Unconstrained Optimization

Steepest descent, Newton and finite difference Newton methods

Rachid El Amrani
Giorgia Rosalia Buccelli

NB For further informations, visit ResearchGate Website here

1 Introductory analysis of the problem

The unconstrained optimization problem considered is

$$\min_{x \in \mathbb{R}^n} 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Therefore, we want to minimize (locally) the smooth (C^2) function $f : \mathbb{R}^2 \to \mathbb{R}$ defined as

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

over all $x \in \mathbb{R}^2$. One way of solving this problem is using the Newton method, whose basic idea is, at every current iterate x_k, to locally approximate f with a quadratic model and, if $\nabla^2 f(x_k)$ is positive definite, to find the descent direction p_k minimizing the local model. Then, to compute $x_{k+1} = x_k + \alpha_k p_k$, that is the next iterate is found moving along a line in the p_k direction and with a step α_k to be determined. More precisely, the quadratic model is given by the second order Taylor expansion of f around x_k:

$$m_k(p) = f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T \nabla^2 f(x_k) p$$

and the descent direction p_k, found minimizing it, is solution of the linear system

$$\nabla^2 f(x_k) p = -\nabla f(x_k)$$

The method is iterated until a stopping criterion is satisfied, such as the norm of the gradient in x_k is under a chosen tolerance or a chosen maximum number of iterations is reached.
In our case, we have that the gradient of the function to minimize is

$$\nabla f(x) = \left(-400x_1(x_2 - x_1^2) - 2(1 - x_1), \quad 200(x_2 - x_1^2)\right)$$

and so the true minimum will be $(1, 1)$, and the Hessian matrix is

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} -400x_2 + 1200x_1^2 + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}$$

Therefore, the Hessian matrix is positive definite for every x and so the quadratic model for f around x_k is always convex and we can apply the Newton method as explained before. To determine the suitable steplength α_k at each step k of the method, in implementations is used the *backtracking strategy*, which consists in looking for a value α_k satisfying the Armijo condition

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

where $c_1 \in (0, 1)$ (and often $c_1 = 10^{-4}$). To find such a value of α_k, is chosen an initial steplength $\alpha_k^{(0)}$ and if it satisfies the Armijo condition, then it is accepted, otherwise it is decreased multiplying it by a chosen factor $\rho \in (0, 1)$ until the Armijo condition is satisfied (and

then this value is accepted). To solve our problem we will hence use the Newton method with backtracking line-search. As seen, to use it is necessary to have the gradient and the Hessian matrix of the function. In our case, we know the exlicit expressions of them, because we know the function analitically, but we can also approximate them using finite differences, to compare the two implementations.
The purpose of this report is in fact to solve the problem in two cases, one using $n = 10^4$ for the dimension of the domain and the other using $n = 10^5$, implementing the Newton method with backtracking line-search in two ways: one using exact derivatives and the other approximating the gradient and the Hessian matrix with finite differences.
Moreover, in the finite differences Newton method will be tested different values for the increment h: for $i = 2, 4, 6, 8, 10, 12, 14$, the increment is $h = 10^{-i}\|\hat{x}\|$, where $\hat{x}$ is the point at which the derivatives have to be approximated. At the end, a comparison will be made in terms of number of iterations and computing time. Before implementing the two methods an important observation is that, when n is large, to avoid storage problems with the Hessian matrix and to reduce the computing time in creating and evaluating it, one can take advantage of the knowledge of the function to minimize and of the structure of the Hessian matrix.
We recall, down below, the expressions for the approximated gradient with forward finite differences and of the Hessian matrix with centered finite differences

$$(\nabla f(x))_i \approx \frac{f(x + he_i) - f(x)}{h}$$

$$(\nabla^2 f(x))_{ii} \approx \frac{f(x + he_i) - 2f(x) + f(x - he_i)}{h}$$

2 Rosenbrock function

The *Rosenbrock function* is the following function:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

The optimal solution is obviously $(x_1, x_2) = (1, 1)$ with corresponding optimal value 0.

Exact Newton with Backtracking line-search

- with starting point $x_0 = (1.2, 1.2)$ the method converges to optimal solution $(1, 1)$ with optimal value $f(x_k) = 5.1956e - 25$ (≈ 0), in $k = 20$ iterations, and computational time 0.003014 seconds.
- with starting point $x_0 = (-1.2, 1)$ the method converges exactly like it did from the previous starting point except for the computation time where it takes a little bit long (0.007594 seconds).

Newton with Backtracking line-search and finite differences

- with starting point $x_0 = (1.2, 1.2)$ the method converges to optimal solution $(1, 1)$ with optimal value $f(x_k) = 6.0194e - 14$ (≈ 0), in $k = 201$ iterations and computational time seconds.
- with starting point $x_0 = (-1.2, 1)$ the method converges to optimal solution $(1, 1)$ with optimal value $f(x_k) = 6.0194e - 14$ (≈ 0), in $k = 201$ iterations and computational time seconds.

