WROCLAW UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

FIELD: Electronics
SPECIALITY: Advanced Applied Electronics

Optimization Methods: Unconstrained Optimization

AUTHOR:
Jaroslaw M. Szumega

SUPERVISOR:
Rafal Zdunek, D.Sc, K-4/W4

GRADE:

WROCŁAW 02.06.2017

Contents

Chapter 1

Solution to the given problems

Problem 1: Check the first- and second-order optimality conditions in the point: $x=\begin{bmatrix}1 & 1\end{bmatrix}^T$ of the Rosenbrock's function: $f(\mathbf{x})=100\left(x_2-x_1^2\right)^2+\left(1-x_1\right)^2$. Draw a contour plot of this function.

The first step will be expanding the given function, so further we can calculate the derivatives for gradient:

$$\begin{aligned} f(x) &= 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ &= 100(x_2^2 - 2x_2x_1^2 + x_1^4) + (1 - 2x_1 + x_1^2) \\ &= 100x_2^2 - 200x_2x_1^2 + 100x_1^4 + 1 - 2x_1 + x_1^2 \\ &= 100x_1^4 + x_1^2 - 2x_1 + 100x_2^2 - 200x_2x_1^2 + 1 \end{aligned}$$

And the point to check is:
$x = [x_1\ x_2]^T = [1\ 1]^T$

The First order optimality condition is:

$$\nabla f(x^*) = 0$$

$$\begin{aligned} \frac{\delta f(x)}{\delta x_2} &= 400x_1^3 + 2x_1 - 2 - 400x_2x_1 \\ &= 400 + 2 - 2 - 400 = 0 \end{aligned}$$

$$\begin{aligned} \frac{\delta f(x)}{\delta x_1} &= 200x_2 - 200x_1^2 \\ &= 200 - 200 = 0 \end{aligned}$$

In given point x = [1 1] the first–order optimality condition is fulfilled.

The Second order optimality condition is:

$$\begin{aligned}\nabla f(x^*) &= 0 \ (calculated\ in\ previous\ step)\\ \nabla^2 f(x^*) &= positive\ semi-definite\ matrix\end{aligned}$$

$$\begin{aligned}\nabla^2 f(x^*) &= \begin{bmatrix}\frac{\delta^2 f(x)}{\delta x_1^2} & \frac{\delta^2 f(x)}{\delta x_1 x_2}\\ \frac{\delta^2 f(x)}{\delta x_1 x_2} & \frac{\delta^2 f(x)}{\delta x_2^2}\end{bmatrix}\\ &= \begin{bmatrix}1200x_1^2 + 2 - 400x_2 & -400x_1\\ -400x_1 & 200\end{bmatrix}\\ for\ point\ [1\ 1] &= \begin{bmatrix}1200 + 2 - 400 & -400\\ -400 & 200\end{bmatrix}\\ H &= \begin{bmatrix}802 & -400\\ -400 & 200\end{bmatrix}\end{aligned}$$

As it can be noticed, all principal minors are positive ($H_{1,1}$ *and* $H_{2,2}$). Therefore the $H(x_*)$ is positive semi-definite.

Octave code below was written to plot contour for this task.

```
1 pkg load symbolic
2
3 syms x1 x2
4 f = @(x1,x2) 100.*(x2 - x1.^2).^2 + (1 - x1).^2;
5
6 ezcontour(f,[-3,3])
```

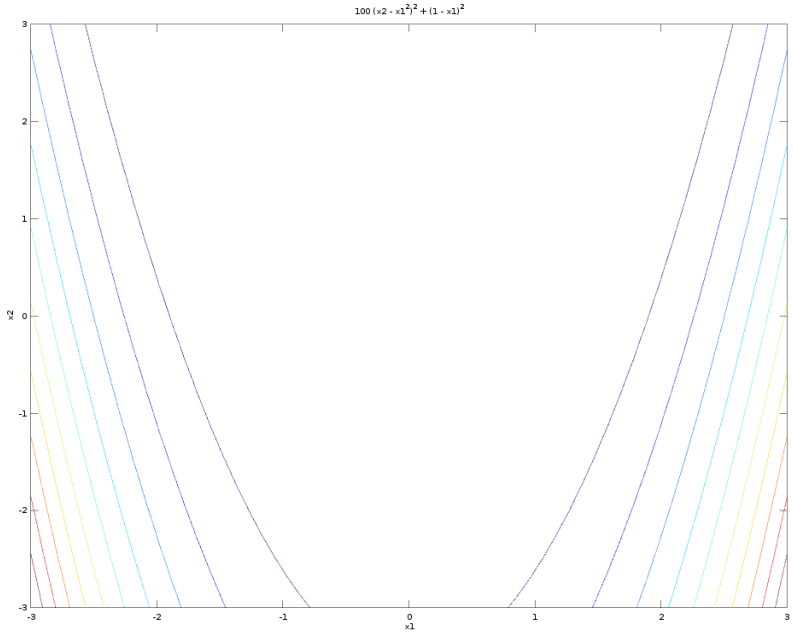


Figure 1.1 Contour plot of given function.

