Unconstrained optimization

Numerical and stochastic optimization for large scale problems

Luca Panichi 316015
Luigi Del Gaudio 303027

Contents

Introduction **i**

0.1 Steepest descent and Newton method i

1 Test function ω - Rosenbrock **1**

2 Test function ϕ - Problem 16 **4**

3 Test function ψ - Problem 27 **6**

4 Test function ξ - Problem 28 **7**

5 Appendix **9**

5.1 Newton method 9
5.2 Steepest descent 11
5.3 Rosenbrock test 12
5.4 Problem 16 - Function 13
5.5 Problem 16 - Gradient 14
5.6 Problem 16 - Hessian 14
5.7 Problem 27 - Function 14
5.8 Problem 27 - Gradient 15
5.9 Problem 27 - Hessian 15
5.10 Problem 28 - Function 16
5.11 Problem 28 - Gradient 16
5.12 Problem 28 - Hessian 17
5.13 Test 17
5.14 3D-Figure 19
5.15 Newton Figure 20
5.16 Steepest descent Figure 20

Introduction

0.1 Steepest descent and Newton method

In this work we are going to discuss and implement two different methods of unconstrained optimization in order to solve problems as

$$\min_{x \in \mathbb{R}^n} f(x)$$

where

$$f : \mathbb{R}^n \to \mathbb{R}$$

is a smooth function. We will face the issue with two iterative methods:

- Steepest descent method
- Newton method

Let's start analysing the steepest descent method. The main idea is to start from $x_0 \in \mathbb{R}^n$ and to compute a sequence $\{x_k\}_{k \in \mathbb{N}}$ moving along the descent direction $p_k = -\nabla f(x_k)$ in the following way:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

where α_k is the steplength that has to be chosen with the backtracking strategy. Defining the Armijo condition

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k)^T p_k$$

starting from an initial steplength $\alpha_k^{(0)}$, for $j \geq 0$, if Armijo condition is satisfied, we accept $\alpha_k^{(j)}$, otherwise for some $\rho < 1$ we compute

$$\alpha_k^{(j+1)} = \rho \alpha_k^{(j)}.$$

Let's introduce now the Newton method from a theoretical point of view. We want to approximate $f(x_k + p)$ with the Taylor polynomial of 2nd order

$$f(x_k + p) \approx f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T \nabla^2 f(x_k) p =: m_{f,x_k}(p)$$

the Netwon steps consist in looking for the minimum of $m_{f,x_k}(p)$ for each k. Therefore, computing the gradient of $m_{f,x_k}(p)$

$$\nabla_p m_{f,x_k}(p) = \nabla f(x_k) + \nabla^2 f(x_k)p$$

we look for the stationary point of $m_{f,x_k}(p)$ and we get

$$p_k = -(\nabla^2 f(x_k))^{-1}\nabla f(x_k)$$

In order to choose α_k appropiately, similarly to steepest descent we use the backtracking strategy. However, if the choice of $\alpha_k^{(0)}$ depends on the method, for Netwon method it is important to choose $\alpha_k^{(0)} = 1$. Finally, we will run both optimization methods for four different objective functions, using the following parameters:

- $n = 10^3$
- $kmax = 10000$
- $\alpha_0 = 1$
- $c_1 = 10^{-4}$
- $btmax = 50$
- $\rho = 0.5$

1 Test function ω - Rosenbrock

Let's start to analyse the results of Newton and steepest descent method on the *Rosenbrock function* (A. 5.3):

$$\omega(x, y) = 100(y - x^2)^2 + (1 - x)^2 \tag{1.1}$$

This function has been tested with two different initial points

$$x_1 = [1.2, 1.2], \quad x_2 = [-1.2, 1].$$

Its gradient is

$$\nabla\omega(x, y) = [400x^3 - 400yx + 2x - 2, 200y - 200x^2]$$

while the hessian matrix is the following:

