Unconstrained Optimization Problem

Abstract: **Optimization is an important tool in decision science and in the analysis of physical systems. There are several types of these problems in which one is unconstrained optimization problems. In this report, we address optimization of Rosenbrock function through two gradient-based methods: steepest-descent and Newton with backtracking strategy. We also test our methods on several test functions to show the performance.**

I. PROBLEM OVERVIEW:

A. Main Test Function:

In mathematical optimization, the Rosenbrock function is a non-convex function used as a performance test problem for optimization algorithms introduced by Howard H. Rosenbrock in 1960. It is also known as Rosenbrock's valley or Rosenbrock's banana function. The global minimum is inside a long, narrow, parabolic shaped flat valley. It has a global minimum at (x, y) = (1, 1) where f(x, y) = 0. A different coefficient of the second term is sometimes given, but this does not affect the position of the global minimum. The function, gradient and hessian of Rosenbrock (n=2) can be seen in equations below, respectively.

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2 \tag{1}$$

$$\nabla f(x,y) = \begin{pmatrix} 400x^3 - 400xy + 2x^2 - 2 \\ 200(y-x^2) \end{pmatrix} \tag{2}$$

$$Hess\_f(x,y) = \begin{pmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{pmatrix} \tag{3}$$

B. Method Definitions:

- *Steepest Descent Method (SDM):* Let the function $f: R^n \to R$ be given. The steepest descent method is an iterative optimization method that, starting from a given vector $x_0 \in R^n$, computes a sequence of vectors $\{x_k\}_{k \in N}$ characterized by equation 4, where the descent direction p_k is the steepest one, i.e., $p_k = -\nabla f(x_k)$, and the step length factor $\alpha \in R$ is arbitrarily chosen.

$$x_{k+1} = x_k + \alpha p_k, \forall\, k \geq 0 \tag{4}$$

- *Newton Method (NM):* Let the function $f: R^n \to R$ be given. The Newton method is an iterative optimization method that, starting from a given vector $x_0 \in R^n$, computes a sequence of vectors $\{x_k\}_{k \in N}$ characterized by again equation 4, where the descent direction p_k is the solution of the linear system $H_f(x_k)p_k = -\nabla f(x_k)$ and ∇f, H_f are the gradient and the Positive Definite (PD) Hessian matrix of f, respectively.

- *Backtracking strategy:* Let the function $f: R^n \to R$ be given. The backtracking strategy for an iterative optimization method consists of a value α_k satisfying the Armijo condition at each step k of the method, i.e.

$$f(x_{k+1}) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T p_k$$
$$x_{k+1} = x_k + \alpha p_k, \forall\, k \geq 0$$

where $c_1 \in (0,1)$ (typically, the standard choice is $c_1 = 10^{-4}$). The backtracking strategy is an iterative process that looks for the value α_k. Given an arbitrary factor $\rho \in (0,1)$ and an arbitrary starting value $\alpha_k^{(0)}$ for α_k, we decrease iteratively $\alpha_k^{(0)}$, multiplying it by ρ, until the Armijo condition is satisfied. Then $\alpha_k = \rho^{t_k}\alpha_k^{(0)}$, for a $t_k \in N$, if it satisfies Armijo but $\alpha_k = \rho^{t_k - 1}\alpha_k^{(0)}$, does not.

II. IMPLEMENTATION AND PARAMETERS:

To accomplish our methods, we need to define some parameters so that the generalization of the problem is satisfied. The evaluation of the methods has been performed in two starting points $x_0 = (1.2,1.2)$ and $x_0 = (-1.2,1)$. There are two stopping criteria: maximum iteration = 1000 and tolerance = 10^{-12} (this is the threshold for gradient norm). There are also stopping criteria for backtracking strategy: maximum iteration = 50 and Armijo condition.

To do this, we take into account following considerations:

- The Armijo condition is often satisfied for very small values of α. Then, it is not enough to ensure that the algorithm makes reasonable progress; indeed, if α is too small, unacceptably short steps are taken.
- The choice of ${\alpha_k}^{(0)}$ is method-dependent, but it is crucial to choose ${\alpha_k}^{(0)} = 1$ in Newton method for (possibly) get the second-order rate of convergence so we set α =1.
- To select best ρ in terms of generalization for high dimensions problems, we have performed with constant $c_1 = 10^{-4}$ and three values for ρ. As it can be seen in table 1 (with approximation).