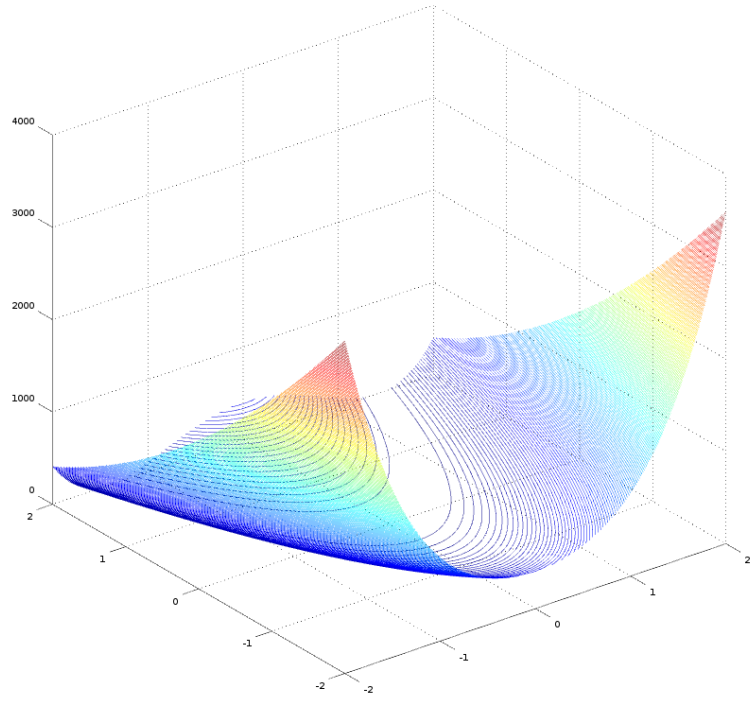


Figure 1.2 Contour plot of function in 3D version.

Problem 2: Check the first- and second-order optimality conditions for the quadratic functions:

a) $f(\mathbf{x}) = 2x_1^2 - x_1x_2 + \frac{1}{2}x_2^2 - 3x_1 + 3.5$,

b) $f(\mathbf{x}) = -\frac{3}{2}x_1^2 + x_1x_2 - \frac{1}{2}x_2^2 + 2x_1 - 1$,

c) $f(\mathbf{x}) = x_1^2 + 8x_1x_2 + \frac{1}{2}x_2^2 - 10x_1 - 9x_2 + \frac{9}{2}$.

Draw their contour plots. Are these functions convex? What kind of stationarity do they have?

Just like in the first task, we need to calculate the gradient and Hessian.// Then it can be determined which type of stationarity the functions have. **Point a)**

$$f(x) = 2x_1^2 - x_1x_2 - 3x_1 + \frac{x_2^2}{2} + 3.5$$
$$\nabla f(x^*) = 0$$

$$\frac{\delta f(x)}{\delta x_2} = 4x_1 - x_2 - 3$$
$$\frac{\delta f(x)}{\delta x_1} = -x_1 + x_2$$

To ensure the equality to zero, we can calculate that stationary point shall be:

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \end{aligned}$$

Now we will calculate the Hessian:

$$\nabla f(x^*) = 0 \ (\textit{calculated in previous step})$$

$$\begin{aligned} \nabla^2 f(x^*) &= \begin{bmatrix} \frac{\delta^2 f(x)}{\delta x_1^2} & \frac{\delta^2 f(x)}{\delta x_1 x_2} \\ \frac{\delta^2 f(x)}{\delta x_1 x_2} & \frac{\delta^2 f(x)}{\delta x_2^2} \end{bmatrix} \\ &= \begin{bmatrix} 4 & -1 \\ -1 & 1 \end{bmatrix} \\ H &= \begin{bmatrix} 4 & -1 \\ -1 & 1 \end{bmatrix} \end{aligned}$$

To establish the type of stationarity we need to check Hessian determinant and the values of principal minors:

$$\begin{aligned} det(H) = 4 + 1 &= 5 > 0 \\ H_{1,1} &= 1 > 0 \\ H_{2,2} &= 4 > 0 \end{aligned}$$

The Hessian is strictly positive definite, so we have the minimizer at calculated point.