The reason why it took this much, in the second method, is due to the fact that the Rosenbrock function is extremely ill-conditioned at the optimal solution.
Indeed,

$$\nabla f(x_1, x_2) = \begin{pmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{pmatrix}$$

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} -400x_2 + 1200x_1^2 + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}$$

It is not difficult to show that $(x_1, x_2) = (1, 1)$ is the unique stationary point. In addition,

$$\nabla^2 f(1, 1) = \begin{pmatrix} 802 & -400 \\ -400 & 200 \end{pmatrix}$$

and hence the condition number is

```
A = [802, -400; -400, 200]
cond(A)

ans =
     2.5080e+03
```

A condition number of more than 2500 should have severe effects on the convergence speed of the gradient method. This run required the huge amount of 201 iterations, so the ill-conditioning effect has a significant impact.

3 Brown Function (problem 12)

The second problem we analyzed is the Brown function, which is defined as follow:

$$F(x) = \sum_{j=1}^{k} \left[(x_{i-1} - 3)^2 / 1000 - (x_{i-1} - x_i) + \exp(20(x_{i-1} - x_i)) \right] + \left(\sum_{j=1}^{k} (x_{i-1} - 3) \right)^2$$

with $i = 2j$ and $k = n/2$. The starting points $\bar{x}, \tilde{x}, \hat{x}$ we used are:

$$\bar{x}_i = \begin{cases} -1 & \text{if } i \text{ even,} \\ 0 & \text{if } i \text{ odd} \end{cases}, \quad \tilde{x}_i = \begin{cases} -2 & \text{if } i \text{ even,} \\ 0 & \text{if } i \text{ odd} \end{cases}, \quad \hat{x}_i = \begin{cases} 1 & \text{if } i \text{ even,} \\ 0 & \text{if } i \text{ odd} \end{cases}$$

To compare the implementations, we built two tables summarizing the results, one table for the dimension of the domain equal to $n = 10^4$ and the other for $n = 10^5$. Each table reports for every method the number of iterations achieved, the computing time and the value of the minimum found with that method, truncated to the sixth decimal place.

Newton with finite differences

x0(j) = -1, j even				n = 10^4			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	10/100	6/100	6/100	7/100	5/100	5/100
Computational time	5.521665 s	0.216982 s	0.137441 s	0.147756 s	0.225675 s	0.102827 s	0.127344 s
Minimum	-1,1724786	-0,6857532	-0,6823619	-0,6823281	-0,68232781	-0,6823278	-0,6823225

x0(j) = -1, j even				n = 10^5			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	12/100	8/100	8/100	7/100	8/100	7/100
Computational time	51.944679 s	2.401929 s	1.514887 s	1.818758 s	1.834647 s	3.482763 s	3.106046 s
Minimum	-447,98379	-0,6932549	-0,6824357	-0,6823289	-0,68232781	-0,6823278	-0,6823248

x0(j) = -2, j even				n = 10^4			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	11/100	7/100	6/100	24/100	6/100	7/100
Computational time	2.798070 s	0.425828 s	0.215117 s	0.181712 s	1.589176 s	0.218982 s	0.315066 s
Minimum	-1,1795090	-0,6857532	-0,6823619	-0,6823281	-0,68232781	-0,6823278	-0,6823384

x0(j) = -2, j even				n = 10^5			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	13/100	8/100	9/100	7/100	10/100	5/100
Computational time	89.539068 s	6.614855 s	4.232658 s	6.165665 s	4.396124 s	7.684626 s	2.701416 s
Minimum	-1.865,6791	-0,6932549	-0,6824357	-0,6823289	-0,68232781	-0,6823278	-0,682328

x0(j) = +1, j even				n = 10^4			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	100/100	100/100	100/100	100/100	100/100	97/100
Computational time	8.033088 s	8.122656 s	7.948274 s	7.991562 s	8.254008 s	8.341066 s	7.939431 s
Minimum	-1,1488274	-0,6857476	-0,6823573	-0,6823235	-0,68232781	-0,6823231	-0,6823152

x0(j) = +1, j even				n = 10^5			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	100/100	100/100	100/100	100/100	100/100	100/100
Computational time	69.123053 s	100.63084 s	116.75297 s	84.862639 s	80.135336 s	80.117024 s	71.564518 s
Minimum	-97,4854204	-0,6932464	-0,6824311	-0,6823243	-0,68232319	-0,6823232	-0,6823215

Looking at the number of iterations, we can see that for each table the finite differences Newton method with $i = 2$ has reached the maximum number allowed and then of course the computing time is higher than the others, especially for the case $n = 10^5$. Also the value of the minimum found is pretty different from the others.