ρ	*Starting Points*	*Method*	*CPU time(s)*	*Function_value*	*N. of Iterations*	*BackTrackMaxIter*	*Gradient_norm*
0.3	(1.2,1.2)	SDM	0.0336	0.0014	1000	6	0.0436
		NM	0.0106	3.155e-30	9	1	6.039e-14
	(−1.2,1)	SDM	0.0279	0.0017	1000	6	0.0470
		NM	0.0059	9.984e-31	22	2	1.99e-15
0.5	(1.2,1.2)	SDM	0.0492	0.0001	1000	10	0.1684
		NM	0.0065	2.555e-28	9	1	8.88e-14
	(−1.2,1)	SDM	0.0645	0.0007	1000	10	0.0738
		NM	0.0191	3.728e-29	22	3	1.221e-14
0.8	(1.2,1.2)	SDM	0.0844	0.0031	1000	29	0.1479
		NM	0.0063	3.155e-30	8	2	6.039e-14
	(−1.2,1)	SDM	0.0927	0.0894	1000	31	0.6103
		NM	0.0041	7.888e-31	21	8	1.776e-15

Table 1: Implementation results

III. DISCUSSION:

It can be seen that steepest descent method (SDM) has lower performance in comparison with Newton method (NM) in all cases. In particular, SDM stops due to iteration criteria and never reaches a gradient norm lower than threshold with 1000 iteration, on the other hand, NM stops in few iterations with satisfactory results for function value and gradient norm rather than SDM.

For choosing ρ considering NM, there is a trade-off between number of iterations and CPU time. Although, CPU time is lower for ρ =0.8 but number of iterations is higher, it should be noted that for ρ =0.8, results are slightly better with respect to other points. According to PC configuration limitation and high number of dimensions for test functions, we have chosen ρ =0.8 as our parameter to perform the analysis.

Different starting points have different impact on the performance of each method. Regarding SDM, the computational cost, CPU time and function value are slightly better for $x_0 = (1.2,1.2)$ rather than $x_0 = (-1.2,1)$. On the other hand, NM has better CPU time and function value for $x_0 = (1.2,1.2)$ but higher iterations. The number of backtracking iterations and contour plot showing that how each method reach to minimum points are shown in following figures.

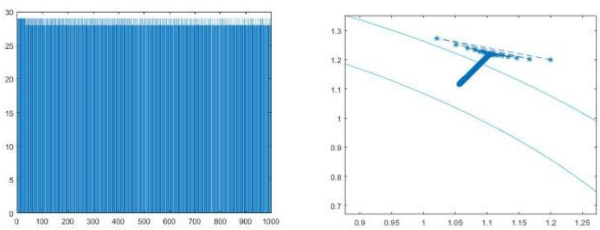

Figure 1: Backtracking iterations and contour plot of SDM for $x_0 = (1.2,1.2)$

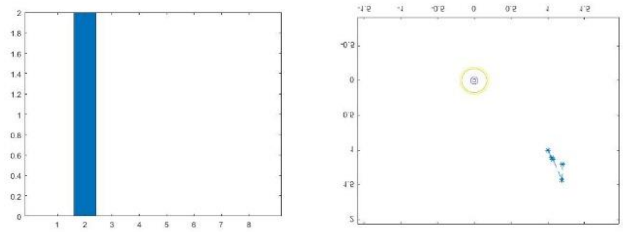

Figure 2: Backtracking iterations and contour plot of NM for $x_0 = (1.2,1.2)$

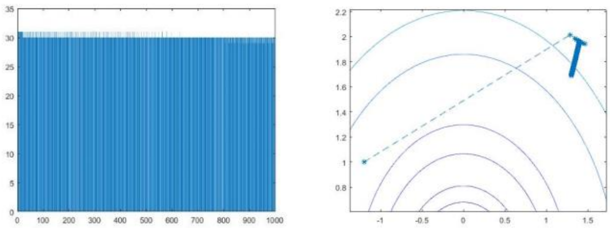

Figure 3: Backtracking iterations and contour plot of SDM for $x_0 = (-1.2,1)$

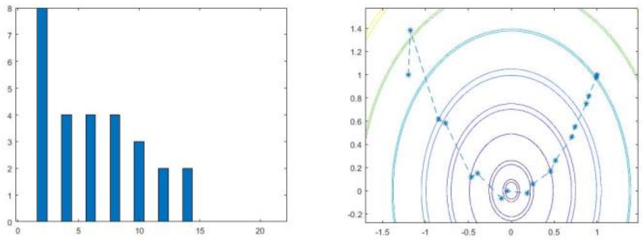

Figure 4: Backtracking iterations and contour plot of NM for $x_0 = (-1.2,1)$

IV. NUMERICAL EXPERIMENTS OF TEST FUNCTIONS:

Numerical experiments are reported in this section in order to evaluate the computational performance of the proposed algorithms for several test functions. In this regard, we have chosen test functions that challenge our methods in different aspects to have various results in terms of convergence, CPU time, iterations and so on.

For this purpose , we have selected test functions that can be evaluated with high dimension and the main function, extended Rosenbrock parabolic valley, have been tested again using large value of the dimension. Three different starting points with dimension n = 10^3 have been chosen to get more insights about performance of the methods: X0=1/3, 1 and -1. Table 2 shows the name of the functions and the results.

