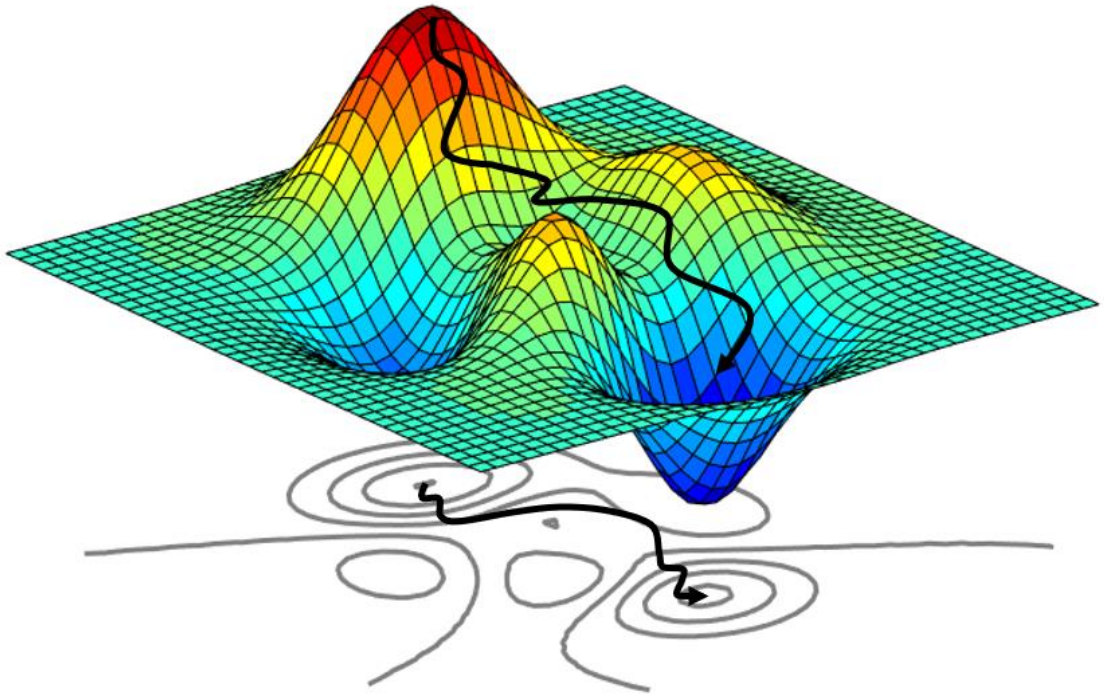


UNCONSTRAINED OPTIMIZATION

NUMERICAL METHODS AND OPTIMIZATION FOR LARGE SCALE PROBLEMS AND STOCHASTIC OPTIMIZATION

PROF. SANDRA PIERACCINI

POLITECNICO DI TORINO

LUIS ARIAS GONZALES S293581

ALBERTO MARTÍN GARRE S293972

Contents

1 Analysis of the problem **1**
- 1.1 Rosenbrock function . . . 1
  - 1.1.1 Two-Dimensional dimensions problem . . . 1
  - 1.1.2 N-Dimensional Problem . . . 1
- 1.2 Chained Wood function and Chained Powel singular function . . . 2
  - 1.2.1 Starting points suggested . . . 3
  - 1.2.2 Singular Hessian Matrix . . . 3
- 1.3 Summarize of methods . . . 3

2 Results and comments **4**
- 2.1 Rosenbrock function . . . 4
  - 2.1.1 Two-dimensional problem . . . 4
  - 2.1.2 N-dimensional problem . . . 6
- 2.2 Chained Wood Function and Chained Powel singular Function . . . 7

3 Appendix - Scripts and Functions Implemented **8**
- 3.1 Functions . . . 8
  - 3.1.1 TIUD28 . . . 8
  - 3.1.2 TFFU28 . . . 9
  - 3.1.3 TFGHU28 . . . 10
  - 3.1.4 Newton Method with Finite Differences . . . 11
  - 3.1.5 Inexact Newton Method with Finite Differences . . . 13
  - 3.1.6 Steepest Descent Method with Finite Differences . . . 15
  - 3.1.7 Nelder Mead Method . . . 17
- 3.2 Scripts - Test for the functions . . . 19
  - 3.2.1 Test for Rosenbrock Function: Two-Dimensional Problem . . . 19
  - 3.2.2 Test for three N-Dimensional problems . . . 22

List of Figures

1 Initial stage - Minimal solution (red) and starting points (yellow and green) 1
- (a) Rosenbrock surface and starting initial points . . . 1
- (b) Contour graph and starting initial points . . . 1

2 Steepest descent solutions . . . 4
- (a) Contour and Surface graphs . . . 4
- (b) Contour graph zoom in . . . 4

3 Newton Method solutions . . . 4
- (a) Contour and Surface graphs . . . 4
- (b) Contour graph zoom in . . . 4

4 Inexact Newton Method solutions . . . 4
- (a) Contour and Surface graphs . . . 4
- (b) Contour graph zoom in . . . 4

5 Nelder Mead method - Initial and final points . . . 5

6 Comparative methods table - Two Dimensional Rosenbrock function . . . 5

7 Comparative methods table - N-Dimensional Rosenbrock function . . . 6

8 Comparative methods table - Chained Wood function . . . 7

9 Comparative methods table - Chained Powel singular function . . . 7

1 Analysis of the problem

1.1 Rosenbrock function

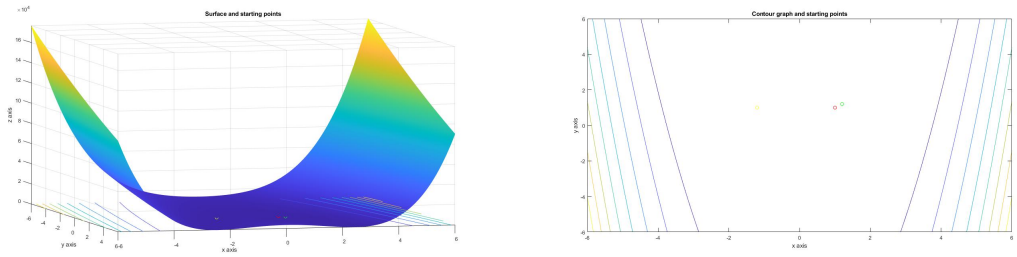
1.1.1 Two-Dimensional dimensions problem

Consider the following function:

$$f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$$

and two possible starting points $x_0 = (1.2, 1.2)$ and $x_0 = (-1.2, 1)$.

It is trivial that minimal solution for this function is $x = (1, 1)$ and the minimum is 0. First of all, it is crucial to understand the best as possible the behaviour of $f(x)$ in 3-D. That is why following pictures will allow to show the initial stage.



(a) Rosenbrock surface and starting initial points (b) Contour graph and starting initial points

Figure 1: Initial stage - Minimal solution (red) and starting points (yellow and green)

Pictures show some conclusions:

- Surface appearance shows that Rosenbrock function **will not** be complex for all values of **x**.
- It is important the choice of the initial point x_0**.**

1.1.2 N-Dimensional Problem

Consider the following function:

$$F(x) = \sum_{i=2}^{n} [100(x_{i-1}^2 - x_i)^2 + (x_{i-1} - 1)^2]$$

$$\overline{x_i} = -1.2, \quad mod(i, 2) = 1, \qquad \overline{x_i} = -1, \quad mod(i, 2) = 1,$$

The main task to take on with N-dimensional function is to prove if it is convex or not.