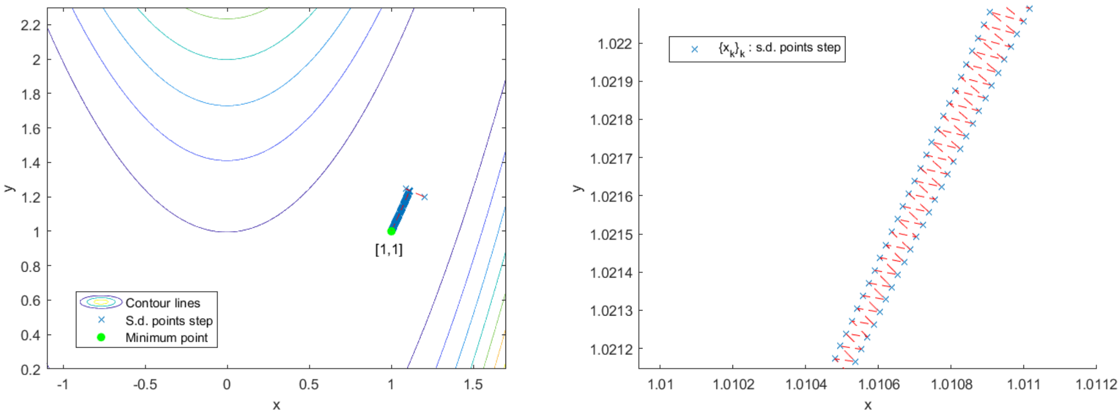
$$Hess(\omega) = \nabla^2\omega = \begin{pmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{pmatrix}.$$

The Table 1.1 contains the result for each method of the minimizer, the actual minimum, the number of iterations and the computation time. Fixed the parameter $\alpha_0 = 1$ for the backtracking strategy, in both methods, we can observe that both of them converge to the minimum point $[1, 1]$, but with a significant difference in the speed of convergence. Indeed, we know from the theory that if the Newton method converges, it reaches the solution very quickly: in particular the Newton method has a quadratic convergence. Firstly, analysing the results of both methods, we note that the choice of starting point can have a powerful

Start	x_k	$f(x_k)$	Iter.	Time	x_k	$f(x_k)$	Iter.	Time
x_1	$[1, 1]$	$3.227e-14$	7	0.005s	$[1, 1]$	$6.117e-07$	4497	0.02s
x_2	$[0.99, 0.99]$	$8.5171e-12$	20	0.02s	$[0.99, 0.99]$	$6.248e-07$	5231	0.03s
	Newton				Steepest Descent			

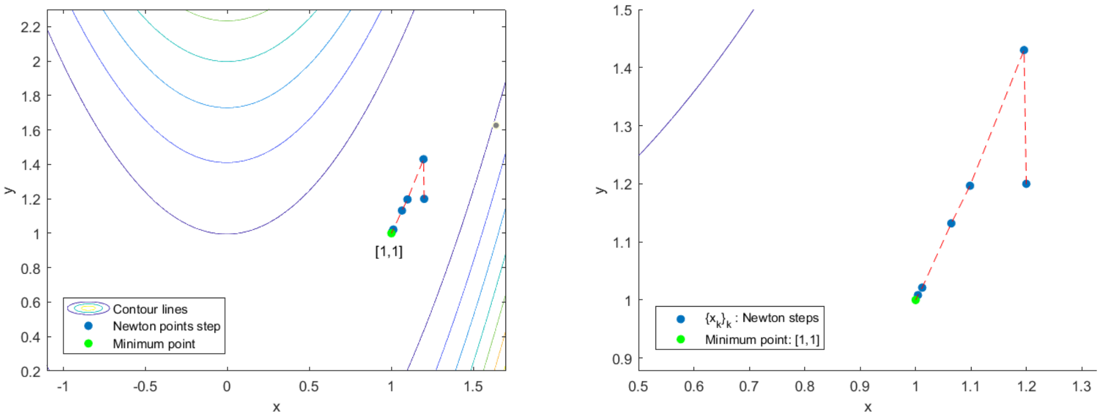
Table 1.1: Newton and SD on Rosenbrock

impact on speed of convergence: it can be seen that, starting from x_1 there is a faster convergence to the solution. This happens because x_1 is closer than x_2 to the solution $[1, 1]$. Furthermore, comparing the methods, Newton is significantly faster in terms of iterations than the steepest descent: infact, as we can also see from the Figure 1.1 (b) and Figure 1.2 (b), which explain the case with x_1 as starting point, the number of iterations used, to reach the solution, by the steepest descent method is considerably greater than those used by Newton method.