Test Function	*Starting Points*	*Method*	*CPU time(s)*	*Function_value*	*N. of Iterations*	*BackTrackMaxIter*	*Gradient_norm*
The extended Rosenbrock parabolic valley	X0 = 1/3	SDM	2.065	4.59	1000	28	7.475
		NM	0.9701	0	13	5	0
	X0 = 1	SDM	0.0034	0	0	0	0
		NM	0.0017	0	0	0	0
	X0 = -1	SDM	2.377	529.0	1000	34	38.05
		NM	1.021	0	20	9	0
The penalty function #1	X0 = 1/3	SDM	0.9938	0.0096	1000	50	2.511e-12
		NM	6.667	0.0096	13	1	3.781e-13
	X0 = 1	SDM	0.1245	0.0103	1000	35	3.195e-05
		NM	7.506	0.0096	15	0	3.683e-15
	X0 = -1	SDM	0.4993	0.0096	1000	50	7.573e-12
		NM	7.553	0.0103	15	0	4.032e-15
The trigonometric function	X0 = 1/3	SDM	8.639	2260	1000	50	1025
		NM	98.39	4.138e-06	1000	50	1.035e-06
	X0 = 1	SDM	8.491	2190	1000	50	1020
		NM	99.89	4.554e-06	1000	50	3.897e-12

	X0 = -1	SDM	8.481	2521	1000	50	1078
		NM	1.645	0	23	0	0
The Hilbert function	X0 = 1/3	SDM	55.89	1.956e-05	1000	5	0.0075
		NM	0.2008	5.624e-11	1	0	3.581e-13
	X0 = 1	SDM	55.47	0.0001	1000	5	0.0226
		NM	29.42	NaN	254	10	NaN
	X0 = -1	SDM	55.51	0.0001	1000	5	0.0226
		NM	29.30	NaN	254	10	NaN
The Gregory and Karney's Tridiagonal Matrix Function	X0 = 1/3	SDM	0.1850	-42.49	1000	4	0.7815
		NM	60.66	-999.9	1000	50	1.006
	X0 = 1	SDM	0.0990	-43.80	1000	4	0.7703
		NM	60.58	-999.6	1000	50	1.110
	X0 = -1	SDM	0.0983	-39.80	1000	4	0.7703
		NM	60.75	-999.6	1000	50	1.108

Table 2: Computation results of test functions

It can be seen that both methods can converge to minimum points in some cases although different starting points and the structure of the functions affect this convergence. For instance, NM converges faster than SDM but it cannot reach to the minimum point for Hilbert function due to the singularity of matrix. We have also the same situation for NM related to the trigonal function but it converges to the points which are far from the minimum point.

V. CONCLUSION:

We have proposed two methods which are based on gradient process for solving non-linear unconstrained problems with backtracking strategy. The optimization of some parameters has been performed by Rosenbrock function with dimension 2 in which the results are satisfactory. Testing our methods on the test function shows that in general, type of the function, starting points, number of dimensions and number of iterations influence the convergence of the methods.