1. **Gradient of Rosenbrock function**

   Knowing that gradient of Rosenbrock function has this structure:

$$G = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_{N-1}}, \frac{\partial f}{\partial x_N} \right]^T$$

It can be distinguished three cases:

- $i = 1$
$$\frac{\partial f}{\partial x_1} = 400x_1^3 + 2x_1 - 400x_2x_1 - 2$$
- $i = N$
$$\frac{\partial f}{\partial x_N} = 200(x_N - x_{N-1}^2)$$
- $i = 2, ..., N-1$
$$\frac{\partial f}{\partial x_i} = 400x_i^3 - 400x_i \cdot x_{i+1} - 200x_{i-1}^2 + 202x_i - 2$$

2. **Hessian of Rosenbrock function**

After doing some calculations, final Hessian matrix can be computed by the following way:

$$H = \begin{pmatrix} 1200x_1^2 - 400x_2 + 2 & -400x1 & 0 & 0 & ... & 0 & 0 \\ -400x1 & 1200x_2^2 - 400x_3 + 202 & -400x_2 & 0 & ... & 0 & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & ... & -400x_{N-1} & 200 \end{pmatrix}$$

Main conclusions that can be inferred after this analysis are the following:

- This Hessian matrix **is not always semipositive definite** (determinant of some minors are less than 0) , so **there are some values where Rosenbrock function is not convex**.
- First conclusion causes that taking first solution (or first point) is an important task. **It will not be possible to get the minimum taking an incorrect initial point.**

This conclusions can be summarized taking into account the dimension of the problem:

- If $N < 4$: Minimum can be reached with any starting point.
- If $4 \leq N < \infty$: Rosenbrock function has an stationary point that is a global minimum and another one that changes with dimension N.

1.2 Chained Wood function and Chained Powel singular function

These are the other functions selected to prove some methods and get the minimum:

- Chained Wood Function:
$$F(x) = \sum_{j=1}^{k} \left[100(x_{i-1}^2 - x_i)^2 + (x_{i-1} - 1)^2 + 90(x_{i+1}^2 - x_{i+2})^2 + (x_{i+1} - 1)^2 + \right.$$
$$\left. + 10(x_i + x_{i+2} - 2)^2 + \frac{(x_i - x_{i+2})^2}{10} \right] \qquad i = 2j, \qquad k = \frac{N-2}{2}$$
- Chained Powel singular function
$$F(x) = \sum_{j=1}^{k} [(x_{i-1} + 10x_i)^2 + 5(x_{i+1} - x_{i+2})^2 + (x_i - 2x_{i+1})^4 + 10(x_{i-1} - x_{i+2})^4]$$
$$i = 2j, \qquad k = \frac{N-2}{2}$$

1.2.1 Starting points suggested

Starting points suggested by the problem are the following:

- Chained Wood Function

$$\overline{x_i} = -3, \quad mod(i,2) = 1, \quad i \leq 4; \qquad \overline{x_i} = -2, \quad mod(i,2) = 1, \quad i > 4$$
$$\overline{x_i} = -1, \quad mod(i,2) = 0, \quad i \leq 4; \qquad \overline{x_i} = 0, \quad mod(i,2) = 0, \quad i > 4$$

- Chained Powel singular function

$$\overline{x_i} = 3, \quad mod(i,4) = 1; \qquad \overline{x_i} = -1, \quad mod(i,4) = 2$$
$$\overline{x_i} = 0, \quad mod(i,4) = 3; \qquad \overline{x_i} = 1, \quad mod(i,4) = 0$$

1.2.2 Singular Hessian Matrix

It is important to realize that these two functions generate a singular Hessian Matrix in both starting points. This fact causes that methods chosen to solve this minimization problems **will not be** those that need a Hessian Matrix (commands *pcg* or *backslash* do not work with singular matrices).

1.3 Summarize of methods

There will be some problems and there will be different methods to solve them considering all facts previously described.

- **Rosenbrock function: Two and N-Dimensional Problem**
  1. Nelder-Mead Method
  2. Steepest Descent
  3. Newton Method
  4. Inexact Newton Method
- **Chained Wood function and Chained Powel function**
  1. Nelder-Mead Method
  2. Steepest Descent

Some aspects to take into account for methods implemented are the following:

- It is known that Gradient vector is needed for Steepest Descent method and both Gradient vector and Hessian Matrix are needed for Newton and Inexact Newton Method. **Both of them have been computed in all cases following finite differences.**
- It is known that an initial simplex with N+1 points is needed for Nelder-Mead Method. Stopping criteria will be $10 \cdot N$ iterations. It can be differentiated two cases:
  - Two - dimensional problem: Initial simplex (3 points) is constructed with the two points suggested and the other one is generated randomly with values between 0 and 2. So, initial simplex can be implemented as:
$$S_3 = \{[1.2; 1.2]^T, [-1.2; 1]^T, [random\{0-2\}; random\{0-2\}]^T\}$$
  - N - dimensional problem: Initial simplex (N+1 points) is made completely randomly with values between 0 and 2.

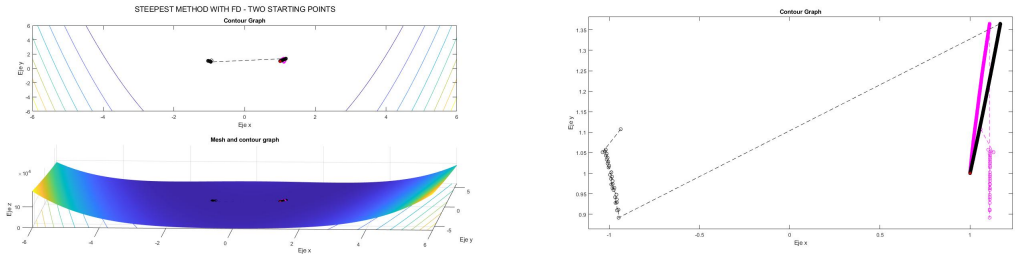
2 Results and comments

2.1 Rosenbrock function

2.1.1 Two-dimensional problem

Results obtained with different methods are presenting in this pictures. In red it is defined the optimal solution $x_{OPT} = [1; 1]^T$ and there are two ways arriving to it from the two starting points $x_0 = [1.2; 1.2]^T$ and $x_0 = [-1.2; 1]^T$.