(a) *Contour lines of Rosenbrock with steepest descent* (b) *Focus on steepest descent steps*

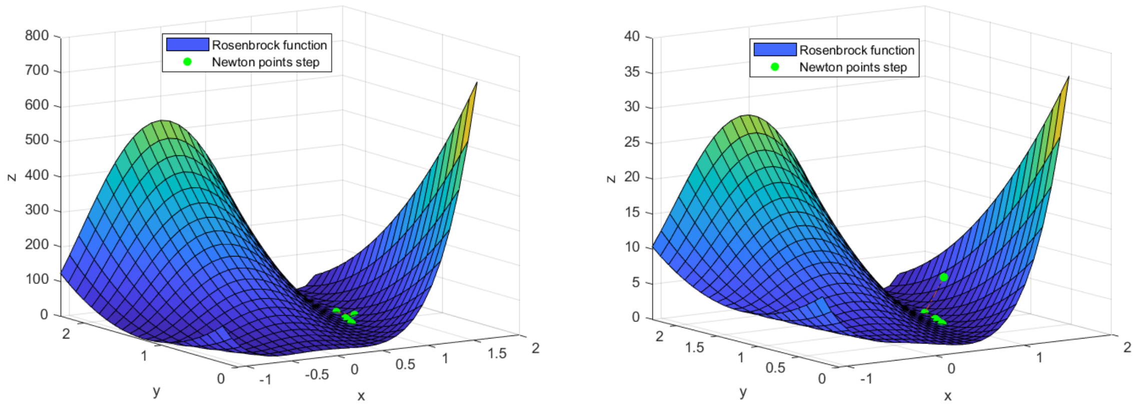
Figure 1.1: Steepest descent method on Rosenbrock with starting point $x_0 = [1.2, 1.2]$



(a) *Contour lines of Rosenbrock with Newton* (b) *Focus on Newton steps*

Figure 1.2: Newton method on Rosenbrock with starting point $x_0 = [1.2, 1.2]$

Lastly, to have a closer look at the Newton method and its steps, we plot also a *3d - view* of the Rosenbrock function 1.1, focusing on how the method works. In particular, the Figure 1.3 (b) illustrates the approach, step by step, to the minimum point of Newton, highlighting the starting point $x_0 = [1.2, 1.2]$ and each one of the x_k, calculated at each iteration.



(a) *3D - Rosenbrock with Newton steps* (b) *Focus on Newton steps*

Figure 1.3: Newton method on 3D - Rosenbrock with starting point $x_0 = [1.2, 1.2]$

2 Test function ϕ - Problem 16

This time, we'll analyze the Problem 16 (A. 5.4), explained by this following function

$$\phi(x) = \frac{1}{2}\sum_{i=1}^{n} i[(1 - cosx_i) + sinx_{i-1} - sinx_{i-1}] \quad s.t. \quad x_0 = x_{n+1} = 0, \tag{2.1}$$

with our selected methods, starting from the points

$$x_1 = [1\ 1\ \cdots\ 1], \quad x_2 = [\underbrace{-50\ \ \cdots - 50}_{n/2}\ \underbrace{-45\ \cdots\ \ -45}_{n/2}] \quad \text{and} \quad x_3 = [-100\ -100\ \cdots\ -100].$$

We obtain the following results:

Start	$f(x_k)$	$\|\|\nabla f(x_k)\|\|$	Iter.	Time	$f(x_k)$	$\|\|\nabla f(x_k)\|\|$	Iter.	Time
x_1	−427.4045	0.00098758	788	6.24s	−427.4045	0.00098624	2307	0.70s
x_2	−427.4045	0.00099909	926	3.60s	−427.4045	0.00096882	1837	1.30s
x_3	586427.09	17994.3714	5000	26.9s	−427.4045	0.00097526	2126	0.42s
	Newton				Steepest Descent			

Table 2.1: Newton and SD on Problem 16

Similarly to the Rosenbrock function 1.1, the Table 2.1 shows how the steepest descent method always converges to the minimum point, but with a high number of iterations. Concerning the other method, the convergence actually depends on the initial point, since the Newton method has the property of local convergence. It means that: if the method begins far from the solution, the success of the method is not guaranteed. In fact, considering x_1 or x_2 as initial points, the Newton method converges quickly if compared to the steepest descent. Nevertheless, starting from the last point x_3, Newton does not converge to the exact solution in the imposed iteration limit.

