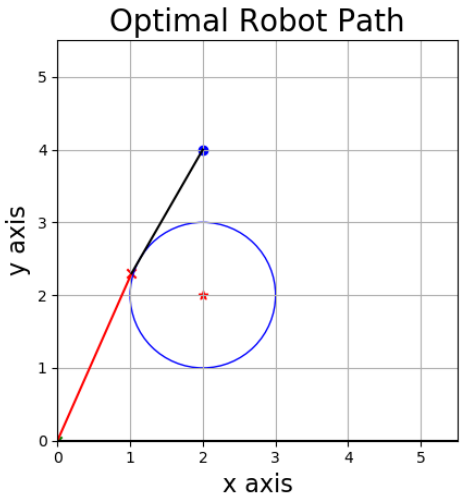

Figure 13: Plot showing the robot path to the given point while avoiding the circle.

3. P = (2, 5), C = (2, 2), r = 1

Optimal Path for Robot: The robot must travel to the intermediate point I = ([3, 1.35]) to avoid the circle while reaching the point P.

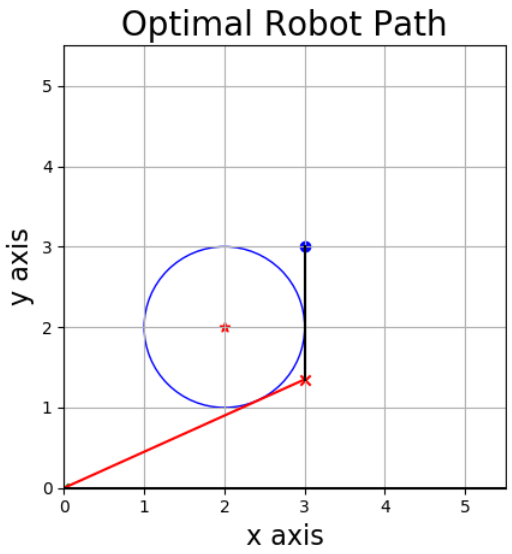

Figure 14: Plot showing the robot path to the given point while avoiding the circle.

Some of the other files in the assignment folder are:

ArmijoLineSearch.py – Weak line search function

BFGS.py – Approximation of the inverse of Hessian called BFGS update

TwoStageRouting.py – Function for the two-stage routing problem

BisectionMethod.py – Used it in Powell method

NewtonRaphson.py

Gradient.py – Calculates gradient of a function of N dimensions