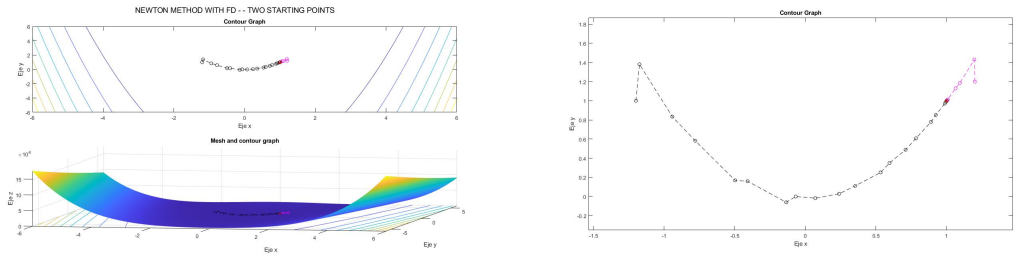
Steepest Descent Method



(a) Contour and Surface graphs (b) Contour graph zoom in

Figure 2: Steepest descent solutions

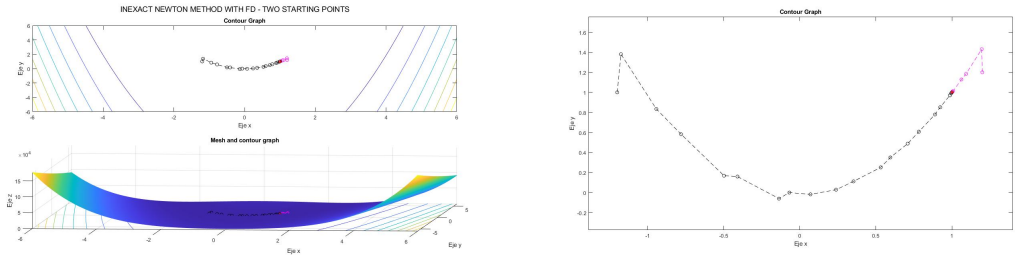
Newton Method



(a) Contour and Surface graphs (b) Contour graph zoom in

Figure 3: Newton Method solutions

Inexact Newton Method



(a) Contour and Surface graphs (b) Contour graph zoom in

Figure 4: Inexact Newton Method solutions

Nelder Mead Method

In order to prove the correct approximation of Nelder Mead method following picture shows the initial and final best points (from initial and final simplex).

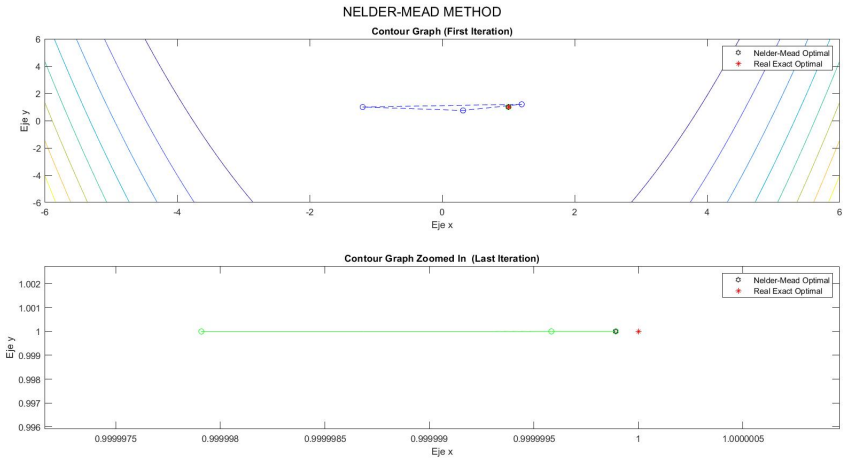


Figure 5: Nelder Mead method - Initial and final points

Some characteristics of different implemented methods are presented in the table to illustrate in a better way how they work on the Rosenbrock Two-Dimensional function.

		$X_0 = [1.2; 1.2]^T$	$X_0 = [-1.2; 1]^T$
Steepest Descent Method	Number of Iterations	10000	10000
	Computation Time	0.180343 sec	0.167382 sec
	Optimal theorical solution	$X = [1; 1]^T$	$X = [1; 1]^T$
	Optimal computed solution	$X = [1.0000; 1.0000]^T$	$X = [1.0000; 1.0000]^T$
	Theorical optimum	OPT = 0	OPT = 0
	Computed optimum	F = 3.7521e-13	F = 3.5997e-13
Newton Method	Number of Iterations	10000	21
	Computation Time	0.397355 sec	0.001141 sec
	Optimal theorical solution	$X = [1; 1]^T$	$X = [1; 1]^T$
	Optimal computed solution	$X = [1.0000; 1.0000]^T$	$X = [1.0000; 1.0000]^T$
	Theorical optimum	OPT = 0	OPT = 0
	Computed optimum	F = 9.6270e-13	7.6964e-12
Inexact Newton Method	Number of Iterations	10000	21
	Computation Time	5.061567 sec	0.016814 sec
	Optimal theorical solution	$X = [1; 1]^T$	$X = [1; 1]^T$
	Optimal computed solution	$X = [1.0000; 1.0000]^T$	$X = [1.0000; 1.0000]^T$
	Theorical optimum	OPT = 0	OPT = 0
	Computed optimum	F = 9.6270e-13	F = 7.6964e-12
		SIMPLEX SET: $S_3 = \{[1.2; 1.2]^T, [-1.2; 1]^T, [0.2569; 0.7354]^T$	
Nelder Mead Method	Number of Iterations	100	
	Computation Time	0.001917 secs	
	Optimal theorical solution	$X = [1; 1]^T$	
	Optimal computed solution	$X = [1.0000; 1.0000]^T$	
	Theorical optimum	OPT = 0	
	Computed optimum	F = 1.0161e-22	

Figure 6: Comparative methods table - Two Dimensional Rosenbrock function

Main conclusions that can be inferred by this table are the following:

- **General conclusions:** First of all, taking into consideration the low dimension of the problem, every algorithm will get to a good optimum (very close to the theoretical one). For this reason, the most important parameter is the computational time. Hence, Nelder Method is the one that provides the best performance. The reason behind this is that Nelder Method does not require neither the Hessian Matrix nor the Gradient vector.
- **Similar solutions between Newton and Inexact Newton Methods:** Inexact and Newton Method difference lies in the computation of the descent direction. Inexact Newton method approximates it in order to ease the linear system. This approximation is determined by a decreasing parameter called *forcing terms*. The nearer the k-iteration is to the real solution, the less value of these terms. Therefore, initial solutions in this problem are not far enough to make significant the effect of the forcing terms. This fact justifies the similarity between solutions obtained by these methods.
- **Relevance of starting points:** This conclusion excludes Nelder Method because it operates in a different way than the others. Focusing on the first starting point, it is shown that the algorithm that reports the best performance is Steepest Descent Method while Newton Method is the one that works better when starting from the second point.

2.1.2 N-dimensional problem

It is proved in Matlab that Newton and Inexact Newton Method work on this problem in the same way:

- If $N < 4$: These methods can reach the theoretical optimum with high accuracy.
- If $4 \leq N < \infty$: There is a moment in which Hessian Matrix is not semipositive definite (proved in Matlab computing its eigenvalues), so it is not convex in some intervals (there is a global minimum and another stationary point). That is why these methods can not reach the theoretical optimum point.

This behaviour is the reason why methods selected to solve this minimization problem are Steepest Descent Method and Nelder Mead Method.