Moreover, the tests on this function underline one of the Newton method aspect: even if, in a convergence case, it reaches the solution very quickly (i.e. low number of iterations), the computational time of Newton method is higher than other numerical methods (as, for instance, in our case, the steepest descent). Indeed one fundamental step in Newton is the computation of the Hessian matrix, which is a step that needs

Start	Iter.	Computational time	Iter.	Computational time
x_1	788	6.24s	2307	0.70s
x_2	926	3.60s	1837	1.30s
	Newton		Steepest Descent	

Table 2.2: Newton vs steepest descent - Focus on computational cost

a very high computational cost. We can observe this aspect in the Table 2.2 comparing the computational times of the methods with initial points x_1 and x_2. Despite the lower number of iterations, the Newton method needs more computational time than the steepest descent method (which needs only to compute the gradient and not the hessian matrix).

3 Test function ψ - Problem 27

Let's start this new test with the definition of the Problem 27 function (A. 5.7)

$$\begin{aligned}
\psi(x) &= \frac{1}{2}\sum_{k=1}^{n+1}\psi_k^2(x) \\
\psi_k(x) &= \frac{1}{\sqrt{100000}}(x_k - 1) \quad \text{for } \ 1 \leq k \leq n \\
\psi_{n+1}(x) &= \sum_{i=1}^{n} x_i^2 - \frac{1}{4}
\end{aligned} \tag{3.1}$$

and the selected starting points for the analysis as follows:

$$x_1 = [1\ 2\ 3\ \cdots\ n], \quad x_2 = [1\ \ \cdots\ 1] \quad \text{and} \quad x_3 = [\underbrace{-10\ \ \cdots -10}_{n/2}\ \underbrace{20\ \cdots\ 20}_{n/2}].$$

Start	$f(x_k)$	$\|\|\nabla f(x_k)\|\|$	Iter.	Time	$f(x_k)$	$\|\|\nabla f(x_k)\|\|$	Iter.	Time
x_1	0.0048431	$2.2546e-09$	41	0.95s	0.0048431	$9.9951e-07$	8110	2.12s
x_2	0.0048431	$2.9513e-08$	14	0.70s	0.0048431	$3.9081e-07$	7	0.03s
x_3	0.0048431	$6.4233e-07$	39	0.88s	0.0048431	$1.5698e-06$	10000	2.31s
	Newton				Steepest Descent			

Table 3.1: Newton and SD on Problem 27.

In the Table 3.1 we can see the results of the tests on the function 3.1. Firstly, we can see that, for each initial point, both method reach the minimum point. However, similarly to the previous cases, the convergence of the Newton method is significantly faster, due to the quadratic rate of convergence. For instance if we focus on the case x_1 the difference is noticeable:

$$\begin{aligned}
\text{Newton method} &\longrightarrow \ \text{Number of iterarions} = 41 \\
\text{Steepest descent method} &\longrightarrow \ \text{Number of iterarions} = 8110
\end{aligned}$$

In fact, in case of convergence of both methods, the steepest descent is less convenient of Newton, in terms of number of iterations, in particular when we don't start from an initial point closer to the solution, as in x_1 and x_3 case.

4 Test function ξ - Problem 28

As before, we start our last test defining the Problem 28 function (A. 5.10)

$$\begin{aligned} \xi(x) &= \frac{1}{2}\sum_{k=1}^{n+2} \xi_k^2(x) \\ \xi_k(x) &= x_k - 1, \quad 1 \le k \le n \\ \xi_k(x) &= \sum_{i=1}^{n} i(x_i - 1), \quad k = n+1 \\ \xi_k(x) &= (\sum_{i=1}^{n} i(x_i - 1))^2, \quad k = n+2. \end{aligned} \tag{4.1}$$

Starting from

$$x_0 = \left[1 - \frac{1}{n}, \ \ 1 - \frac{2}{n}, \ \cdots \ 1 - \frac{n-1}{n}, \ \ 0\right]$$

we note that both methods stopped after a few iterations, without reaching any convergence value (NaN), as we can see from Table 4.1. Let's analyse separately the two methods:

Method	$f(x_k)$	$\|\|\nabla f(x_k)\|\|$	Iter.
NM	NaN	NaN	1
SDM	NaN	NaN	6

Table 4.1: Newton and SD on Problem 28

- Regarding the steepest descent, it can be osberved that, at start, mantaining the max number of the backtracking iterations *btmax* (A. 5.13) equal to 50, as the other tests, we get the following function and gradient norm values

$$f(x_k) = 6.2099e + 21 \quad \|\nabla f(x_k)\| = 1.3595e + 21 \tag{4.2}$$

It can be possibile to observe that in each of the first 4 iterations, the method runs the backtracking 'while cycle' reaching always the max number of *btmax*. Immediately afterwards, for a clear numeric overflow problem, the method returns 'Not a Number!'. Therefore, we changed the *btmax* value, increasing it to 100, and we finally obtained the following results

$$f(x_k) = 5.2205e - 22 \quad ||\nabla f(x_k)|| = 5.9039e - 07 \tag{4.3}$$

reaching the solution with 25 iterations, in less than a second.

- Concerning Newton, instead, any kind of attempt to modify parameters leads always to the same result: it never converges to the solution. This happens because the Hessian matrix is not actually positive definite, therefore the Newton method does not work. Indeed, in order to guarantee

$$p_k = -(\nabla^2 f(x_k))^{-1} \cdot \nabla f(x_k)$$

to be a descent direction, we have to ask the SPD property for the Hessian matrix $\nabla^2 f(x_k)$.

Moreover, we tried with severals points, as for instance

$$x_2 = [1.2\ 1.2\ \cdots\ 1.2] \quad \text{and} \quad x_3 = [50\ 50\ \cdots\ 50]$$

obtaining these results for the SD method: In each of the cases analysed, the steepest descent fails to reach

Start	$f(x_k)$	$\|\|\nabla f(x_k)\|\|$	Iter.	Time
x_2	4.9915	3.5315	10000	17.5s
x_3	$2996e + 02$	867.992	10000	17.8s

Steepest descent

Table 4.2: SDM on Problem 28 with $btmax = 100$.

the minimum despite having touched the max number of iterations. Nevertheless, starting from a point as x_2, which is not too far from the solution, the method manages to approach it.

5 Appendix

5.1 Newton method

```
function [xk, fk, gradfk_norm, k, xseq_Newton, btseq] = ...
newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
tolgrad, c1, rho, btmax)

%
% INPUTS:
% x0 = n-dimensional column vector, starting point;
% f = function handle (R^n->R);
% gradf = function handle : gradient of f;
% Hessf = function handle : Hessian of f;
% kmax = max number of iterations;
% tolgrad = value used as stopping criterion w.r.t. the norm of the
% gradient;
% c1 = the factor of the Armijo condition (a scalar in (0,1));
% rho =fixed factor, lesser than 1, used for reducing alpha0;
% btmax =maximum number of steps for 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 armijo condition
    farmijo = @(fk, alpha, gradfk, pk) ...
```

```
    fk + c1 * alpha * gradfk' * pk;
% Initializations
xseq_Newton = 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;
    alpha = 1;
    % Compute new xk
    xnew = xk + alpha * pk;
    % Compute f in the new xk
    fnew = f(xnew);
    bt = 0;
    % Backtracking strategy:
    while bt < btmax && fnew > farmijo(fk, alpha, gradfk, pk)
        alpha = rho * alpha;
        xnew = xk + alpha * pk;
        fnew = f(xnew);
        %counter backtracking iter.
        bt = bt + 1;
    end
    % Update xk, fk, gradfk_norm
    xk = xnew;
    fk = fnew;
    gradfk = gradf(xk);
    gradfk_norm = norm(gradfk);
    % Counter step
    k = k + 1;
    % Storing actual xk
    xseq_Newton(:, k) = xk;
    % Storing actual bt
 btseq(k) = bt;
end
% xseq and btseq to the correct size
xseq_Newton = xseq_Newton(:, 1:k);
```

```
    btseq = btseq(1:k);

    end
```