The reason of this bad behaviour is that with this value of i the increment h is not sufficiently small and so the approximations of the gradient and of the Hessian matrix are not good.

To see that, we can observe that $h = 10^{-2}||\hat{x}||$ valued in $\hat{x} = x_0$ vector of all $x_0(i) = -1$ (and $x_0(i) = 1$) is $h = 10^{-2}\sqrt{n}$, and so for $n = 10^4$ we have $h = 1$ and for $n = 10^5$ we have $h = \sqrt{10}$, which are too large values for the increment. As a result of this poor approximation, the method doesn't converge to the real minimum.

We can observe that $h = 10^{-2}||\hat{x}||$ valued in $\hat{x} = x_0$ vector of all $x_0(i) = -2$ is $h = 10^{-2}2\sqrt{n}$, and so for $n = 10^4$ we have $h = 2$ and for $n = 10^5$ we have $h = 2\sqrt{10}$, which are too large values for the increment. As a result of this poor approximation, the method doesn't converge to the real minimum.

For all the other values of i tested, the number of iterations is 100, the chosen threshold, and the minimums found are quite similar.

Steepest Descent

For this method we use as fixed parameters $\alpha_0 = 5$, $btmax = 50$, $c_1 = 10^{-4}$, $\rho = 0.5$, a tolerance of $1.0e^{-5}$, $h = \sqrt{\epsilon_m}||\mathbf{x}||$, and FDgrad='c', which means we use the centered finite differences method to compute the gradient of the function. The results we have accomplished, for different starting points, number of iterations and dimension of the problem, are the following:

starting point	dimension	$kmax$	$f(x_0)$	$f(x_k)$	k	time
$\bar{x}$	10^4	10^3	2,4258e+11	8,6835e+04	k_{max}	5,754
$\bar{x}$	10^4	10^4	2,4258e+11	7,2943e+04	k_{max}	378,533
$\bar{x}$	10^4	10^5	2,4258e+11	99,8933	k_{max}	5.308,953
$\bar{x}$	10^5	10^2	2,42605e+12	4,31743e+05	k_{max}	452,432
$\bar{x}$	10^5	10^3	2,42605e+12	4,30733e+05	k_{max}	3.840,403

We notice that for $n = 10^5$, the method takes the whole number of iterations to reach the minimum and accordingly enough computation time. For the the other values however, we notice that the method doesn't converge even when it reaches the maxmimum number of iterations.

starting point	dimension	$kmax$	$f(x_0)$	$f(x_k)$	k	time
$\tilde{x}$	10^4	10^3	1,1769e+20	1,0449e+07	k_{max}	38,948
$\tilde{x}$	10^4	10^4	1,1769e+20	1,0435e+07	k_{max}	332,357
$\tilde{x}$	10^4	10^5	1,1769e+20	1,0296e+07	k_{max}	2.968,300
$\tilde{x}$	10^5	10^2	1,1769e+21	1,0453e+08	k_{max}	434,054
$\tilde{x}$	10^5	10^3	1,1769e+21	1,0452e+08	k_{max}	3.9222,890

We notice that, independently of the dimension, the method converges to the same minimum reaching the maximum number of iterations and accordingly taking enough time.

starting point	dimension	$kmax$	$f(x_0)$	$f(x_k)$	k	time
$\hat{x}$	10^4	10^5	2,2505e+06	99,8933	346	26,360
$\hat{x}$	10^5	10^2	2,25005e+08	6,3984e+03	k_{max}	1.042,70
$\hat{x}$	10^5	10^3	2,25005e+08	5,38808e+03	k_{max}	8.295,867

For this starting point, we can observe that the method still takes the maximum number of iterations except for $n = 10^4$ without converging.

4 Banded trigonometric function (problem 16)

The third problem we analyzed is the Brown function, which is defined as follow:

$$F(x) = \sum_{k=1}^{n} i\left[(1 - \cos x_i) + \sin x_{i-1} - \sin x_{i+1}\right]$$

with $x_0 = x_{n+1} = 0$. The starting points $\bar{x}, \tilde{x}, \hat{x}$ we used are:

$$\bar{x}_i = 1, \quad \tilde{x}_i = 1/2, \quad \hat{x}_i = 3, \quad \forall i \geq 1$$

To compare the implementations, we built two tables summarizing the results, one table for the dimension of the domain equal to $n = 10^4$ and the other for $n = 10^5$. Each table reports for every method the number of iterations achieved, the computing time and the value of the minimum found with that method, truncated to the sixth decimal place.