PROBLEM 1		N=100	N=500	N=1000
Steepest Descent Method	Number of Iterations	100N=10.000	N=500	N=1000
	Computation Time	166,369114 secs	553,799533 secs	7.750,290881 secs
	Theorical optimum	OPT= 0	OPT= 0	OPT= 0
	Computed optimum	F=7,6310e-07	F=486,9777	F=974,6037
Nelder-Mead Method	Number of Iterations	100N=10.000	100N= 50.000	100N= 100.000
	Computation Time	1,068294 secs	98,988697 secs	749,709070 secs
	Theorical optimum	OPT= 0	OPT= 0	OPT= 0
	Computed optimum	F= 8,9627	F=20,399	F=34,3974

Figure 7: Comparative methods table - N-Dimensional Rosenbrock function

Table 7 shows that Steepest Descent is the best method when there is not a huge dimension (lower or equal to 100). The accuracy of the optimum obtained compensates the fact of that Steepest Descent is slower. For a greater dimension, Nelder Mead Method is the one that provides the best results (both in computational time and accuracy of the optimum).

2.2 Chained Wood Function and Chained Powel singular Function

As previously mentioned, Hessian matrices in both Chained Wood Function and Chained Powel Function are singular. Therefore, the methods selected to solve this minimization problems are the same as the N-Dimensional Rosenbrock function.

PROBLEM 2		N=100	N=500	N=1000
Steepest Descent Method	Number of Iterations	100N=10.000	N=500	N=1000
	Computation Time	217,078466 secs	719,344608 secs	10.261,955566 secs
	Theorical optimum	OPT= 0	OPT= 0	OPT= 0
	Computed optimum	F= 318,4934	F=1,9493e+03	F=3,9261e+03
Nelder-Mead Method	Number of Iterations	100N=10.000	100N= 50.000	100N= 100.000
	Computation Time	1,123467 secs	108,634605 secs	850,803436 secs
	Theorical optimum	OPT= 0	OPT= 0	OPT= 0
	Computed optimum	F=3,3395	F=10,0382	F=14,1632

Figure 8: Comparative methods table - Chained Wood function

Table 8 shows that Nelder Mead Method is the one that provides the best performance, taking into consideration all parameters presented, as number of iterations, computational time and computed optimum.

PROBLEM 3		N=100	N=500	N=1000
Steepest Descent Method	Number of Iterations	100N=10.000	N/10= 50	N/50 = 20
	Computation Time	612,895696 secs	1.792,361473 secs	5.913,683085 secs
	Theorical optimum	OPT= 0	OPT= 0	OPT= 0
	Computed optimum	F=4,5239e-07	F=0.0013	F=0,064
Nelder-Mead Method	Number of Iterations	100N=10.000	100N= 50.000	10N= 10.000
	Computation Time	15,118409 secs	1.848,298892 secs	2.023,549894 secs
	Theorical optimum	OPT= 0	OPT= 0	OPT= 0
	Computed optimum	F=90,6388	F=2,7248e+03	F=3,1637e+04

Figure 9: Comparative methods table - Chained Powel singular function

Table 9 presents that Steepest Descent Method is the most appropriate one to solve this problem (it provides a reliable optimum and needs less iterations than Nelder Mead Method). Indeed, is has been tried to reach a balance between the quality of the solution and the computational cost.

3 Appendix - Scripts and Functions Implemented

3.1 Functions

3.1.1 TIUD28

Function to get correctly initial points according to the problem.

```
1
2function [X, IERR, FMIN, XMAX] = TIUD28 (N, NEXT)
3
4%
5% Description - Help of the function
6%
7% function [X, IERR, FMIN, XMAX] = TIUD28(N, NEXT)
8%
9% Function that computes the initial X in different problems.
10%
11% INPUTS:
12% N = dimension of the problem
13% NEXT = number of the problem selected
14%
15% OUTPUTS:
16% X = column N-dimensional vector
17% IERR = error indicator (0 - correct data; 1 - N is too small)
18% FMIN = lower bound of the objective function value
19% XMAX = maximum stepsize
20
21X = zeros(N,1);
22
23switch NEXT
24
25    case 1
26        FMIN = 0;
27        XMAX = 1;
28        if N <= 1
29            IERR = 1;
30        else
31            for i=1:N
32                if mod(i,2)==1
33                    X(i) = -1.2;
34                else
35                    X(i) = 1;
36                end
37            end
38            IERR = 0;
39        end
40
41
42    case 2
43        FMIN = 0;
44        XMAX = 1;
45        if N <= 1
46            IERR = 1;
47        else
48            for i = 1:N
49                if i<=4
50                    if mod(i,2)==1
51                        X(i) = -3;
52                    else
53                        X(i) = -1;
54                    end
55                else
```

```
                    if mod(i,2)==1
                        X(i) = -2;
                    else
                        X(i) = 0;
                    end
                end
            end
            IERR = 0;
        end


    case 3
        FMIN = 0;
        XMAX = 1;
        if N <= 1
            IERR = 1;
        else
            for i = 1:N
                if mod(i,4)==1
                    X(i) = 3;
                else if mod(i,4)==2
                    X(i) = -1;
                else if mod(i,4)==3
                    X(i) = 0;
                else
                    X(i) = 1;
                end
                end
                end
            end
            IERR = 0;
        end
end
```

3.1.2 TFFU28

Function to get the value of functions in any point.

```

function [F] = TFFU28 (N, X, NEXT)

%
% Description - Help of the function
%
% function [F] = TFFU28(N, X, NEXT)
%
% Function that computes the value of different functions (from several
% selected problems) in a point X.
%
% INPUTS:
% N = dimension of the problem
% X = N-dimensional column vector;
% NEXT = number of the problem selected
%
% OUTPUTS:
% F = scalar value corresponding to the function evaluation in point X
% (depending on NEXT, the problem selected)

switch NEXT

    case 0
        F=(100*(X(2)-(X(1))^2)^2)+(1-X(1))^2;

```

```
    case 1
        aux = 0;
        for i=2:N
            suma = aux + (100*((X(i-1)^2-X(i))^2)+((X(i-1)-1)^2));
            aux = suma;
        end
        F = suma;

    case 2
        k = (N-2)/2;
        aux = 0;
        for j = 1:k
            i = 2*j;
            suma = aux + 100*((X(i-1)^2-X(i))^2)+(X(i-1)-1)^2+90*((X(i+1)^2-X
                (i+2))^2)+...
                    +(X(i+1)-1)^2+10*((X(i)+X(i+2)-2)^2)+((X(i)-X(i+2))^2)
                        /10;
            aux = suma;
        end
        F = suma;

    case 3
        k = (N-2)/2;
        aux = 0;
        for j = 1:k
            i = 2*j;
            suma = aux + (X(i-1)+10*X(i))^2 + 5*((X(i+1)-X(i+2))^2)+...
                    +(X(i)-2*X(i+1))^4 + 10*(X(i-1)-X(i+2))^4;
            aux = suma;
        end
        F = suma;
end
```