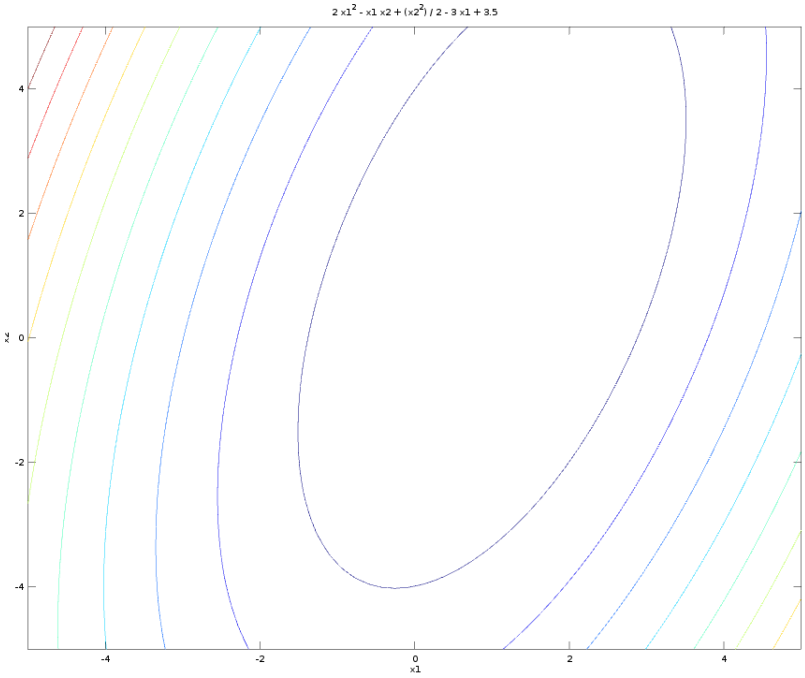


Figure 1.3 Contour plot of point a) function.

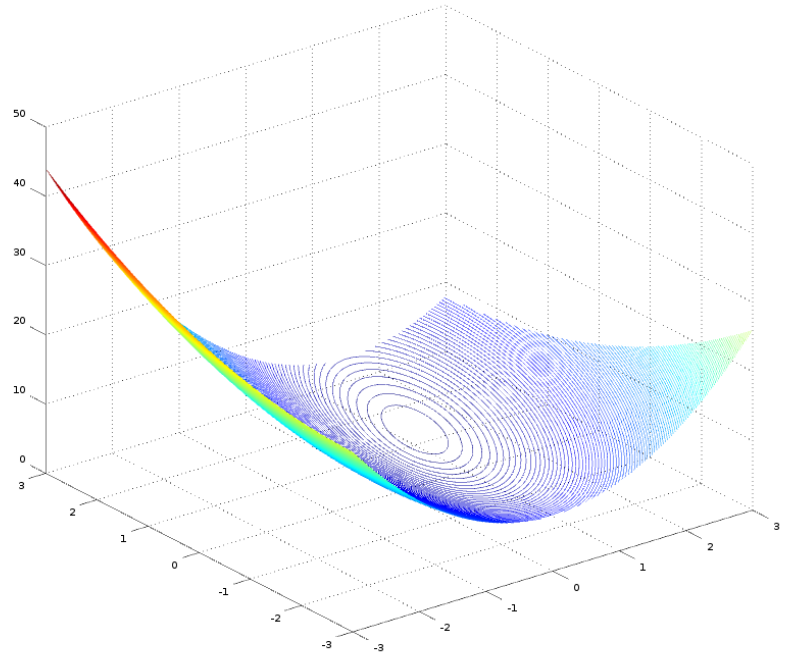


Figure 1.4 And its 3D version. We can notice the minimum.

Point b)

$$f(x) = -\frac{3x_1^2}{2} + x_1x_2 + 2x_1 - \frac{x_2^2}{2} - 1$$
$$\nabla f(x^*) = 0$$

$$\frac{\delta f(x)}{\delta x_2} = -3x_1 + x_2 + 2$$
$$\frac{\delta f(x)}{\delta x_1} = x_1 - x_2$$

The calculated values for fulfilling the condition of $\nabla f(x^*) = 0$:

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \end{aligned}$$

Now we will calculate the Hessian:

$$\nabla f(x^*) = 0 \ (\textit{calculated in previous step})$$

$$\begin{aligned} \nabla^2 f(x^*) &= \begin{bmatrix} \frac{\delta^2 f(x)}{\delta x_1^2} & \frac{\delta^2 f(x)}{\delta x_1 x_2} \\ \frac{\delta^2 f(x)}{\delta x_1 x_2} & \frac{\delta^2 f(x)}{\delta x_2^2} \end{bmatrix} \\ &= \begin{bmatrix} -3 & 1 \\ 1 & -1 \end{bmatrix} \\ H &= \begin{bmatrix} -3 & 1 \\ 1 & -1 \end{bmatrix} \end{aligned}$$

To establish the type of stationarity we need to check Hessian determinant and the values of principal minors:

$$\begin{aligned} det(H) = 3 - 1 = 2 &> 0 \\ H_{1,1} = -1 &< 0 \\ H_{2,2} = -3 &< 0 \end{aligned}$$

The Hessian is strictly negative definite, The maximizer is present.