Newton with finite differences

x0(j) = 1				n = 10^4			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	100/100	100/100	100/100	100/100	100/100	85/100
Computational time	4.785905 s	5.934816 s	5.926383 s	5.881361 s	6.072661 s	6.137113 s	5.130632 s
Minimum	-1,1550909	-0,6857478	-0,6823572	-0,6823234	-0,6823231	-0,6823231	-0,6823111

x0(j) = 1				n = 10^5			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	100/100	100/100	100/100	100/100	100/100	93/100
Computational time	59.942896 s	57.281979 s	57.372946 s	58.489673 s	58.268828 s	58.098811 s	54.008985 s
Minimum	-7348,5282	-0,6932476	-0,682431	-0,6823242	-0,6823231	-0,6823231	-0,6823171

x0(j) = 1/2				n = 10^4			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	100/100	100/100	100/100	100/100	100/100	82/100
Computational time	4.714346 s	5.820057 s	5.823699 s	5.928033 s	6.095671 s	6.112395 s	5.023237 s
Minimum	-1,1550727	-0,6857502	-0,6823593	-0,6823255	-0,6823252	-0,6823251	-0,6823085

x0(j) = 1/2				n = 10^5			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	100/100	100/100	100/100	100/100	100/100	93/100
Computational time	58.627966 s	73.766435 s	57.696266 s	68.149253 s	57.094169 s	57.888305 s	66.188321 s
Minimum	-4279,4596	-0,6932507	-0,6824331	-0,6823263	-0,6823252	-0,6823252	-0,6823232

x0(j) = 3				n = 10^4			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	100/100	100/100	100/100	100/100	100/100	88/100
Computational time	6.264862 s	7.623657 s	7.570333 s	7.684897 s	7.947349 s	7.654255 s	6.885262 s
Minimum	-1,1589637	-0,6857468	-0,6823561	-0,6823224	-0,6823220	-0,6823221	-0,6823088

x0(j) = 3				n = 10^5			
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	100/100	100/100	100/100	100/100	100/100	100/100	97/100
Computational time	128.43825 s	69.463164 s	69.647954 s	69.393777 s	69.193194 s	69.375918 s	67.490473 s
Minimum	-33.287,00	-0,6932458	-0,6824299	-0,6823231	-0,682322	-0,682322	-0,682318

Looking at the number of iterations, we can see that for each table the finite differences Newton method with $i = 2$ has reached the maximum number allowed and then of course the computing time is higher than the others, especially for the case $n = 10^5$. Also the value of the minimum found is pretty different from the others.

The reason of this bad behaviour is that with this value of i the increment h is not sufficiently small and so the approximations of the gradient and of the Hessian matrix are not good.

To see that, we can observe that $h = 10^{-2}||\hat{x}||$ valued in $\hat{x} = x_0$ vector of all ones (our initial point) is $h = 10^{-2}\sqrt{n}$, and so for $n = 10^4$ we have $h = 1$ and for $n = 10^5$ we have $h = \sqrt{10}$, which are too large values for the increment. As a result of this poor approximation, the method doesn't converge to the real minimum.

We can observe that $h = 10^{-2}||\hat{x}||$ valued in $\hat{x} = x_0$ vector of all 1/2 is $h = 10^{-2}\frac{\sqrt{n}}{2}$, and so for $n = 10^4$ we have $h = 1/2$ and for $n = 10^5$ we have $h = \frac{1}{2}\sqrt{10}$, which are too large values for the increment. As a result of this poor approximation, the method doesn't converge to the real minimum.

We can observe that $h = 10^{-2}||\hat{x}||$ valued in $\hat{x} = x_0$ vector of all 3 is $h = 10^{-2}3\sqrt{n}$, and so for $n = 10^4$ we have $h = 3$ and for $n = 10^5$ we have $h = 3\sqrt{10}$, which are too large values for the increment. As a result of this poor approximation, the method doesn't converge to the real minimum.

For all the other values of i tested, the number of iterations is 100, the chosen threshold, and the minimums found are quite similar.

Steepest Descent

For this method we use as fixed parameters $\alpha_0 = 5$, $btmax = 50$, $c_1 = 10^{-4}$, $\rho = 0.5$, a tolerance of $1.0e^{-5}$, $h = \sqrt{\epsilon_m}||\mathbf{x}||$, and FDgrad='c', which means we use the centered finite differences method to compute the gradient of the function. The results we have accomplished, for different starting points, number of iterations and dimension of the problem, are the following:

starting point	dimension	$kmax$	$f(x_0)$	$f(x_k)$	k	time
$\bar{x}$	10^4	10^3	2,30919e+05	-428,100	k_{max}	165,986
$\bar{x}$	10^4	10^4	2,30919e+05	-428,168	k_{max}	1.812,583
$\bar{x}$	10^4	10^2	2,29956e+07	-545,752	k_{max}	1.737,224
$\bar{x}$	10^5	10^3	2,29956e+07	-4,157e+03	k_{max}	17.627,437