3.1.3 TFGHU28

Function to get the value of Gradient vector and Hessian matrix in any point.

```
function [G, H] = TFGHU28(N,X,NEXT,h)

% Description - Help of the function
%
% function [G,H] = TFGHU28(N, X, NEXT)
%
% Function that approximate the Gradient and the Hessian of F(evaluated in
% TFFU28) in X (column vector) with the forward finite difference method.
%
% INPUTS:
% N = dimension of the problem
% X = N-dimensional column vector;
% h = approximation step used for the finite difference computation of G
% NEXT = number of the problem selected
%
% OUTPUTS:
% G = column vector (same size of X) corresponding to the approximation
% of the Gradient of F in X.
% H = matrix (NxN) corresponding to the approximation of the Hessian of F
% in X.

G = zeros (N,1);

    for i=1:N
        X_h = X;
        X_h(i) = X_h(i) + h;
```

```
        G(i) = (TFFU28 (N, X_h, NEXT) - TFFU28 (N, X, NEXT))/ h;
    end

H = zeros(N, N);
% h for the Hessian is suggested to be greater the the h used for gradient
% in order to avoid numericall cancelation problems (h_hess = sqrt(h_grad))
h1 = sqrt(h);

for j=1:N
    % Elements on the diagonal
    Xh_plus = X;
    Xh_minus = X;
    Xh_plus(j) = Xh_plus(j) + h1;
    Xh_minus(j) = Xh_minus(j) - h1;
    H(j,j) = (TFFU28 (N, Xh_plus, NEXT) - 2*TFFU28 (N, X, NEXT) + TFFU28 (N,
        Xh_minus, NEXT))/(h1^2);

    % Elements of the other elements
    for i=j+1:N
        Xh_plus_ij = X;
        Xh_plus_ij([i, j]) = Xh_plus_ij([i, j]) + h1;
        Xh_plus_i = X;
        Xh_plus_i(i) = Xh_plus_i(i) + h1;
        Xh_plus_j = X;
        Xh_plus_j(j) = Xh_plus_j(j) + h1;
        H(i,j) = (TFFU28(N,Xh_plus_ij,NEXT) - TFFU28(N,Xh_plus_i,NEXT) -
            TFFU28(N,Xh_plus_j,NEXT) +...
                TFFU28(N,X,NEXT))/(h1^2);
        H(j,i)=H(i,j);
    end
end

end
```

3.1.4 Newton Method with Finite Differences

Function to get the solution applying Newton Method complemented with backtracking obtaining Gradient vector and Hessian matrix with finite differences.

```
function [Xk_N, F_k_N, G_k_norm_N, k_N, Xseq_N, btseq_N] = ...
    Newton_FinDiff_Back(X, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h)

% Description - Help of the function
%
% function [Xk, F_k, G_k_norm, k, Xseq, btseq] =
%    Newton_FinDiff_Back(X, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h)
%
% Function that performs Newton Method - Aspects to take into account:
% 1. Gradient and Hessian of F obtained by finite difference approximations
% 2. Newton Method complemented with a backtracking strategy (line search)
%
% INPUTS:
% X = n-dimensional initial column vector;
% k_max = 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;
% bt_max =  maximum  number of steps for updating alpha during the
% N = dimension of the problem;
% NEXT = number of the problem selected;
% h = approximation step used for the finite difference computation of G;
```

```
%
%
% OUTPUTS:
% Xk = last x computed by the function (best solution);
% F_k = value of F(Xk);
% G_k_norm = value of the norm of G(Xk)
% k = index of the last iteration performed
% Xseq = Matrix (NxK) with all solutions Xk computed during the iterations
% btseq = Row vector (size k) where elements are the number of backtracking
% iterations at each optimization step.


% Armijo Condition - Function handle
farmijo = @(F_k, alpha, G_k, pk) F_k + c1 * alpha * G_k' * pk;

% Initializations
Xseq_N = zeros(N, k_max);
btseq_N = zeros(1, k_max);

Xk_N = X;
F_k_N = TFFU28 (N, Xk_N, NEXT);
k_N = 0;
[G_k, H_k] = TFGHU28(N, Xk_N, NEXT, h);
G_k_norm_N = norm(G_k);
NonConvex = 0;


while k_N < k_max && G_k_norm_N >= tolgrad && NonConvex == 0

    % Preconditioned Conjugate Gradient Method to solve this:
    % H_k * p + G_k <= eta_k * G_k
    pk = -H_k\G_k;

    % Reset alpha value
    alpha = 1;

    % Column vector - New candidate Xk
    Xnew = Xk_N + alpha * pk;
    % Compute the value of f in the candidate new xk
    F_new = TFFU28(N,Xnew,NEXT);

    % Application of Backtracking strategy
    bt = 0;
    % While loop until Armijo condition is satisfied
    while bt < bt_max && F_new > farmijo(F_k_N, alpha, G_k, pk)
        % Reduce the value of alpha
        alpha = rho * alpha;
        % Update X_new and F_new w.r.t. the reduced alpha
        Xnew = Xk_N + alpha * pk;
        F_new = TFFU28(N,Xnew,NEXT);
        % Increase bactracking counter ("inner iteration")
        bt = bt + 1;
    end

    % Update Xk, F_k, G_k
    Xk_N = Xnew;
    F_k_N = F_new;
    [G_k, H_k] = TFGHU28(N,Xk_N,NEXT,h);
    G_k_norm_N = norm(G_k);
    M = eig (H_k);

    for i=1:N
        if M(i)<=0
```

```
87              NonConvex = 1;
88          end
89      end
90
91      % Increase step counter ("outer iteration")
92      k_N = k_N + 1;
93
94      % Store current xk in xseq
95      Xseq_N(:, k_N) = Xk_N;
96      % Store bt iterations in btseq
97      btseq_N(k_N) = bt;
98  end
99
100 % "Cut" xseq and btseq to the correct size
101 Xseq_N = [X Xseq_N(:, 1:k_N)];
102 btseq_N = btseq_N(1:k_N);
103
104 end
105
106 end
```