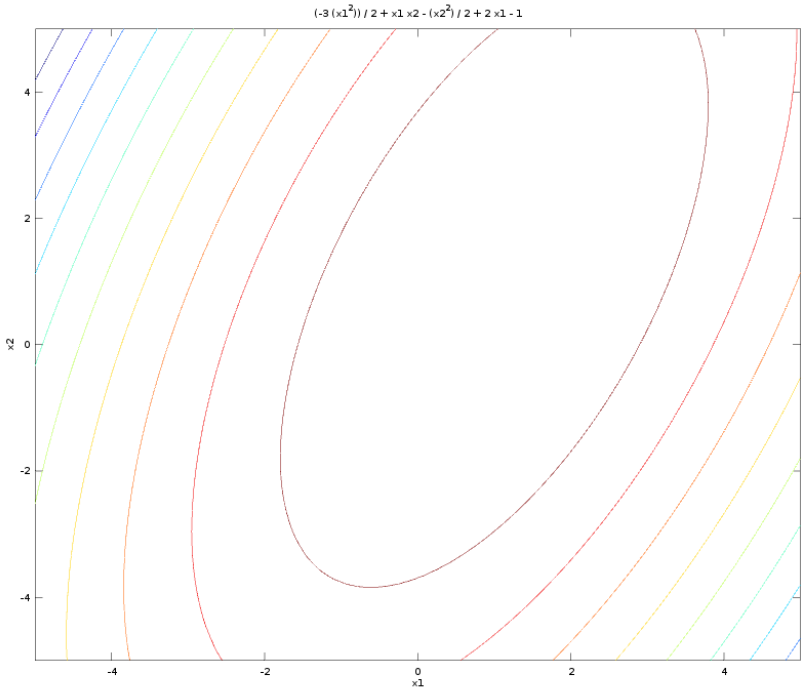


Figure 1.5 Contour plot of point b) function.

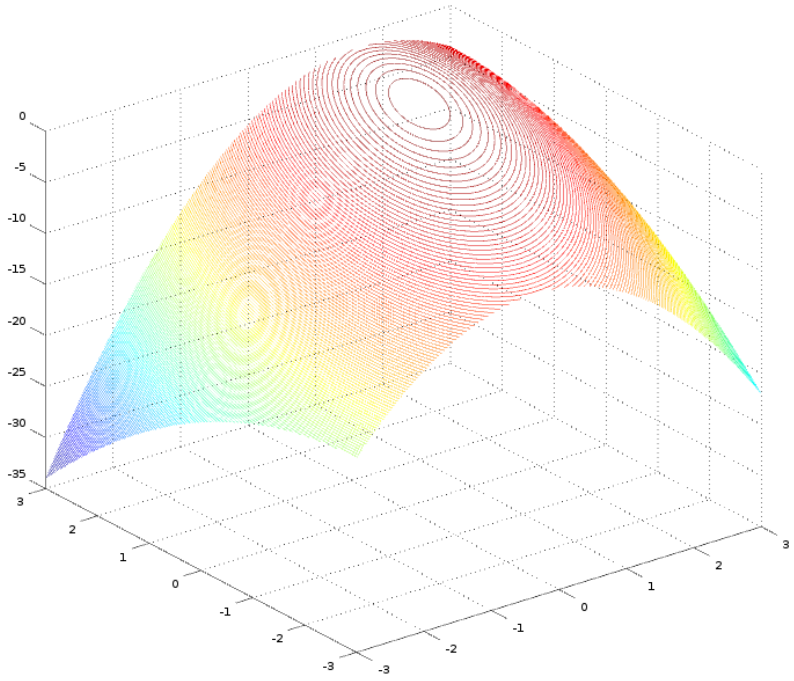


Figure 1.6 3D version with maximum observed.