We notice that for $kmax < 10^3$ the method doesn't converge. Besides, the method requires only $kmax = 10^3$ to reach the solution in the case of $n = 10^4$ taking less time.

starting point	dimension	$kmax$	$f(x_0)$	$f(x_k)$	k	time
$\tilde{x}$	10^4	10^3	6,1749e+04	-428,147	k_{max}	170,485
$\tilde{x}$	10^4	10^4	6,1749e+04	-428,168	k_{max}	1.769,867
$\tilde{x}$	10^5	10^2	6,126278e+06	-932,035	k_{max}	1.782,895

We notice that it would be enough a dimension of $n = 10^4$ to have convergence independently of kmax with the only difference that the smaller kmax, the lesser the time spent to reach the solution.

starting point	dimension	$kmax$	$f(x_0)$	$f(x_k)$	k	time
$\hat{x}$	10^4	10^3	9,9613e+05	-428,168	k_{max}	657,951
$\hat{x}$	10^5	10^2	9,95109e+07	278,334	k_{max}	7.552,269

As previously said, we notice that we have convergence for $n = 10^4$ with always kmax and a lesser time.

5 Penalty Function (problem 27)

The first problem we analyzed is the Penalty function, which is defined as follow:

$$F(x) = \frac{1}{2}\left(\sum_{k=1}^{n}\left(\frac{1}{\sqrt{100000}}(x_k - 1)\right)\right)^2 + \left(\sum_{k=1}^{n} x_i^2 - \frac{1}{4}\right)^2$$

The starting points $\bar{x}, \tilde{x}, \hat{x}$ we used are:

$$\bar{x}_i = i, \quad \tilde{x}_i = i/2, \quad \hat{x}_i = i - 10, \quad \forall i \geq 1$$

To compare the implementations, we built two tables summarizing the results, one table for the dimension of the domain equal to $n = 10^4$ and the other for $n = 10^5$. Each table reports for every method the number of iterations achieved, the computing time and the value of the minimum found with that method, truncated to the sixth decimal place.

Newton with finite differences

x0(j) = j	n = 10^4						
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	5/100	10/100	100/100	100/100	100/100	100/100	100/100
Computational time	0.368248 s	6.488914 s	6.448412 s	6.379307 s	6.353478 s	6.398567 s	6.394592 s
Minimum	-1,72E+18	-0,685753	-0,6823619	-0,68232814	-0,68232781	-0,68232778	-0,6823278

x0(j) = j	n = 10^5						
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	4/100	45/100	100/100	100/100	100/100	100/100	100/100
Computational time	4.555052 s	11.3936 s	55.56131 s	55.1557 s	55.46293 s	57.37087 s	57.45360 s
Minimum	-4,76E+24	-0,693255	-0,6824357	-0,682329	-0,682327	-0,682328	-0,682335

x0(j) = j /2	n = 10^4						
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	6/100	100/100	100/100	100/100	100/100	100/100	100/100
Computational time	0.341460 s	4.892251 s	4.811771 s	4.827550 s	4.809355 s	4.778308 s	4.994548 s
Minimum	-1,29E+43	-0,685753	-0,6823619	0,68232814	-0,68232781	-0,68232781	-0,68232779

x0(j) = j/2	n = 10^5						
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	4/100	42/100	100/100	100/100	100/100	100/100	100/100
Computational time	3.598622 s	8.615142 s	45.630755 s	46.72577 s	45.14173 s	45.42843 s	45.53254 s
Minimum	-2,58E+24	-0,693255	-0,6824357	-0,682329	-0,682327	-0,682328	-0,682335

x0(j) = j - 10	n = 10^4						
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	5/100	100/100	100/100	100/100	100/100	100/100	100/100
Computational time	0.319544 s	4.959278 s	4.798030 s	4.782426 s	5.010224 s	8.321094 s	9.690795 s
Minimum	-3,65E+34	-0,685753	-0,6823619	-0,68232814	-0,68232781	-0,68232781	-0,68232779

x0(j) = j - 10	n = 10^5						
Method	i = 2	i = 4	i = 6	i = 8	i = 10	i = 12	i = 14
Iterations	4/100	45/100	100/100	100/100	100/100	100/100	100/100
Computational time	7.378518 s	18.00996 s	44.735028 s	78.77133 s	56.12209 s	45.42163 s	45.33254 s
Minimum	-1,14E+39	-0,693255	-0,6824357	-0,682329	-0,682327	-0,682328	-0,682335

Looking at the number of iterations, we can see that for each table the finite differences Newton method with $i = 2$ has performed fewer iterations than the other values i and then of course the computing time is smaller than the others, also the value of the minimum found is pretty different from the others.