3.1.5 Inexact Newton Method with Finite Differences

Function to get the solution applying Inexact Newton Method complemented with backtracking obtaining Gradient vector and Hessian matrix with finite differences.

```
1
2 function [Xk_IN, F_k_IN, G_k_norm_IN, k_IN, Xseq_IN, btseq_IN] =
      InexactNewton_FinDiff_Back...
3     (X, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h, FT, pcg_maxit)
4
5 % Description - Help of the function
6 %
7 % function [Xk, F_k, G_k_norm, k, Xseq, btseq] = InexactNewton_FinDiff_Back
8 %           (X, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h, FT, pcg_maxit)
9 %
10 % Function that performs Inexact Newton Method - Aspects to take into
       account:
11 % 1. Gradient and Hessian of F obtained by finite difference approximations
12 % 2. Preconditioned Conjugate Gradient Method to solve this:
13 %    H_k * p + G_k <= eta_k * G_k
14 % 3. Newton Method complemented with a backtracking strategy (line search)
15 %
16 % INPUTS:
17 % X = n-dimensional initial column vector;
18 % k_max = maximum number of iterations permitted;
19 % tolgrad = value used as stopping criterion w.r.t. the norm of the
20 % gradient;
21 % c1 = the factor of the Armijo condition that must be a scalar in (0,1);
22 % rho = fixed factor, lesser than 1, used for reducing alpha0;
23 % bt_max = maximum number of steps for updating alpha during the
24 % N = dimension of the problem;
25 % NEXT = number of the problem selected;
26 % h = approximation step used for the finite difference computation of G;
27 % FT = Forcing Terms - Possible values
28 %            FT = 1 - Linear rate convergence
29 %            FT = 2 - Superlinear rate convergence
30 %            FT = 3 - Quadratic rate convergence
31 % pcg_maxit = maximum number of iterations for the pcg solver
32 %
33 %
34 % OUTPUTS:
```

```
% Xk = last x computed by the function (best solution);
% F_k = value of F(Xk);
% G_k_norm = value of the norm of G(Xk)
% k = index of the last iteration performed
% Xseq = Matrix (NxK) with all solutions Xk computed during the iterations
% btseq = Row vector (size k) where elements are the number of backtracking
% iterations at each optimization step.


% Armijo Condition - Function handle
farmijo = @(F_k, alpha, G_k, pk) F_k + c1 * alpha * G_k' * pk;

% Initializations
Xseq_IN = zeros(N, k_max);
btseq_IN = zeros(1, k_max);

Xk_IN = X;
F_k_IN = TFFU28 (N, Xk_IN, NEXT);
k_IN = 0;
[G_k, H_k] = TFGHU28(N, Xk_IN, NEXT, h);
G_k_norm_IN = norm(G_k);
NonConvex = 0;

while k_IN < k_max && G_k_norm_IN >= tolgrad && NonConvex == 0

    switch FT
        case 1
            % Rate convergence is linear
            eta_k = 0.5;

        case 2
            % Rate convergence is superlinear
            eta_k = min(0.5, sqrt(G_k_norm_IN));

        case 3
            % Rate convergence is quadratic
            eta_k = min(0.5, G_k_norm_IN);
    end

    % INEXACT NEWTON METHOD
    % Not solving this linear system: H_k * p = - G_k
    % Instead of this, we approximate H_k * p + G_k <= small quantity
    % Small quantity: eta_k * G_k

    % The smaller G_k is, the more precise is the direction p (because we
    % are close to the solution)

    % Preconditioned Conjugate Gradient Method to solve this:
    % H_k * p + G_k <= eta_k * G_k
    R = ichol(sparse(H_k));
    pk = pcg(H_k, -G_k, eta_k, pcg_maxit, R, R');

    % Reset alpha value
    alpha = 1;

    % Column vector - New candidate Xk
    Xnew = Xk_IN + alpha * pk;
    % Compute the value of f in the candidate new xk
    F_new = TFFU28(N,Xnew,NEXT);

    % Application of Backtracking strategy
    bt = 0;
    while bt < bt_max && F_new > farmijo(F_k_IN, alpha, G_k, pk)
```

```
98          % Reduce the value of alpha
99          alpha = rho * alpha;
100         % Update X_new and F_new w.r.t. the reduced alpha
101         Xnew = Xk_IN + alpha * pk;
102         F_new = TFFU28(N,Xnew,NEXT);
103         % Increase bactracking counter ("inner iteration")
104         bt = bt + 1;
105     end
106
107     % Update Xk, F_k, G_k
108     Xk_IN = Xnew;
109     F_k_IN = F_new;
110     [G_k, H_k] = TFGHU28(N,Xk_IN,NEXT,h);
111     G_k_norm_IN = norm(G_k);
112
113     % Increase step counter ("outer iteration")
114     k_IN = k_IN + 1;
115
116     % Store current xk in xseq
117     Xseq_IN(:, k_IN) = Xk_IN;
118     % Store bt iterations in btseq
119     btseq_IN(k_IN) = bt;
120
121     M = eig (H_k);
122
123     for i = 1:N
124         if M(i) < 0
125             NonConvex = 1;
126         end
127     end
128 end
129
130 % "Cut" xseq and btseq to the correct size
131 Xseq_IN = [X Xseq_IN(:, 1:k_IN)];
132 btseq_IN = btseq_IN(1:k_IN);
133
134 end
```

3.1.6 Steepest Descent Method with Finite Differences

Function to get the solution applying Steepest Descent Method complemented with backtracking obtaining Gradient vector with finite differences.

```
1
2 function [Xk_SD, F_k_SD, G_k_norm_SD, k_SD, Xseq_SD, btseq_SD] = ...
3     SD_FinDiff_Back(X, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h)
4 %
5 % [xk, fk, gradfk_norm, k, xseq] = steepest_descent(x0, f, gradf, alpha,
      kmax,
6 % tollgrad)
7 %
8 % Function that performs the steepest descent optimization method, for a
9 % given function for the choice of the step length alpha.
10 %
11 % INPUTS:
12 % x0 = n-dimensional column vector;
13 % f = function handle that describes a function R^n->R;
14 % gradf = function handle that describes the gradient of f;
15 % alpha0 = the initial factor that multiplies the descent direction at each
16 % iteration;
17 % kmax = maximum number of iterations permitted;
18 % tolgrad = value used as stopping criterion w.r.t. the norm of the
```

```
19 % gradient;
20 % c1 =  the  factor of the Armijo condition that must be a scalar in (0,1);
21 % rho =  fixed  factor, lesser than 1, used for reducing alpha0;
22 % btmax =  maximum  number of steps for updating alpha during the
23 % backtracking strategy.
24 %
25 % OUTPUTS:
26 % xk = the last x computed by the function;
27 % fk = the value f(xk);
28 % gradfk_norm = value of the norm of gradf(xk)
29 % k = index of the last iteration performed
30 % xseq = n-by-k matrix where the columns are the xk computed during the
31 % iterations
32 % btseq = 1-by-k vector where elements are the number of backtracking
33 % iterations at each optimization step.
34 %
35
36 % Armijo Condition - Function handle
37 farmijo = @(F_k, alpha, G_k, pk) F_k + c1 * alpha * G_k' * pk;
38
39 % Initializations
40 Xseq_SD = zeros(N, k_max);
41 btseq_SD = zeros(1, k_max);
42
43 Xk_SD = X;
44 F_k_SD = TFFU28 (N, Xk_SD, NEXT);
45 k_SD = 0;
46 [G_k, ~] = TFGHU28(N, Xk_SD, NEXT, h);
47 G_k_norm_SD = norm(G_k);
48 NonConvex = 0;
49 alpha0 = 5;
50
51 while k_SD < k_max && G_k_norm_SD >= tolgrad && NonConvex == 0
52     % Compute the descent direction
53     pk = -G_k;
54
55     % Reset the value of alpha
56     alpha = alpha0;
57
58     % Compute the candidate new xk
59     Xk_new = Xk_SD + alpha * pk;
60     % Compute the value of f in the candidate new xk
61     Fk_new = TFFU28 (N, Xk_new, NEXT);
62
63     bt = 0;
64     % Backtracking strategy:
65     % 2nd condition is the Armijo condition not satisfied
66     while bt < bt_max && Fk_new > farmijo(F_k_SD, alpha, G_k, pk)
67         % Reduce the value of alpha
68         alpha = rho * alpha;
69         % Update xnew and fnew w.r.t. the reduced alpha
70         Xk_new = Xk_SD + alpha * pk;
71         Fk_new = TFFU28 (N, Xk_new, NEXT);
72
73         % Increase the counter by one
74         bt = bt + 1;
75
76     end
77
78     % Update xk, fk, gradfk_norm
79     Xk_SD = Xk_new;
80     F_k_SD = Fk_new;
81     [G_k, ~] = TFGHU28(N, Xk_SD, NEXT, h);
```

```
82      G_k_norm_SD = norm(G_k);
83
84      % Increase the step by one
85      k_SD = k_SD + 1;
86
87      % Store current xk in xseq
88      Xseq_SD(:, k_SD) = Xk_SD;
89      % Store bt iterations in btseq
90      btseq_SD(k_SD) = bt;
91  end
92
93  % "Cut" xseq and btseq to the correct size
94  Xseq_SD = Xseq_SD(:, 1:k_SD);
95  btseq_SD = btseq_SD(1:k_SD);
96
97  end
```