Point c)

In third point, the following function needs to be analyzed:

$$f(x) = x_1^2 + 8x_1x_2 - 10x_1 + \frac{x_2^2}{2} - 9x_2 + 4$$

The task routine remains all the same as before:

$$\nabla f(x^*) = 0$$

$$\frac{\delta f(x)}{\delta x_2} = 2x_1 + 8x_2 - 10$$
$$\frac{\delta f(x)}{\delta x_1} = 8x_1 + x_2 - 9$$

To ensure the equality to zero, we can calculate that stationary point shall be:

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 1 \end{aligned}$$

Now we will calculate the Hessian:

$$\nabla f(x^*) = 0 \ (\textit{calculated in previous step})$$

$$\begin{aligned} \nabla^2 f(x^*) &= \begin{bmatrix} \frac{\delta^2 f(x)}{\delta x_1^2} & \frac{\delta^2 f(x)}{\delta x_1 x_2} \\ \frac{\delta^2 f(x)}{\delta x_1 x_2} & \frac{\delta^2 f(x)}{\delta x_2^2} \end{bmatrix} \\ &= \begin{bmatrix} 2 & 8 \\ 8 & 1 \end{bmatrix} \\ H &= \begin{bmatrix} 2 & 8 \\ 8 & 1 \end{bmatrix} \end{aligned}$$

To establish the type of stationarity we need to check Hessian determinant and the values of principal minors:

$$det(H) \ = \ 2 \ - \ 64 \ = \ -62 \ < \ 0$$

The determinant of Hessian is smaller than zero – at this point we know that the critical point of function is a saddle point.

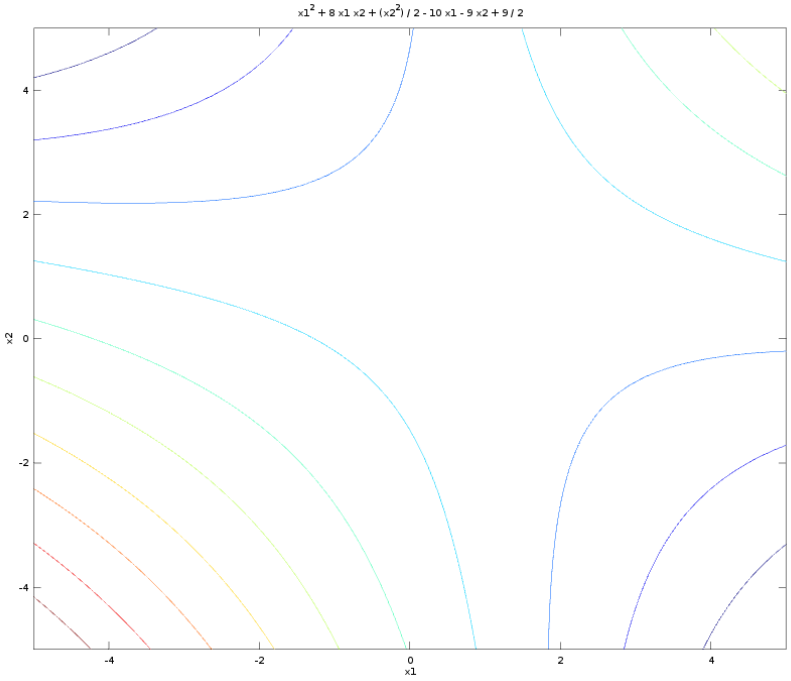


Figure 1.7 The contour plot of point c) function.

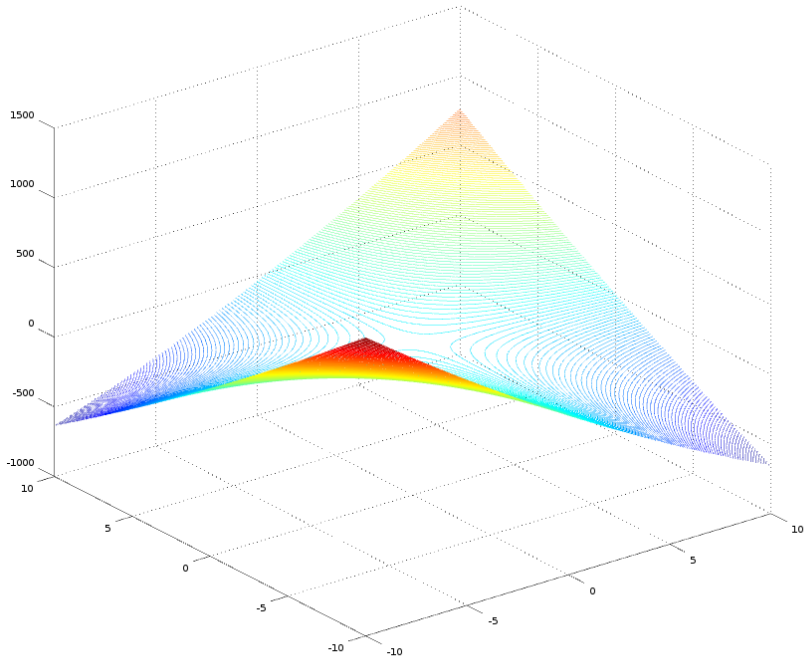


Figure 1.8 Visualization of point c) saddlepoint.