The reason of this bad behaviour is that with this value of i the increment h is not sufficiently small and so the approximations of the gradient and of the Hessian matrix are not good.

To see that, we can observe that $h = 10^{-2}||\hat{x}||$ valued in $\hat{x} = x_0$ vector of all $i = 2$ is $h = 10^{-2}2\sqrt{n}$, and so for $n = 10^4$ we have $h = 2$ and for $n = 10^5$ we have $h = 2\sqrt{10}$, which are too large values for the increment. As a result of this poor approximation, the method doesn't converge to the real minimum.

We can observe that $h = 10^{-2}||\hat{x}||$ valued in $\hat{x} = x_0$ vector of all ones, because $x_0(i) = i/2$, is $h = 10^{-2}\sqrt{n}$, and so for $n = 10^4$ we have $h = 1$ and for $n = 10^5$ we have $h = \sqrt{10}$, which are too large values for the increment. As a result of this poor approximation, the method doesn't converge to the real minimum.

We can observe that $h = 10^{-2}||\hat{x}||$ valued in $\hat{x} = x_0$ vector of $x_0(i) = i - 10$, is $h = 10^{-2}8\sqrt{n}$, and so for $n = 10^4$ we have $h = 8$ and for $n = 10^5$ we have $h = 8\sqrt{10}$, which are too large values for the increment. As a result of this poor approximation, the method doesn't converge to the real minimum.

For all the other values of i tested, the number of iterations is 100, the chosen threshold, and the minimums found are quite similar.

Steepest Descent

For this method we use as fixed parameters $\alpha_0 = 5$, $btmax = 50$, $c_1 = 10^{-4}$, $\rho = 0.5$, a tolerance of $1.0e^{-5}$, $h = \sqrt{\epsilon_m}||\mathbf{x}||$, and FDgrad='c', which means we use the centered finite differences method to compute the gradient of the function. The results we have accomplished, for different starting points, number of iterations and dimension of the problem, are the following:

starting point	dimension	$kmax$	$f(x_0)$	$f(x_k)$	k	time
$\bar{x}$	10^4	10^3	5,5722e+16	4,9178e-03	k_{max}	27,240
$\bar{x}$	10^4	10^4	5,5722e+16	4,8431e-03	k_{max}	81,737
$\bar{x}$	10^5	10^2	5,5722e+22	4,9529e-02	k_{max}	361,646
$\bar{x}$	10^5	10^3	5,5722e+22	4,9501e-02	k_{max}	2.781,179
$\bar{x}$	10^5	10^4	5,5722e+22	4,9501e-02	k_{max}	22.146,334

We notice that for $n = 10^4$, the method converges the same minimum with the a faster convergence when kmax is smaller. For $n = 10^5$ on the other hand, the method still converges to another minimum with the only difference that, the smaller kmax, the lesser the time spent to reach the solution.

starting point	dimension	$kmax$	$f(x_0)$	$f(x_k)$	k	time
$\tilde{x}$	10^4	10^3	3,48265e+15	4,8431e-03	k_{max}	332,357
$\tilde{x}$	10^4	10^4	3,48265e+15	4,8431e-03	k_{max}	89,924
$\tilde{x}$	10^5	10^2	3,473264e+21	0,0503	k_{max}	401,2118
$\tilde{x}$	10^5	10^3	3,473264e+21	4,9501e-02	k_{max}	2.929,60

We observe that for $n = 10^4$, the method converges to the same minimum requiring always kmax iterations with the only difference that the greater kmax, the lesser the time spent to reach the solution. For $n = 10^5$, there is no convergenec of the method.

starting point	dimension	$kmax$	$f(x_0)$	$f(x_k)$	k	time
$\hat{x}$	10^4	10^3	5,2463e+16	4,9175e-03	k_{max}	25,782
$\hat{x}$	10^4	10^4	5,2463e+16	4,8431e-03	k_{max}	74,615
$\hat{x}$	10^5	10^2	5,5239e+22	4,9529e-02	k_{max}	288,925
$\hat{x}$	10^5	10^3	5,5239e+22	4,9501e-02	k_{max}	2.801,539

For $n = 10^4$, the method converges to the same minimum requiring kmax iterations with the only difference that the smallest kmax, the lesser the time spent to reach the solution. For $n = 10^5$, the method converges to another minimum under the same aforementioned conditions.