3.1.7 Nelder Mead Method

Function to get the solution applying Nelder Mead Method with properties explained previously.

```
1
2  function [X0,Xk_N, F_k_N, k_N,Xseq_N] = ...
3      Nelder_Method(k_max, N, NEXT,rho,chi,gamma,sigma)
4
5  % Description - Help of the function
6  %
7  % function [X0,Xk_N, F_k_N, k_N,Xseq_N] = ...
8  %             Nelder_Method(k_max, N, NEXT,rho,chi,gamma,sigma)
9  %
10 % INPUTS:
11 % k_max = maximum number of iterations permitted;
12 % N = dimension of the problem;
13 % NEXT = number of the problem selected;
14 % tolgrad = value used as stopping criterion w.r.t. the norm of the
15 % gradient;
16 % c1 =   the   factor of the Armijo condition that must be a scalar in (0,1);
17 %
18 % Typical values:
19 %  rho=1
20 %  chi= 2
21 %  gamma=0.5
22 %  sigma=0.5
23 %
24 % OUTPUTS:
25 % X0 = initial simplex set (randomly selected in N-Dimension);
26 % Xk_N = final optimal solution obtained by Nelder Mead Method;
27 % F_k_N = final optimum (value of the function at the optimal point);
28 % Xseq_N = When NEXT = 0, it represents the group of simplex sets;
29 %
30 %
31 % Initialize the matrix for defining the simplex(each column is a vertice of
32 % the simple)
33
34 switch NEXT
35     case 0
36         X=[1.2,-1.2,0;1.2,1,0];
37         %Generate randomly a third point with values between 0-2
38         for i=1:2
39             X(i,3)=2*rand();
40         end
```

```
        Xseq_N=[X X(:,1)];

    otherwise
        X=zeros(N,N+1);
        %Generate randomly points with values between 0-2
        for i=1:N
            for j=1:(N+1)
                X(i,j)=2*rand();
            end
        end
end

X0=X;
F=zeros(1,N+1);
k=0;

%Start the loop

while k<k_max
    k=k+1;

%1. ORDERING PHASE
%Compute the value of the function at he vertices
for i=1:N+1
    F(i)=TFFU28(N,X(:,i),NEXT);
end

%Order both the function vector and the points
[F_ordered,I]=sort(F);

X_ordered=zeros(N,N+1);
for j=1:(N+1)
    X_ordered(:,j)=X(:,I(j));
end

%2. REFLECTION PHASE
%Compute the barycenter

x_baricenter=zeros(N,1);
for j=1:N
    x_baricenter=x_baricenter+X_ordered(:,j);
end
x_baricenter=x_baricenter/N;

%Compute the reflection of x(N+1)
X_R=zeros(N,1);
X_E=zeros(N,1);
X_C=zeros(N,1);

X_R=x_baricenter+(rho*(x_baricenter-X_ordered(:,N+1)));
F_R=TFFU28(N,X_R,NEXT);

if F_R>=F_ordered(1) && F_R<F_ordered(N)
    %Enought reduction with xR%
    %Accept xR and go to next step
    X_ordered(:,N+1)=X_R;
else
    if F_R<F_ordered(1)
        %Large reduction
        X_E=x_baricenter+(chi*(X_R-x_baricenter));
        F_E=TFFU28(N,X_E,NEXT);
        %EXPANSION PHASE
        if F_E<F_R
```

```
104             %Accept xE and go to next step
105             X_ordered(:,N+1)=X_E;
106         else
107             %Accept xR and go to next step
108             X_ordered(:,N+1)=X_R;
109         end
110     else
111         %Poor Reduction (fR>=fN)
112         %CONTRACTION PHASE%
113         X_C=x_baricenter+(gamma*(X_ordered(:,N+1)-x_baricenter));
114         F_C= TFFU28(N,X_C,NEXT);
115         if F_C<F_ordered(N+1)
116             %Accept xC and go to next step
117             X_ordered(:,N+1)=X_C;
118         else
119             %SHRINKAGE PHASE
120             for i=2:(N+1)
121                 X_ordered(:,i)=X_ordered(:,1)-(sigma*(X_ordered(:,i)-
                        X_ordered(:,1)));
122             end
123         end
124     end
125 end
126
127 %Determine Xseq for base case(N=2)
128 if NEXT==0
129     g=k_max/10;
130     if mod(k,g)==0
131         Xseq_N=[Xseq_N X_ordered X_ordered(:,1)];
132     end
133 end
134
135 %Reset values for the next iteration
136 X=X_ordered;
137
138 end
139
140 k_N=k;
141 %Order the solutions to report the best one
142 for i=1:N+1
143     F(i)=TFFU28(N,X(:,i),NEXT);
144 end
145
146 [F_ordered,I]=sort(F);
147
148 X_ordered=zeros(N,N+1);
149 for j=1:(N+1)
150     X_ordered(:,j)=X(:,I(j));
151 end
152
153 Xk_N=X_ordered(:,1);
154 F_k_N=F_ordered(1);
155 end
```