VI. MATLAB CODE:

A. *TEST:*

```
%% LOADING THE VARIABLES FOR THE TEST FOR X0 = [1.2;1.2]

clear
close all
clc

c1 = 1e-4;
rho = 0.8;
btmax = 50;

disp('++++++++++++++++++++++++++ MAIN FUNCTION
++++++++++++++++++++++++++++++++')
load('mytest1.mat')
disp('//////////////////////// FIRST POINT ///////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0 = [1.2;1.2]

disp('**** STEEPEST DESCENT: START WITH X0=[1.2;1.2] *****')
tic
[xk1, fk1, gradfk_norm1, k1, xseq1, btseq1] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0)])
disp(['xk: ', mat2str(xk1), ' (actual minimum: [1; 1]);'])
disp(['f(xk): ', num2str(fk1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq1))])
disp(['grad_norm: ', mat2str(gradfk_norm1)])
disp('***************** DONE *******************')


%% RUN THE NEWTON FOR X0 = [1.2;1.2]

disp('**** NEWTON: START WITH X0=[1.2;1.2] *****')
tic
[xk_n1, fk_n1, gradfk_norm_n1, k_n1, xseq_n1, btseq_n1] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0)])
disp(['xk: ', mat2str(xk_n1), ' (actual minimum: [1; 1]);'])
disp(['f(xk): ', num2str(fk_n1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_n1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_n1))])
disp(['grad_norm: ', mat2str(gradfk_norm_n1)])
disp('****************** DONE ******************')
```

```
%% LOADING THE VARIABLES FOR THE TEST FOR X0 = [-1.2;1]

load('mytest2.mat')
disp('///////////////////////// SECOND POINT ////////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0 = [-1.2;1]

disp('**** STEEPEST DESCENT: START WITH X0=[-1.2;1] *****')
tic
[xk2, fk2, gradfk_norm2, k2, xseq2, btseq2] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0)])
disp(['xk: ', mat2str(xk2), ' (actual minimum: [1; 1]);'])
disp(['f(xk): ', num2str(fk2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq2))])
disp(['grad_norm: ', mat2str(gradfk_norm2)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0 = [-1.2;1]

disp('**** NEWTON: START WITH X0=[-1.2;1] *****')
tic
[xk_n2, fk_n2, gradfk_norm_n2, k_n2, xseq_n2, btseq_n2] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0)])
disp(['xk: ', mat2str(xk_n2), ' (actual minimum: [1; 1]);'])
disp(['f(xk): ', num2str(fk_n2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_n2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_n2))])
disp(['grad_norm: ', mat2str(gradfk_norm_n2)])
disp('****************** DONE ******************')

%% LOADING THE VARIABLES FOR THE TEST FOR X0_Rosenbrock = 1/3
disp('++++++++++++++++++++++++++ EXTENDED ROSENBROCK VALLEY TEST FUNCTION
++++++++++++++++++++++++++++++++')
load('ExRosenbrockValley_test(.33).mat')
disp('///////////////////////// POINT 0.33 ////////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_Rosenbrock = 1/3

disp('**** STEEPEST DESCENT: START WITH X0_Rosenbrock = 1/3 *****')
tic
[xkr, fkr, gradfk_normr, kr, xseqr, btseqr] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
```

```
disp(['xk_min: ', mat2str(min(xkr)), ', xk_max: ', mat2str(max(xkr))])
disp(['f(xk): ', num2str(fkr), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kr),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqr))])
disp(['grad_norm: ', mat2str(gradfk_normr)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_Rosenbrock = 1/3

disp('**** NEWTON: START WITH X0_Rosenbrock = 1/3 *****')
tic
[xk_nr, fk_nr, gradfk_norm_nr, k_nr, xseq_nr, btseq_nr] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('************************************')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_nr)), ', xk_max: ', mat2str(max(xk_nr))])
disp(['f(xk): ', num2str(fk_nr), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_nr),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_nr))])
disp(['grad_norm: ', mat2str(gradfk_norm_nr)])
disp('****************** DONE ******************')


%% LOADING THE VARIABLES FOR THE TEST FOR X0_Rosenbrock = 1

load('ExRosenbrockValley_test(1).mat')
disp('/////////////////////// POINT 1 ///////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_Rosenbrock = 1

disp('**** STEEPEST DESCENT: START WITH X0_Rosenbrock = 1 *****')
tic
[xkr1, fkr1, gradfk_normr1, kr1, xseqr1, btseqr1] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('************************************')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkr1)), ', xk_max: ', mat2str(max(xkr1))])
disp(['f(xk): ', num2str(fkr1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kr1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqr1))])
disp(['grad_norm: ', mat2str(gradfk_normr1)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_Rosenbrock = 1

disp('**** NEWTON: START WITH X0_Rosenbrock = 1 *****')
tic
[xk_nr1, fk_nr1, gradfk_norm_nr1, k_nr1, xseq_nr1, btseq_nr1] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
```

```
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_nr1)), ', xk_max: ',
mat2str(max(xk_nr1))])
disp(['f(xk): ', num2str(fk_nr1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_nr1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_nr1))])
disp(['grad_norm: ', mat2str(gradfk_norm_nr1)])
disp('****************** DONE ******************')


%% LOADING THE VARIABLES FOR THE TEST FOR X0_Rosenbrock = -1

load('ExRosenbrockValley_test(-1).mat')
disp('/////////////////////// POINT -1 ///////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_Rosenbrock = -1

disp('**** STEEPEST DESCENT: START WITH X0_Rosenbrock = -1 *****')
tic
[xkr2, fkr2, gradfk_normr2, kr2, xseqr2, btseqr2] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkr2)), ', xk_max: ', mat2str(max(xkr2))])
disp(['f(xk): ', num2str(fkr2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kr2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqr2))])
disp(['grad_norm: ', mat2str(gradfk_normr2)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_Rosenbrock = -1

disp('**** NEWTON: START WITH X0_Rosenbrock = -1 *****')
tic
[xk_nr2, fk_nr2, gradfk_norm_nr2, k_nr2, xseq_nr2, btseq_nr2] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_nr2)), ', xk_max: ',
mat2str(max(xk_nr2))])
disp(['f(xk): ', num2str(fk_nr2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_nr2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_nr2))])
disp(['grad_norm: ', mat2str(gradfk_norm_nr2)])
disp('****************** DONE ******************')
```

```matlab
%% LOADING THE VARIABLES FOR THE TEST FOR X0_pt1 = 1/3
disp('++++++++++++++++++++++++++ PENALTY TEST FUNCTION
+++++++++++++++++++++++++++++++++')
load('penaltyfunc1_test(.33).mat')
disp('//////////////////////// POINT 0.33 ///////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_pt1 = 1/3

disp('**** STEEPEST DESCENT: START WITH X0_pt1 = 1/3 *****')
tic
[xkp, fkp, gradfk_normp, kp, xseqp, btseqp] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('---------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkp)), ', xk_max: ', mat2str(max(xkp))])
disp(['f(xk): ', num2str(fkp), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kp),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqp))])
disp(['grad_norm: ', mat2str(gradfk_normp)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_pt1 = 1/3