Appendix

```
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 :
    % 2 nd 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
```

```
tic
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)];

x0 = [1.2; 1.2];
% x0 = [-1.2; 1];
kmax = 100000;
tolgrad = 1e -6;
rho = 0.5;
c1 = 1e -04;
btmax = 50;
alpha0 = 5;

[xk , fk , gradfk_norm , k , xseq , btseq ] = ...
steepest_desc_bcktrck ( x0 , f , gradf , alpha0 , ...
kmax , tolgrad , c1 , rho , btmax );

% Creation of the meshgrid for the contour - plot
[X , Y] = meshgrid ( linspace (-6, 6, 500) , linspace (-6, 25, 500));
% Computation of the values of f for each point of the mesh

% Computation of the values of f for each point of the mesh
Z = 100*(Y-X .^2).^2+(1-X).^2;

% Plots

% Simple Plot
fig_n = figure ();
% Contour plot with curve levels for each point in xseq
[C , ¬] = contour (X, Y, Z);
hold on
% plot of the points in xseq
plot ([ x0(1) xseq(1, :)] , [x0(2) xseq(2, :)] , '--*')
hold off
title ('Steepest Descent - Rosenbrock')
```

```

% Much more interesting plot
fig_surf = figure ();
surf (X, Y, Z, 'EdgeColor' , 'none')
hold on
plot3 ([x0 (1) xseq(1, :)] , [x0(2) xseq(2 , :)] , [ f(x0), f(xseq)] , 'r...
    --*')
hold off
title ('Steepest Descent - Rosenbrock')
time = toc;
```

```
function [ xk , fk , gradfk_norm , k , xseq , btseq ] = ...
 steepest_desc_bcktrck_findiff ( x0 , f , gradf , alpha0 , ...
 kmax , tolgrad , c1 , rho , btmax , FDgrad , h )

switch FDgrad
    case 'fw'
        % OVERWRITE gradf WITH A F. HANDLE THAT USES findiff_grad
        % (with option 'fw')
        gradf = @ (x) findiff_grad (f, x, h, 'fw');

    case 'c'
        % OVERWRITE gradf WITH A F. HANDLE THAT USES findiff_grad
        % (with option 'c')
        gradf = @ (x) findiff_grad (f, x, h,   c       );
    otherwise
        % WE USE THE INPUT FUNCTION HANDLE gradf ...
        %
        % THEN WE DO NOT NEED TO WRITE ANYTHING !
        % ACTUALLY WE COULD DELETE THE OTHERWISE BLOCK
end

% 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 :
    % 2 nd 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
```

```
function y = penalty_gen(x)
%
% Penalty function 1
% Problem 27 of the file
%
n = length(x(:,1));
sum1 = 0;
sum2 = 0;

for k = 1:n
    sum1 = sum1+(1/sqrt(100000)*(x(k,:)-1)).^2;
    sum2 = sum2+x(k,:).^2;
end
y = 1/2*(sum1+(sum2-1/4).^2);

end
```

```
function y = brown_gen(x)
%
% Generalization of the Brown function
% Problem 12 of the file
%
k = length(x(:,1))/2;
sum1 = 0;
sum2 = 0;

for j = 1:k
    i = 2*j ;
    sum1 = sum1+((x((i-1),:)-3).^2)/1000-(x(i-1,:)-x(i,:))...
    +exp(20*(x(i-1,:)-x(i,:)));
    sum2 = sum2+(x(i-1,:)-3);
end
y = sum1+sum2.^2;

end
```

```
function y = band_trig_gen(x)
%
% Banded trigonometric problem
% Problem 16 of the file
```

```
%
n = length(x(:,1));
sum = 0;

for i = 2:n-1
    sum = sum+i*((1-cos(x(i,:)))+sin(x(i-1,:))-sin(x(i+1,:)));
end
sum = sum+1*(-sin(x(2,:)));
sum = sum+n*((1-cos(x(n,:)))+sin(x(n-1,:)));

y = sum;

end
```