Problem 3: For the quadratic function $f : \Re^3 \to \Re$:

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{G}\mathbf{x}\text{, where } \mathbf{G} = \begin{bmatrix} \alpha & 1 & 1 \\ 3 & 2 & 1 \\ 2 & 2 & 3 \end{bmatrix},$$

determine the parameter α for which the function f is strictly convex.

To ensure the function is convex, the Hessian matrix of the function has to be strictly positive–definite.

$\nabla^2 f(x^*) = (x^T G x)'' = 2G$

$$H = \begin{bmatrix} 2\alpha & 2 & 2 \\ 6 & 4 & 2 \\ 4 & 4 & 6 \end{bmatrix}$$

Following the requirements, it needs to fulfill the following conditions:

- matrix determinant > 0,
- first principal minor > 0.

So now we establish the determinant:

$$\begin{aligned} det(H) &= 48\alpha + 16 + 48 - 32 - 16\alpha - 72 \\ &= 32\alpha - 40 \\ 32\alpha - 40 &> 0 \\ 4\alpha - 5 &> 0 \\ \alpha &> 5/4 \end{aligned}$$

And the first principal minor:

$$det(M1) = 8\alpha - 12$$

$$\begin{aligned} 8\alpha - 12 &> 0 \\ 2\alpha - 3 &> 0 \\ \alpha &> 3/2 \end{aligned}$$

In the end, if $alpha > 1.5$, then the given function will be convex.

Problem 4: The Himmelblau function $f(x,y)=\left(x^2+y-11\right)^2+\left(x+y^2-7\right)^2$ has four distinct minima: $f(3;2)=0$, $f(-3.78;-3.28)=0.0054$, $f(-2.81;3.13)=0.0085$, $f(3.58;-1.85)=0.0011$. Initialize the selected gradient descent and quasi-Newton algorithms on the sampled uniform rectangular grid: $x_i=-5+10\left(\frac{i-1}{29}\right)$, $y_i=-5+10\left(\frac{i-1}{29}\right)$, $i=1,\ldots,30$, and classify the initialization point according to the minima to which the selected algorithms converge.

For better insight of the problem, the contour plot of the Himmelblau function was shown/

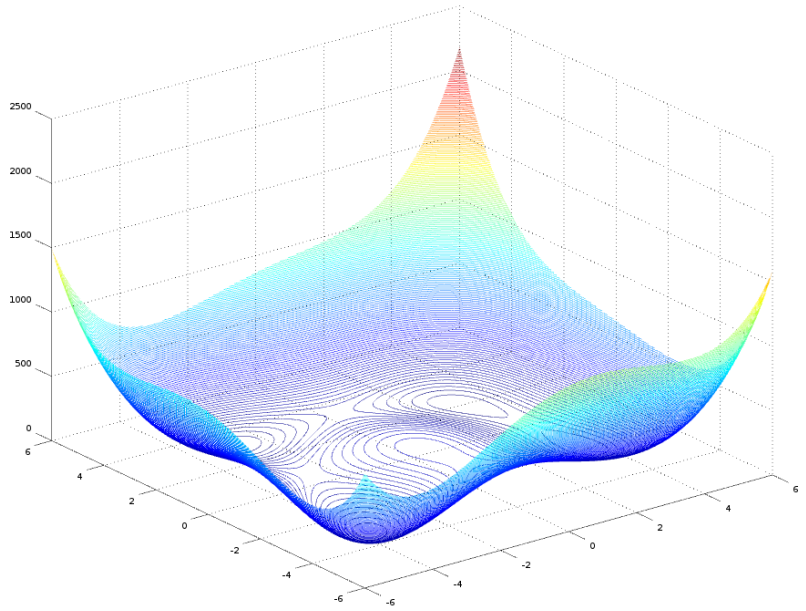


Figure 1.9 Contour plot of Himmelblau function.

On the figure 1.9 can observe the four regions that are supposed to be the listed minimas.

Considering the nature of the presented function, we can assume that depending on the starting point the algorithms will hit one of them. Of course there is also a possibility that method would not converge and will give a result that is not expected.