disp('**** NEWTON: START WITH X0_pt1 = 1/3 *****')
tic
[xk_np, fk_np, gradfk_norm_np, k_np, xseq_np, btseq_np] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('---------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_np)), ', xk_max: ', mat2str(max(xk_np))])
disp(['f(xk): ', num2str(fk_np), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_np),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_np))])
disp(['grad_norm: ', mat2str(gradfk_norm_np)])
disp('****************** DONE ******************')


%% LOADING THE VARIABLES FOR THE TEST FOR X0_pt1 = 1

load('penaltyfunc1_test(1).mat')
disp('//////////////////////// POINT 1 ///////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_pt1 = 1

disp('**** STEEPEST DESCENT: START WITH X0_pt1 = 1 *****')
tic
[xkp1, fkp1, gradfk_normp1, kp1, xseqp1, btseqp1] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('---------------------------------------')
```

```
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkp1)), ', xk_max: ', mat2str(max(xkp1))])
disp(['f(xk): ', num2str(fkp1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kp1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqp1))])
disp(['grad_norm: ', mat2str(gradfk_normp1)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_pt1 = 1

disp('**** NEWTON: START WITH X0_pt1 = 1 *****')
tic
[xk_np1, fk_np1, gradfk_norm_np1, k_np1, xseq_np1, btseq_np1] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_np1)), ', xk_max: ',
mat2str(max(xk_np1))])
disp(['f(xk): ', num2str(fk_np1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_np1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_np1))])
disp(['grad_norm: ', mat2str(gradfk_norm_np1)])
disp('****************** DONE ******************')


%% LOADING THE VARIABLES FOR THE TEST FOR X0_pt1 = -1

load('penaltyfunc1_test(-1).mat')
disp('//////////////////////// POINT -1 ///////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_pt1 = -1

disp('**** STEEPEST DESCENT: START WITH X0_pt1 = -1 *****')
tic
[xkp2, fkp2, gradfk_normp2, kp2, xseqp1, btseqp2] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkp2)), ', xk_max: ', mat2str(max(xkp2))])
disp(['f(xk): ', num2str(fkp2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kp2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqp2))])
disp(['grad_norm: ', mat2str(gradfk_normp2)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_pt1 = -1

disp('**** NEWTON: START WITH X0_pt1 = -1 *****')
tic
[xk_np2, fk_np2, gradfk_norm_np2, k_np2, xseq_np2, btseq_np2] = ...
```

```
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_np2)), ', xk_max: ',
mat2str(max(xk_np2))])
disp(['f(xk): ', num2str(fk_np2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_np2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_np2))])
disp(['grad_norm: ', mat2str(gradfk_norm_np2)])
disp('****************** DONE ******************')


%% LOADING THE VARIABLES FOR THE TEST FOR X0_trigonometric = 1/3
disp('+++++++++++++++++++++++++++ TRIGONOMETRIC TEST FUNCTION
++++++++++++++++++++++++++++++++')
load('trigonometric_test(.33).mat')
disp('//////////////////////// POINT 0.33 ////////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_trigonometric = 1/3

disp('**** STEEPEST DESCENT: START WITH X0_trigonometric = 1/3 *****')
tic
[xkt, fkt, gradfk_normt, kt, xseqt, btseqt] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkt)), ', xk_max: ', mat2str(max(xkt))])
disp(['f(xk): ', num2str(fkt), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kt),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqt))])
disp(['grad_norm: ', mat2str(gradfk_normt)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_trigonometric = 1/3

disp('**** NEWTON: START WITH X0_trigonometric = 1/3 *****')
tic
[xk_nt, fk_nt, gradfk_norm_nt, k_nt, xseq_nt, btseq_nt] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_nt)), ', xk_max: ', mat2str(max(xk_nt))])
disp(['f(xk): ', num2str(fk_nt), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_nt),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_nt))])
disp(['grad_norm: ', mat2str(gradfk_norm_nt)])
disp('****************** DONE ******************')
```

```matlab
%% LOADING THE VARIABLES FOR THE TEST FOR X0_trigonometric = 1

load('trigonometric_test(1).mat')
disp('//////////////////////// POINT 1 ///////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_trigonometric = 1

disp('**** STEEPEST DESCENT: START WITH X0_trigonometric = 1 *****')
tic
[xkt1, fkt1, gradfk_normt1, kt1, xseqt1, btseqt1] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkt1)), ', xk_max: ', mat2str(max(xkt1))])
disp(['f(xk): ', num2str(fkt1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kt1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqt1))])
disp(['grad_norm: ', mat2str(gradfk_normt1)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_trigonometric = 1

disp('**** NEWTON: START WITH X0_trigonometric = 1 *****')
tic
[xk_nt1, fk_nt1, gradfk_norm_nt1, k_nt1, xseq_nt1, btseq_nt1] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_nt1)), ', xk_max: ',
mat2str(max(xk_nt1))])
disp(['f(xk): ', num2str(fk_nt1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_nt1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_nt1))])
disp(['grad_norm: ', mat2str(gradfk_norm_nt1)])
disp('****************** DONE ******************')


%% LOADING THE VARIABLES FOR THE TEST FOR X0_trigonometric = -1

load('trigonometric_test(-1).mat')
disp('//////////////////////// POINT -1 ///////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_trigonometric = -1

disp('**** STEEPEST DESCENT: START WITH X0_trigonometric = -1 *****')
tic
[xkt2, fkt2, gradfk_normt2, kt2, xseqt2, btseqt2] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
```

```matlab
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkt2)), ', xk_max: ', mat2str(max(xkt2))])
disp(['f(xk): ', num2str(fkt2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kt2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqt2))])
disp(['grad_norm: ', mat2str(gradfk_normt2)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_trigonometric = -1

disp('**** NEWTON: START WITH X0_trigonometric = -1 *****')
tic
[xk_nt2, fk_nt2, gradfk_norm_nt2, k_nt2, xseq_nt2, btseq_nt2] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_nt2)), ', xk_max: ',
mat2str(max(xk_nt2))])
disp(['f(xk): ', num2str(fk_nt2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_nt2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_nt2))])
disp(['grad_norm: ', mat2str(gradfk_norm_nt2)])
disp('****************** DONE ******************')

%% LOADING THE VARIABLES FOR THE TEST FOR X0_hilbert = 1/3
disp('+++++++++++++++++++++++++ HILBERT TEST FUNCTION
++++++++++++++++++++++++++++++++')
load('hilbert_test(.33).mat')
disp('/////////////////////// POINT 0.33 //////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_hilbert = 1/3

disp('**** STEEPEST DESCENT: START WITH X0_hilbert = 1/3 *****')
tic
[xkh, fkh, gradfk_normh, kh, xseqh, btseqh] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkh)), ', xk_max: ', mat2str(max(xkh))])