Trying the aforementioned codes on test functions

```
% Brown function (12th problem)
x = zeros (n ,1);
for j = 2:2: n
    x(j)=-1; % = -2; % = +1;
end

f = @(x) brown_gen(x);

% Penalty function 1 (27th problem)
% x = zeros (n ,1);
% for i = 1:n
%     x(i) = i ; % = i/2; % = i-10;
% end
%
% f = @(x) penalty_gen(x);

% Banded trigonometric (16th problem)
% x = zeros(n,1);
% for i = 1:n
%     x(i) = 1; % = 1/2; % = 3;
% end
%
% f = @(x) band_trig_gen(x);

n = 10^3; % or different
kmax = 10^3; % or different

% NEWTON WITH BACKTRACKING LINE-SEARCH AND FINITE DIFFERENCES
% declaration of the variables for the implementation
f = @brown_func; % function handle of the function to minimize
alpha0 = 1; % initial steplength
kmax = 100; % maximum number of iterations of xk
tollgrad = 1.0000e-12; % tolerance for the norm of the gradient
c1 = 1e-4; % armijo conditions constant
rho = 0.8; % steplengths reduction factor
btmax = 10; % maximum number of iterations for the backtracking strategy
for n = [10^4, 10^5] % dimensions of the domain of the function to test
    x0 = zeros (n ,1);
    for j = 2:2:n
        x0(j) = -1; % = -2; % = +1;
    end
    % Implementation of the Newton method with line-search (in which the ...
    optimal
    % steplength is chosen by means of the backtracking strategy) using ...
    finite
    % differences to approximate the gradient and the Hessian matrix
    for i=[2, 4, 6, 8, 10, 12, 14]
        disp('**** NEWTON APPROXIMATED: START *****')
        tic
        [xk, fk, gradfk_norm, k, xseq, btseq] = newton_approximated(x0, f,...
     alpha0, ...
```

```
            kmax, tollgrad, c1, rho, btmax, i);
        disp('**** NEWTON APPROXIMATED: FINISHED *****')
        toc
        disp('**** NEWTON APPROXIMATED: RESULTS *****')
        disp('************************************')
        disp(['Dimension: ', num2str(n)])
        disp(['i: ', num2str(i)])
        disp(['xk: ', mat2str(xk)])
        disp(['N. of Iterations: ', num2str(k),'/',num2str(kmax), ';'])
        disp('************************************')
    end
end

% STEEPEST DESCENT WITH BACKTRACKING LINE-SEARCH AND FINITE DIFFERENCES
tolgrad = 1e-5;
rho = 0.5;
c1 = 1e-04;
btmax = 50;
alpha0 = 5;
gradf = '';
FDgrad = 'c';
h = sqrt(eps)*norm(x);

[xk, fk, gradfk_norm, k, xseq, btseq] = ...
steepest_desc_bcktrck_findiff(x, f, gradf, alpha0 , ...
kmax, tolgrad, c1, rho, btmax, FDgrad, h);
```

Test Rosenbrock using exact Newton with backtracking line-search

```
% declaration of the variables for the implementation
clear
close all
clc

f = @(x) (1 - x(1,:)).^2 + 100 * (x(2,:) - x(1,:).^2).^2;
gradf = @(x) [(-2*(1 - x(1)) - 200*(x(2) - x(1)^2)*2*x(1)); ...
    (200 * (x(2) - x(1)^2))];
Hessf = @(x) [(2 - 400*x(2) + 1200*x(1)^2), -400*x(1); ...
    -400*x(1), 200];

x0 = [-1.2; 1];
alpha0=1; % initial steplength
kmax=100; % maximum number of iterations of xk
tollgrad=1.0000e-12; % tolerance for the norm of the gradient
c1=1e-4; % armijo conditions constant
rho=0.8; % steplengths reduction factor
btmax=10; % maximum number of iterations for the backtracking strategy

disp('**** NEWTON EXACT: START *****')
tic
[xk, fk, gradfk_norm, k, xseq, btseq] = newton_bcktrck(x0, f, gradf, ...
Hessf, alpha0, kmax, tollgrad, c1, rho, btmax);
disp('**** NEWTON EXACT: FINISHED *****')
toc
disp('**** NEWTON EXACT: RESULTS *****')
disp('************************************')
disp(['xk: ', mat2str(xk)])
disp(['N. of Iterations: ', num2str(k),'/',num2str(kmax), ';'])
disp('************************************')
```

Test Rosenbrock using Newton with backtracking line-search and finite differences

```
% declaration of the variables for the implementation
f = @(x) (1 - x(1,:)).^2 + 100 * (x(2,:) - x(1,:).^2).^2; % function handle of the function to minimize
alpha0=1; % initial steplength
kmax=1000; % maximum number of iterations of xk
tollgrad=1.0000e-12; % tolerance for the norm of the gradient
c1=1e-4; % armijo conditions constant
rho=0.5; % steplengths reduction factor
btmax=10; % maximum number of iterations for the backtracking strategy
%x0 = [-1.2; 1];
x0 = [2; 5];
%x0 = [1.2; 1.2];
%for i=[2, 4, 6, 8, 10, 12, 14]
i=8;
disp('**** NEWTON APPROXIMATED: START *****')
tic
[xk, fk, gradfk_norm, k, xseq, btseq] = newton_approximated(x0, f, alpha0,...
    kmax, tollgrad, c1, rho, btmax, i);
disp('**** NEWTON APPROXIMATED: FINISHED *****')
toc
disp('**** NEWTON APPROXIMATED: RESULTS *****')
disp('************************************')
disp(['i: ', num2str(i)])
disp(['xk: ', mat2str(xk)])
disp(['N. of Iterations: ', num2str(k),'/',num2str(kmax), ';'])
disp('************************************')
```