At the very beginning, there was an variant of the coded functions that symbolically calculated the necessary functions:

```
1  function [f0,g0] = himmelblau(xp)
2
3  syms x y
4  f = (x.^2 + y - 11).^2 + (x + y.^2 - 7).^2;
5  fs = function_handle(f);
6  f0 = fs(xp(1),xp(2));
7
8  g = gradient(f)';
9  gs = function_handle(g);
10 g0 = gs(xp(1),xp(2));
11
12 endfunction
```

Unfortunately, it turned out, that it is not the best approach. The calculations time was long – it took almost 70 seconds to calculate the minimum for one entry:

```
Entry = [5, 5]

solution =
-3.7793   -3.2832
Elapsed time is 66.5192 seconds.
```

Definitely, it is too long. So it was decided to hardcode the function and its gradient in a form that can be directly evaluated with argument in the future:

```
function [f,gf] = himmelblau_explicit(x)
    f = (x(1).^2 + x(2) - 11).^2 + (x(1) + x(2).^2 - 7).^2;

    gf = [2*(2*x(1)*(x(1).^2 + x(2) - 11) + x(1) + x(2).^2 - 7); 2*(x(1)
        .^2 + 2*x(2)*(x(1) + x(2).^2 - 7) + x(2) - 11)];
end
```

The explicit way of defining either function and gradient resulted in great speed–up (it is over 300x faster than symbolic operations).

```
Entry = [5, 5]

solution =
-3.2832   -3.2832
Elapsed time is 0.021183 seconds.
```

The table on the 1.10 presents the result of used algorithms.

Starting point	Gradient descent		BFGS algorithm	
[-5.000000 -5.000000]	[-2.805118 3.131313]	M3	[-3.779310 -3.283186]	M2
[-4.655172 -4.655172]	[3.000000 2.000000]	M1	[-3.779310 -3.283186]	M2
[-4.310345 -4.310345]	[-2.805118 3.131313]	M3	[-3.779310 -3.283186]	M2
[-3.965517 -3.965517]	[-3.779310 -3.283186]	M2	[-3.779310 -3.283186]	M2
[-3.620690 -3.620690]	[-3.779310 -3.283186]	M2	[-3.779310 -3.283186]	M2
[-3.275862 -3.275862]	[-3.779310 -3.283186]	M2	[-3.779310 -3.283186]	M2
[-2.931034 -2.931034]	[-3.779310 -3.283186]	M2	[-3.779310 -3.283186]	M2
[-2.586207 -2.586207]	[-3.779310 -3.283186]	M2	[-3.779310 -3.283186]	M2
[-2.241379 -2.241379]	[-3.779310 -3.283186]	M2	[-2.805118 3.131313]	M3
[-1.896552 -1.896552]	[-3.779310 -3.283186]	M2	[-0.270845 -0.923039]	fail
[-1.551724 -1.551724]	[-3.779310 -3.283186]	M2	[-0.270845 -0.923039]	fail
[-1.206897 -1.206897]	[-3.779310 -3.283186]	M2	[-0.270845 -0.923039]	fail
[-0.862069 -0.862069]	[-3.779310 -3.283186]	M2	[-0.270845 -0.923039]	fail
[-0.517241 -0.517241]	[-2.805118 3.131313]	M3	[-0.270845 -0.923039]	fail
[-0.172414 -0.172414]	[3.584428 -1.848127]	M4	[-0.270845 -0.923039]	fail
[0.172414 0.172414]	[3.000000 2.000000]	M1	[-0.270845 -0.923039]	fail
[0.517241 0.517241]	[3.000000 2.000000]	M1	[-0.270845 -0.923039]	fail
[0.862069 0.862069]	[3.000000 2.000000]	M1	[-2.805118 3.131312]	M3
[1.206897 1.206897]	[3.000000 2.000000]	M1	[3.000000 2.000000]	M1
[1.551724 1.551724]	[3.000000 2.000000]	M1	[3.000000 2.000000]	M1
[1.896552 1.896552]	[3.000000 2.000000]	M1	[3.000000 2.000000]	M1
[2.241379 2.241379]	[3.000000 2.000000]	M1	[3.000000 2.000000]	M1
[2.586207 2.586207]	[3.000000 2.000000]	M1	[3.000000 2.000000]	M1
[2.931034 2.931034]	[3.000000 2.000000]	M1	[3.000000 2.000000]	M1
[3.275862 3.275862]	[3.000000 2.000000]	M1	[3.000000 2.000000]	M1
[3.620690 3.620690]	[3.000000 2.000000]	M1	[3.000000 2.000000]	M1
[3.965517 3.965517]	[-3.779310 -3.283186]	M2	[3.000000 2.000000]	M1
[4.310345 4.310345]	[-3.779310 -3.283186]	M2	[3.000000 2.000000]	M1
[4.655172 4.655172]	[-3.779310 -3.283186]	M2	[3.385154 0.073851]	fail
[5.000000 5.000000]	[-3.779310 -3.283186]	M2	[3.584428 -1.848127]	M4

Figure 1.10 Starting point along with the corresponding hit minima for each algorithms.