5.2 Steepest descent

```
function[xk,fk,gradfk_norm,k,xseq_sd,bt,bt_seq]=...
   steepest_desc_bcktrck(x0,f ,gradf ,alpha0,kmax,tolgrad ,c1,rho,btmax)

   % INPUTS:
   % x0 = n-dimensional column vector, starting point
   % f = function handle
   % gradf = function handle : gradient of f
   % alpha0= start value of alpha for backtracking strategy
   % kmax = max number of iterations
   % tolgrad = tolerance value respect to the norm of the gradient
   % c1 = the factor of the Armijo condition (a scalar in (0,1))
   % rho =fixed factor, lesser than 1, used for reducing alpha
   % btmax =maximum number of steps for the backtracking strategy.
   %
   % OUTPUTS :
   % xk: solution computed by the method, i .e. last vector
   % fk: the value f(xk)
   % gradfk norm : the norm of gradient of f(xk)
   % k: number of iterations executed
   % xseq_sd : vector of all xk of each step of method
   %bt: last value of bt (number of executing times of backtracking)
   %bt_seq: all the number of backtracking iterations
   %at each optimization step.

       % Armijo Condition
       farmijo=@(fk ,alpha ,gradfk ,pk)fk+c1*alpha*gradfk'*pk;
       % Initializations
       xseq_sd=zeros(length(x0),kmax);
       bt_seq=zeros(1,kmax);
       xk=x0;
       fk=f(xk);
       gradfk=gradf(xk);
       gradfk_norm=norm(gradf(xk));
```

```
        k=0;
        while k<kmax && gradfk_norm>tolgrad
         % Compute the descent direction
            pk=-gradf(xk);
            alpha=alpha0;
            xknew=xk+alpha*pk;
            fknew=f(xknew);
            bt=0;
         % Backtracking strategy
         while fknew > farmijo(fk ,alpha ,gradfk ,pk) && bt<btmax
                alpha=alpha*rho;
                xknew=xk+alpha*pk;
                fknew=f(xknew);
                bt=bt+1;
                %bt_seq=bt_seq+bt;
            end
        xk=xknew;
        fk=fknew;
        gradfk=gradf(xk);
        gradfk_norm=norm(gradfk);
        k=k+1;
        xseq_sd(:,k)=xk;
        bt_seq(k) = bt;
        end
        fk=f(xk) ;
xseq_sd=xseq_sd(:,1:k);
end
```

5.3 Rosenbrock test

```
alpha0=1;
btmax=50;
c1=1e-4;
kmax = 10000;
rho=0.5;
tolgrad=1e-3;

%starting points
x0=[1.2;1.2];  % x1
%x0=[-1.2;1];  % x2
```

```
% Define Rosenbrock Function f, gradf, Hessf
f=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
gradf=@(x) [-400*(x(1)*x(2)-x(1)^3)-2*(1-x(1));200*(x(2)-x(1)^2)];
Hessf=@(x) [1200*x(1)^2-400*x(2)+2,-400*x(1);-400*x(1),200];

% STEEPEST DESCENT ON ROS
tic
[xk_SD,fk_SD,gradfk_norm_SD,k_SD,xseq_SD]=...
   steepest_desc_bcktrck(x0,f,gradf,alpha0,kmax,tolgrad,c1,rho,btmax);
toc

disp('STEEPEST DESCENT')
disp(['xk_SD: ', mat2str(xk_SD), '(min point: [1;1]);'])
disp(['f(xk_SD): ', num2str(fk_SD), '(min: 0);'])
disp(['norm gradient: ', num2str(gradfk_norm_SD)])
disp(['Iterations: ', num2str(k_SD) , '/' ,num2str(kmax), ';'])


% NEWTON METHOD ON ROS

tic
[xk, fk, gradfk_norm, k, xseq_Newton, btseq] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax)
toc

disp('NEWTON METHOD')
disp(['xk: ', mat2str(xk), '(min point: [1;1]);'])
disp(['f(xk): ', num2str(fk), '(min: 0);'])
disp(['norm gradient: ', num2str(gradfk_norm)])
disp(['Iterations: ', num2str(k) , '/' ,num2str(kmax), ';'])
```

5.4 Problem 16 - Function

```
function[f]=function_problem16(x)
    n=length(x);
    f=0;
    %i=1
    f=f+1-cosd(x(1,:))-sind(x(2,:));
```

```
    %i=n
    f=f+n*(1-cosd(x(n,:))+sind(x(n-1,:)));
    for i=2:n-1
        f=f+i*(1-cosd(x(i,:))+sind(x(i-1,:))-sind(x(i+1 ,:)));
    end
    end
```