3.2 Scripts - Test for the functions

3.2.1 Test for Rosenbrock Function: Two-Dimensional Problem

Script to get the optimal solutions and optimums by four different methods. Also, it provides some useful pictures to analyze better the behaviour and the convergence of each method.

```
%% EVALUATION CODE - ROSENBROCK FUNCTION - N = 2

N = 2;
NEXT = 0;
X1 = [1.2; 1.2];
X2 = [-1.2; 1];
k_max = 1e4;
tolgrad = 1e-6;
c1 = 1e-4;
rho = 0.5;
bt_max = 100;
h = 1e-8;
FT = 3;
pcg_maxit = 100*N;

% Nelder Method
k_max_ND = 70;
rho_ND = 1;
chi=2;
gamma=0.5;
sigma=0.5;

%% Newton Method
tic
[Xk_N1, F_k_N1, G_k_norm_N1, k_N1, Xseq_N1, btseq_N1] = ...
    Newton_FinDiff_Back(X1, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h);
toc

tic
[Xk_N2, F_k_N2, G_k_norm_N2, k_N2, Xseq_N2, btseq_N2] = ...
    Newton_FinDiff_Back(X2, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h);
toc

%% Inexact Newton Method
tic
[Xk_IN1, F_k_IN1, G_k_norm_IN1, k_IN1, Xseq_IN1, btseq_IN1] =
    InexactNewton_FinDiff_Back...
    (X1, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h, FT, pcg_maxit);
toc

tic
[Xk_IN2, F_k_IN2, G_k_norm_IN2, k_IN2, Xseq_IN2, btseq_IN2] =
    InexactNewton_FinDiff_Back...
    (X2, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h, FT, pcg_maxit);
toc

%% Steepest Descent Method
tic
[Xk_SD1, F_k_SD1, G_k_norm_SD1, k_SD1, Xseq_SD1, btseq_SD1] = ...
    SD_FinDiff_Back(X1, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h);
toc

tic
[Xk_SD2, F_k_SD2, G_k_norm_SD2, k_SD2, Xseq_SD2, btseq_SD2] = ...
    SD_FinDiff_Back(X2, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h);
toc

%% Nelder-Mead Method
tic
[X0_ND,Xk_ND, F_k_ND, k_ND, Xseq_ND] = Nelder_Method...
    (k_max_ND, N, NEXT,rho_ND,chi,gamma,sigma);
toc
```

```
62
63%% PLOTS
64
65% Creation of the meshgrid for the contour-plot
66[L, M] = meshgrid(linspace(-6, 6, 500), linspace(-6, 6, 500));
67% Computation of the values of f for each point of the mesh
68Z = 100*(M-L.^2).^2+(1-L).^2;
69
70% PLOT - NEWTON METHOD WITH FINITE DIFFERENCES
71
72fig1 = figure();
73sgtitle ('NEWTON METHOD WITH FD - TWO STARTING POINTS')
74subplot (2,1,1)
75contour(L, M, Z);
76hold on
77title ('Contour Graph') , xlabel ('Eje x'), ylabel ('Eje y')
78plot(Xseq_N1(1,:), Xseq_N1(2,:), '-om')
79plot(Xseq_N2(1,:), Xseq_N2(2,:), '-ok')
80plot(1,1,'*r')
81hold off
82
83subplot (2,1,2)
84meshc(L, M, Z);
85hold on
86title ('Mesh and contour graph') , xlabel ('Eje x'), ylabel ('Eje y'),
    zlabel ('Eje z')
87plot(Xseq_N1(1,:), Xseq_N1(2,:), '-om')
88plot(Xseq_N2(1,:), Xseq_N2(2,:), '-ok')
89plot(1,1,'*r')
90hold off
91
92% PLOT - INEXACT NEWTON METHOD WITH FINITE DIFFERENCES
93
94fig2 = figure();
95sgtitle ('INEXACT NEWTON METHOD WITH FD - TWO STARTING POINTS')
96subplot (2,1,1)
97contour(L, M, Z);
98hold on
99title ('Contour Graph') , xlabel ('Eje x'), ylabel ('Eje y')
100plot(Xseq_IN1(1,:), Xseq_IN1(2,:), '-om')
101plot(Xseq_IN2(1,:), Xseq_IN2(2,:), '-ok')
102plot(1,1,'*r')
103hold off
104
105subplot (2,1,2)
106meshc(L, M, Z);
107hold on
108title ('Mesh and contour graph') , xlabel ('Eje x'), ylabel ('Eje y'),
    zlabel ('Eje z')
109plot(Xseq_IN1(1,:), Xseq_IN1(2,:), '-om')
110plot(Xseq_IN2(1,:), Xseq_IN2(2,:), '-ok')
111plot(1,1,'*r')
112hold off
113
114% PLOT - STEEPEST DESCENT WITH FINITE DIFFERENCES
115
116fig3 = figure();
117sgtitle ('STEEPEST METHOD WITH FD - TWO STARTING POINTS')
118subplot (2,1,1)
119contour(L, M, Z);
120hold on
121title ('Contour Graph') , xlabel ('Eje x'), ylabel ('Eje y')
122plot(Xseq_SD1(1,:), Xseq_SD2(2,:), '-om')
```

```
plot(Xseq_SD2(1,:), Xseq_SD2(2,:), '--ok')
plot(1,1,'*r')
hold off

subplot (2,1,2)
meshc(L, M, Z);
hold on
title ('Mesh and contour graph') , xlabel ('Eje x'), ylabel ('Eje y'),
    zlabel ('Eje z')
plot(Xseq_SD1(1,:), Xseq_SD1(2,:), '--om')
plot(Xseq_SD2(1,:), Xseq_SD2(2,:), '--ok')
plot(1,1,'*r')
hold off

%PLOT - NELDER-MEAD METHOD

fig4 = figure();
sgtitle ('NELDER-MEAD METHOD ')
subplot (2,1,1)
contour(L, M, Z);
hold on
title ('Contour Graph (First Iteration)') , xlabel ('Eje x'), ylabel ('Eje y
    ')

plot(Xseq_ND(1,1:4), Xseq_ND(2,1:4), '--ob')
plot(Xseq_ND(1,29:32), Xseq_ND(2,29:32), '--ok')
plot(Xseq_ND(1,41:44), Xseq_ND(2,41:44), '--og')

b1=plot(Xk_ND(1),Xk_ND(2),'hk');
b2=plot(1,1,'*r');
legend([b1 b2],'Nelder-Mead Optimal','Real Exact Optimal')

hold off

subplot (2,1,2)
contour(L, M, Z);
hold on
title ('Contour Graph Zoomed In  (Last Iteration)') , xlabel ('Eje x'),
    ylabel ('Eje y')

plot(Xseq_ND(1,1:4), Xseq_ND(2,1:4), '--ob')
plot(Xseq_ND(1,29:32), Xseq_ND(2,29:32), '--ok')
plot(Xseq_ND(1,41:44), Xseq_ND(2,41:44), '--og')

b1=plot(Xk_ND(1),Xk_ND(2),'hk');
b2=plot(1,1,'*r');
legend([b1 b2],'Nelder-Mead Optimal','Real Exact Optimal')

hold off
```

3.2.2 Test for three N-Dimensional problems

Script to get the optimal solutions and optimums by two different methods, concretely Steepest Descent Method and Nelder Mead Method.

```

%% EVALUATION OF THE CODE - N-DIMENSIONAL PROBLEMS

% Initializions
N = 50;
NEXT = 1;
[X, IERR, FMIN, XMAX] = TIUD28 (N, NEXT);
```

```
k_max = 1e4;
tolgrad = 1e-4;
c1 = 1e-5;
rho = 0.75;
bt_max = 100;
h = 1e-8;
FT = 1;
pcg_maxit = 100*N;

% Nelder Method
k_max_ND = 1000;
rho_ND = 1;
chi=2;
gamma=0.5;
sigma=0.5;

%% Steepest Descent Method
tic
[Xk_SD, F_k_SD, G_k_norm_SD, k_SD, Xseq_SD, btseq_SD] = ...
    SD_FinDiff_Back(X, k_max, tolgrad, c1, rho, bt_max, N, NEXT, h);
toc

%% NELDER-MEAD method
tic
[X0,Xk_ND, F_k_ND, k_ND] = Nelder_Method(k_max_ND, N, NEXT, rho_ND, chi,
    gamma, sigma);
toc
```