disp(['f(xk): ', num2str(fkh), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kh),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqh))])
disp(['grad_norm: ', mat2str(gradfk_normh)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_hilbert = 1/3

disp('**** NEWTON: START WITH X0_hilbert = 1/3 *****')
tic
```

```
[xk_nh, fk_nh, gradfk_norm_nh, k_nh, xseq_nh, btseq_nh] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_nh)), ', xk_max: ', mat2str(max(xk_nh))])
disp(['f(xk): ', num2str(fk_nh), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_nh),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_nh))])
disp(['grad_norm: ', mat2str(gradfk_norm_nh)])
disp('****************** DONE ******************')


%% LOADING THE VARIABLES FOR THE TEST FOR X0_hilbert = 1

load('hilbert_test(1).mat')
disp('///////////////////////// POINT 1 ////////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_hilbert = 1

disp('**** STEEPEST DESCENT: START WITH X0_hilbert = 1 *****')
tic
[xkh1, fkh1, gradfk_normh1, kh1, xseqh1, btseqh1] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkh1)), ', xk_max: ', mat2str(max(xkh1))])
disp(['f(xk): ', num2str(fkh1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kh1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqh1))])
disp(['grad_norm: ', mat2str(gradfk_normh1)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_hilbert = 1

disp('**** NEWTON: START WITH X0_hilbert = 1 *****')
tic
[xk_nh1, fk_nh1, gradfk_norm_nh1, k_nh1, xseq_nh1, btseq_nh1] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_nh1)), ', xk_max: ',
mat2str(max(xk_nh1))])
disp(['f(xk): ', num2str(fk_nh1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_nh1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_nh1))])
disp(['grad_norm: ', mat2str(gradfk_norm_nh1)])
disp('****************** DONE ******************')
```

```matlab
%% LOADING THE VARIABLES FOR THE TEST FOR X0_hilbert = -1

load('hilbert_test(-1).mat')
disp('////////////////////////// POINT -1 ////////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_hilbert = -1

disp('**** STEEPEST DESCENT: START WITH X0_hilbert = -1 *****')
tic
[xkh2, fkh2, gradfk_normh2, kh2, xseqh2, btseqh2] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkh2)), ', xk_max: ', mat2str(max(xkh2))])
disp(['f(xk): ', num2str(fkh2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kh2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqh2))])
disp(['grad_norm: ', mat2str(gradfk_normh2)])
disp('****************** DONE ******************')



%% RUN THE NEWTON FOR X0_hilbert = -1

disp('**** NEWTON: START WITH X0_hilbert = -1 *****')
tic
[xk_nh2, fk_nh2, gradfk_norm_nh2, k_nh2, xseq_nh2, btseq_nh2] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_nh2)), ', xk_max: ',
mat2str(max(xk_nh2))])
disp(['f(xk): ', num2str(fk_nh2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_nh2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_nh2))])
disp(['grad_norm: ', mat2str(gradfk_norm_nh2)])
disp('****************** DONE ******************')

%% LOADING THE VARIABLES FOR THE TEST FOR X0_Gregory = 1/3
disp('+++++++++++++++++++++++++ Gregory@Karney Tridiagonal Matrix TEST
FUNCTION +++++++++++++++++++++++++++++++')
load('Gregory_test(.33).mat')
disp('////////////////////////// POINT 0.33 ////////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_Gregory = 1/3

disp('**** STEEPEST DESCENT: START WITH X0_Gregory = 1/3 *****')
tic
[xkg, fkg, gradfk_normg, kg, xseqg, btseqg] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
```

```matlab
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkg)), ', xk_max: ', mat2str(max(xkg))])
disp(['f(xk): ', num2str(fkg), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kg),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqg))])
disp(['grad_norm: ', mat2str(gradfk_normg)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_Gregory = 1/3

disp('**** NEWTON: START WITH X0_Gregory = 1/3 *****')
tic
[xk_ng, fk_ng, gradfk_norm_ng, k_ng, xseq_ng, btseq_ng] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_ng)), ', xk_max: ', mat2str(max(xk_ng))])
disp(['f(xk): ', num2str(fk_ng), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_ng),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_ng))])
disp(['grad_norm: ', mat2str(gradfk_norm_ng)])
disp('****************** DONE ******************')


%% LOADING THE VARIABLES FOR THE TEST FOR X0_Gregory = 1

load('Gregory_test(1).mat')
disp('///////////////////////// POINT 1 ///////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_Gregory = 1

disp('**** STEEPEST DESCENT: START WITH X0_Gregory = 1 *****')
tic
[xkg1, fkg1, gradfk_normg1, kg1, xseqg1, btseqg1] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('--------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkg1)), ', xk_max: ', mat2str(max(xkg1))])
disp(['f(xk): ', num2str(fkg1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kg1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqg1))])
disp(['grad_norm: ', mat2str(gradfk_normg1)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_Gregory = 1

disp('**** NEWTON: START WITH X0_Gregory = 1 *****')
tic
[xk_ng1, fk_ng1, gradfk_norm_ng1, k_ng1, xseq_ng1, btseq_ng1] = ...
```

```
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_ng1)), ', xk_max: ',
mat2str(max(xk_ng1))])
disp(['f(xk): ', num2str(fk_ng1), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_ng1),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_ng1))])
disp(['grad_norm: ', mat2str(gradfk_norm_ng1)])
disp('****************** DONE ******************')


%% LOADING THE VARIABLES FOR THE TEST FOR X0_Gregory = -1

load('Gregory_test(-1).mat')
disp('/////////////////////////// POINT -1 ////////////////////////////')
%% RUN THE STEEPEST DESCENT FOR X0_Gregory = -1

disp('**** STEEPEST DESCENT: START WITH X0_Gregory = -1 *****')
tic
[xkg2, fkg2, gradfk_normg2, kg2, xseqg2, btseqg2] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** STEEPEST DESCENT: FINISHED *****')
disp('**** STEEPEST DESCENT: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xkg2)), ', xk_max: ', mat2str(max(xkg2))])
disp(['f(xk): ', num2str(fkg2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(kg2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseqg2))])
disp(['grad_norm: ', mat2str(gradfk_normg2)])
disp('****************** DONE ******************')


%% RUN THE NEWTON FOR X0_Gregory = -1

disp('**** NEWTON: START WITH X0_Gregory = -1 *****')
tic
[xk_ng2, fk_ng2, gradfk_norm_ng2, k_ng2, xseq_ng2, btseq_ng2] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc
disp('**** NEWTON: FINISHED *****')
disp('**** NEWTON: RESULTS *****')
disp('-------------------------------------')
disp(['START POINT: ', mat2str(x0(1))])
disp(['xk_min: ', mat2str(min(xk_ng2)), ', xk_max: ',
mat2str(max(xk_ng2))])
disp(['f(xk): ', num2str(fk_ng2), ' (actual min. value: 0);'])
disp(['N. of Iterations: ', num2str(k_ng2),'/',num2str(kmax), ';'])
disp(['BackTrackMax: ', mat2str(max(btseq_ng2))])
disp(['grad_norm: ', mat2str(gradfk_norm_ng2)])
disp('****************** DONE ******************')
```