5.5 Problem 16 - Gradient

```
function [gradf]=grad_problem16(x)
    n=length(x) ;
    gradf=zeros(1,n) ;
        for i=1:n-1
            gradf(i)=2*cosd(x(i,:))+i*sind(x(i ,:));
        end
        gradf(n)=(1-n)*cosd(x(n,:))+n*sind(x(n,:));
    end
```

5.6 Problem 16 - Hessian

```
function [hessf]=hess_problem16(x)
    n=length (x) ;
    hessf=zeros(n,n);
    for i=1:n-1
        hessf(i,i)=i*cosd(x(i,:))-2*sind(x(i,:));
    end
    hessf(n,n)=n*cosd(x(n,:))+(n-1)*sind(x(n,:));
```

5.7 Problem 27 - Function

```
 function[f]=function_problem27(x)
    n=length(x);
    f=0;
    W=1/sqrt(100000);

    fk=@(x,k) W*(x(k, :)-1);
    %k=n+1
    f=f+0.5*(((norm(x)^2)-0.25)^2);
```

```
    for k=1:n
      f=f+0.5*(fk(x,k)^2);
    end
    end
```

5.8 Problem 27 - Gradient

```
function[gradf]=grad_problem27(x)
    n=length(x);
    gradf=zeros(1,n);
    W=1/sqrt(100000);
    fk=@(x,k) W*(x(k,:)-1);

    fy=(norm(x)^2)-0.25;

    for k=1:n
        gradf(k)=fk(x,k)*W+2*x(k,:)*fy;
    end
    end
```

5.9 Problem 27 - Hessian

```
function[Hessf] = hess_problem27(x)
     n = length(x);
     Hessf = zeros(n,n);
     W = 1/sqrt(100000);
     fy= (norm(x)^2)-0.25;
     for i=1:n
         for j=1:n
             if i==j
                 Hessf(i,i) = (W^2)+4*(x(i,:)^2)+2*fy;
             else
                 Hessf(i,j) = 4*x(i,:)*x(j,:);
             end
         end
     end
     end
```

5.10 Problem 28 - Function

```
function[f]=function_problem28(x)
    n=length(x);
    f=0;
    sum = 0;

    fk=@(x,k) (x(k)-1);

    for i=1:n
        sum = sum + i*fk(x,i);
    end
    sumq = sum^2;

    for k=1:n
        if k<=n
            f=f+0.5*(fk(x,k)^2);
        end
    end
    % adding k=n+1 and k=n+2 terms
    f = f + 0.5*sumq + 0.5*(sumq)^2;
    end
```

5.11 Problem 28 - Gradient

```
function[gradf]=grad_problem28(x)
    n=length(x);
    gradf=zeros(1,n);
    sum = 0;

    fk=@(x,k) (x(k)-1);
    for j =1:n
        sum = sum + j*(fk(x,j));
    end

    for i=1:n
        gradf(i)= fk(x,i)+ i*sum + 2*i*(sum^3);
    end
```

5.12 Problem 28 - Hessian

```
function[Hessf] = hess_problem28(x)
    n = length(x);
    Hessf = zeros(n,n);
    sum = 0;

    fk = @(x,k) k*(x(k)-1);
    for k=1:n
    sum = sum + fk(x,k);
    end
    sumq = sum^2;
    for i=1:n
        for j=1:n
        if i==j
                Hessf(i,i) = 1+(i)^2 + 6*(i)^2*sumq;
        else
                Hessf(i,j) = i*j + 6*(i)*(j)*sumq;
        end
    end
    end
end
```