B. STEEPEST_DESC_BCKTRCK:

```
function [xk, fk, gradfk_norm, k, xseq, btseq] = ...
    steepest_desc_bcktrck(x0, f, gradf, alpha0, ...
    kmax, tolgrad, c1, rho, btmax)
%
% [xk, fk, gradfk_norm, k, xseq] = steepest_descent(x0, f, gradf, alpha,
kmax,
% tollgrad)
%
% Function that performs the steepest descent optimization method, for a
% given function for the choice of the step length alpha.
%
% INPUTS:
% x0 = n-dimensional column vector;
% f = function handle that describes a function R^n->R;
% gradf = function handle that describes the gradient of f;
% alpha0 = the initial factor that multiplies the descent direction at each
% iteration;
% kmax = maximum number of iterations permitted;
% tolgrad = value used as stopping criterion w.r.t. the norm of the
% gradient;
% c1 = ?the factor of the Armijo condition that must be a scalar in (0,1);
% rho = ?fixed factor, lesser than 1, used for reducing alpha0;
% btmax = ?maximum number of steps for updating alpha during the
% backtracking strategy.
%
% OUTPUTS:
% xk = the last x computed by the function;
% fk = the value f(xk);
% gradfk_norm = value of the norm of gradf(xk)
% k = index of the last iteration performed
% xseq = n-by-k matrix where the columns are the xk computed during the
% iterations
% btseq = 1-by-k vector where elements are the number of backtracking
% iterations at each optimization step.
%

% Function handle for the armijo condition
farmijo = @(fk, alpha, gradfk, pk) ...
    fk + c1 * alpha * gradfk' * pk;

% Initializations
xseq = zeros(length(x0), kmax);
btseq = zeros(1, kmax);

xk = x0;
fk = f(xk);
gradfk = gradf(xk);
k = 0;
gradfk_norm = norm(gradfk);

while k < kmax && gradfk_norm >= tolgrad
    % Compute the descent direction
    pk = -gradf(xk);

    % Reset the value of alpha
    alpha = alpha0;
```

```
    % Compute the candidate new xk
    xnew = xk + alpha * pk;
    % Compute the value of f in the candidate new xk
    fnew = f(xnew);

    bt = 0;
    % Backtracking strategy:
    % 2nd condition is the Armijo condition not satisfied
    while bt < btmax && fnew > farmijo(fk, alpha, gradfk, pk)
        % Reduce the value of alpha
        alpha = rho * alpha;
        % Update xnew and fnew w.r.t. the reduced alpha
        xnew = xk + alpha * pk;
        fnew = f(xnew);

        % Increase the counter by one
        bt = bt + 1;

    end

    % Update xk, fk, gradfk_norm
    xk = xnew;
    fk = fnew;
    gradfk = gradf(xk);
    gradfk_norm = norm(gradfk);

    % Increase the step by one
    k = k + 1;

    % Store current xk in xseq
    xseq(:, k) = xk;
    % Store bt iterations in btseq
    btseq(k) = bt;
end

% "Cut" xseq and btseq to the correct size
xseq = xseq(:, 1:k);
btseq = btseq(1:k);

end
```