5.13 Test

```
%%INITIALISE VARIABLES OF THE PROBLEM
alpha0=1;
btmax=50; %btmax=100;
c1=1e-4;
rho=0.5;
tolgrad=1e-6; %tolgrad=1e-3
kmax=10000;
n=1e3;

%initializing the starting point and function, gradient
%and hessian for each problem

%% PROBLEM 16

%starting point
```

```
%x0=ones(1,n)';  % x1
%x0=[-50*ones(1,n/2), -40*ones(1,n/2)]';  % x2
x0 = -100*ones(1,n)'; %x3

%function, gradient and hessian
f=@(x) function_problem16(x);
gradf=@(x) grad_problem16(x)';
Hessf=@(x) hess_problem16(x);

%% PROBLEM 27

%starting point
%x0=zeros(1,n)';
%for j =1:n
    %x0(j)=j;  % x1
%end
%x0=ones(1,n)';  % x2
x0=[-10*ones(1,n/2), 20*ones(1,n/2)]';  % x3

%function, gradient and hessian
f=@(x) function_problem27(x);
gradf=@(x) grad_problem27(x)';
Hessf=@(x) hess_problem27(x);

%% PROBLEM 28

%starting points
%x0=zeros(1,n)';
%for j =1:n
   %x0(j)=1-j/n;  % x1
%end
%x0=50*ones(1,n)';  % x2
x0 = 1.2*ones(1,n)'; %x3

f=@(x) function_problem28(x);
gradf=@(x) grad_problem28(x)';
Hessf=@(x) hess_problem28(x);

%% METHOD
%RUN THE STEEPEST DESCENT
tic
```

```
[xk_SD,fk_SD,gradfk_norm_SD,k_SD,xseq_SD] = steepest_desc_bcktrck(x0,f,...
    gradf,alpha0,kmax,tolgrad,c1,rho,btmax);
toc

disp('STEEPEST DESCENT')
disp(['xk_SD: ', mat2str(xk_SD)])
disp(['gradfk_norm_SD: ', num2str(gradfk_norm_SD)])
disp(['f(xk_SD): ', num2str(fk_SD)])
disp(['Iterations: ', num2str(k_SD), '/', num2str(kmax)])

% RUN NEWTON METHOD
tic
[xk, fk, gradfk_norm, k, xseq, btseq] = ...
    newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax);
toc

disp('NEWTON METHOD')
disp(['xk: ', mat2str(xk)])
disp(['gradfk_norm: ', num2str(gradfk_norm)])
disp(['f(xk): ', num2str(fk)])
disp(['Iterations: ', num2str(k), '/', num2str(kmax), ';'])
```

5.14 3D-Figure

```
figure
x = -1.1:0.1:1.7;
y = 0.2:0.1:2.3;
[X,Y] = meshgrid(x,y);
Z = 100*(Y-X.^2).^2 + (1-X).^2;
surf(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')
hold on
a = [x0(1) xseq_Newton(1, :)];
b = [x0(2) xseq_Newton(2, :)];
z = 100*(b-a.^2).^2 + (1-a).^2;
scatter3(a,b,z,'filled','g')
hold on
```

```
plot3(a,b,z,'-.')
hold on
legend('Rosenbrock function','Newton points step')
```

5.15 Newton Figure

```
figure
%Creation of the meshgrid for the contourlines
[X1, X2] = meshgrid(linspace(-1.1, 1.7, 500), linspace(0.2, 2.3, 500));
% Computation of the values of f for each point of the mesh
Z_Rosenbrock = 100*(X2-X1.^2).^2 + (1-X1).^2;
% Contour lines
[C,~] = contour(X1, X2, Z_Rosenbrock);
% plot of the points in xseq_Newton
hold on
scatter([x0(1) xseq_Newton(1, :)], [x0(2) xseq_Newton(2, :)], 'filled')
hold on
scatter(1,1,'filled','g')
hold on
plot([x0(1) xseq_Newton(1, :)], [x0(2) xseq_Newton(2, :)], '--',Color='r');
hold on
text(0.9,0.9,'[1,1]')
hold on
legend('Contour lines','Newton points step','Minimum point')
xlabel('x')
ylabel('y')
hold off
```

5.16 Steepest descent Figure

```
figure
% Creation of the meshgrid for the contourlines
[X1, X2] = meshgrid(linspace(-1.1, 1.7, 500), linspace(0.2, 2.3, 500));
% Computation of the values of f for each point of the mesh
Z_Rosenbrock = 100*(X2-X1.^2).^2 + (1-X1).^2;
% Contour lines
[C,~] = contour(X1, X2, Z_Rosenbrock);
% plot of the points in xseq_Newton
hold on
```

```
scatter([x0(1) xseq_sd(1, :)], [x0(2) xseq_sd(2, :)],'x')
hold on
scatter(1,1,'filled','g')
hold on
plot([x0(1) xseq_sd(1, :)], [x0(2) xseq_sd(2, :)], '--',Color='r');
hold on
text(0.9,0.9,'[1,1]')
hold on
legend('Contour lines','S.d. points step','Minimum point')
xlabel('x')
ylabel('y')
hold off
```