C. NEWTON_BCKTRCK:

```
function [xk, fk, gradfk_norm, k, xseq, btseq] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax)
%
% [xk, fk, gradfk_norm, k, xseq] = ...
%     newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
%     tolgrad, c1, rho, btmax)
%
% Function that performs the newton optimization method,
% implementing the backtracking strategy.
%
% INPUTS:
% x0 = n-dimensional column vector;
% f = function handle that describes a function R^n->R;
% gradf = function handle that describes the gradient of f;
% Hessf = function handle that describes the Hessian of f;
% kmax = maximum number of iterations permitted;
```

```matlab
% tolgrad = value used as stopping criterion w.r.t. the norm of the
% gradient;
% c1 = ?the factor of the Armijo condition that must be a scalar in (0,1);
% rho = ?fixed factor, lesser than 1, used for reducing alpha0;
% btmax = ?maximum number of steps for updating alpha during the
% backtracking strategy.
%
% OUTPUTS:
% xk = the last x computed by the function;
% fk = the value f(xk);
% gradfk_norm = value of the norm of gradf(xk)
% k = index of the last iteration performed
% xseq = n-by-k matrix where the columns are the xk computed during the
% iterations
% btseq = 1-by-k vector where elements are the number of backtracking
% iterations at each optimization step.
%

% Function handle for the armijo condition
farmijo = @(fk, alpha, gradfk, pk) ...
    fk + c1 * alpha * gradfk' * pk;

% Initializations
xseq = zeros(length(x0), kmax);
btseq = zeros(1, kmax);

xk = x0;
fk = f(xk);
k = 0;
gradfk = gradf(xk);
gradfk_norm = norm(gradfk);

while k < kmax && gradfk_norm >= tolgrad
    % Compute the descent direction as solution of
    % Hessf(xk) p = - graf(xk)
    pk = -Hessf(xk)\gradfk;

    % Reset the value of alpha
    alpha = 1;

    % Compute the candidate new xk
    xnew = xk + alpha * pk;
    % Compute the value of f in the candidate new xk
    fnew = f(xnew);

    bt = 0;
    % Backtracking strategy:
    % 2nd condition is the Armijo condition not satisfied
    while bt < btmax && fnew > farmijo(fk, alpha, gradfk, pk)
        % Reduce the value of alpha
        alpha = rho * alpha;
        % Update xnew and fnew w.r.t. the reduced alpha
        xnew = xk + alpha * pk;
        fnew = f(xnew);

        % Increase the counter by one
        bt = bt + 1;

    end
```

```matlab
    % Update xk, fk, gradfk_norm
    xk = xnew;
    fk = fnew;
    gradfk = gradf(xk);
    gradfk_norm = norm(gradfk);

    % Increase the step by one
    k = k + 1;

    % Store current xk in xseq
    xseq(:, k) = xk;
    % Store bt iterations in btseq
    btseq(k) = bt;
end

% "Cut" xseq and btseq to the correct size
xseq = xseq(:, 1:k);
btseq = btseq(1:k);

end
```

D. ***MAIN TEST FUNCTION:***

HESSF: @(X)[1200*X(1)^2-400*X(2)+2,-400*X(1);-400*X(1),200]

ALPHA: 1

F: @(X)100*(X(2,:)-X(1,:).^2).^2+(1-X(1,:)).^2

GRADF: @(X)[400*X(1,:).^3-400*X(1,:).*X(2,:)+2*X(1,:)-2;200*(X(2,:)-X(1,:).^2)]

KMAX: 1000

TOLGRAD: 1.0000E-12

X0: [2×1 DOUBLE]