Problem 5: Use gradient descent algorithms with the backtracking line search to find the minimizer of the Rosenbrock function. Set the initial step length $\alpha_0 = 1$ and print the step length used by each method at each iteration. First try the initial point $\mathbf{x}_0 = [1.2 \quad 1.2]^T$ and then a more difficult point $\mathbf{x}_0 = [-1.2 \quad 1]^T$.

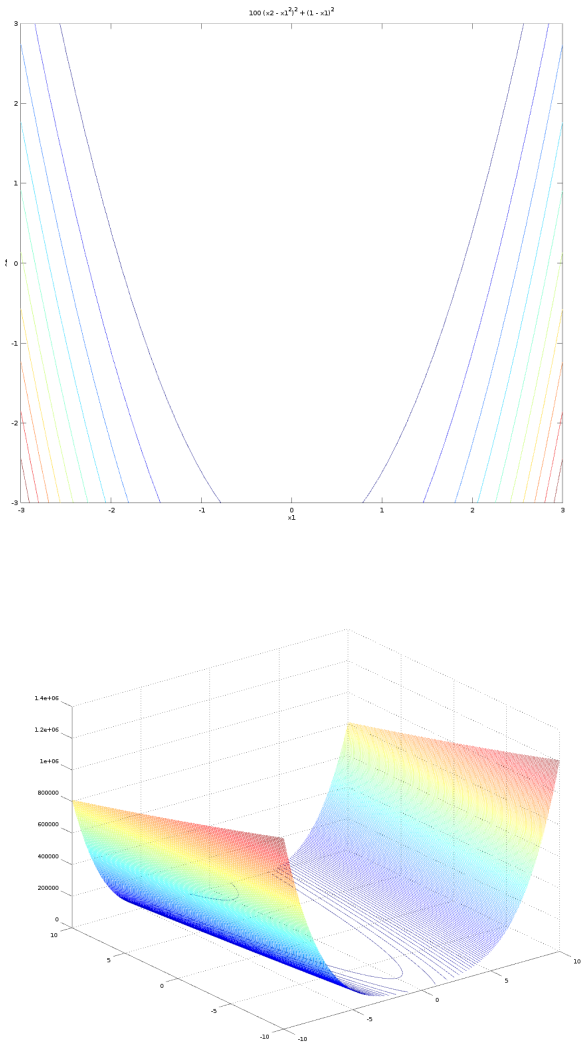


Figure 1.11 Two contour plots of the Rosenbrock functions (called also the valley/banana function.)

There was trial of calculus using both initial points. Gradient Descent in the first case found the minimum [1, 1]:

```
x1 = [1.2, 1.2]

iterations   =   13013

solution =
1.0000    1.0000

Elapsed time is 22.0776 seconds.
```

In the case of "more dificult" initial point, the GS algorithm was stuck. Running in debug mode has proven that it fail during backtracking search and goes into infinite loop.

Here is also requested print of step lengths (shortened a little because of amount of steps taken):

```
iteration =  100
a =    9.7656e-04
iteration =  200
a =    9.7656e-04
iteration =  300
a =    9.7656e-04
iteration =  400
a =    9.7656e-04
iteration =  500
a =    9.7656e-04
iteration =  600
a =  0.015625
iteration =  700
a =  0.0019531
iteration =  800
a =  0.0019531
iteration =  900
a =  0.0019531
iteration =  1000
a =  0.0019531

iteration =  2000
a =  0.0019531

iteration =  7000
a =  0.0019531

iteration =  9000


iteration =  10300
a =  0.0019531
iteration =  10400
a =  0.0039062

a =  0.0019531
iteration =  13000
a =  0.0019531
```

Problem 6: Let $f(\mathbf{x}) = 10\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2$. At $\mathbf{x} = [0 \quad -1]^T$ draw the contour lines of the quadratic model $m_k(\mathbf{p}) = f_k(\mathbf{x}) + \left(\nabla_x f_k(\mathbf{x})\right)^T \mathbf{p} + \frac{1}{2}\mathbf{p}^T \mathbf{B}_k \mathbf{p}$, assuming that $\mathbf{B}_k$ is the Hessian of $f(\mathbf{x})$. Draw the family of solutions of $\min_{\mathbf{p}} m_k(\mathbf{p})$, *s.t.* $\|\mathbf{p}\| \le \Delta_k$ as the trust region radius varies from $\Delta = 0$ to $\Delta = 2$. Repeat this at $\mathbf{x} = [0 \quad 0.5]^T$.

Firstly, the gradient and Hessian will be computed:

$$\nabla f = \begin{bmatrix} -40x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 20(x_2 - x_1^2) \end{bmatrix}$$

$$\nabla^2 f = \begin{bmatrix} -40x_2 + 120x_1^2 + 2 & -40x_1 \\ -40x_1 & 20 \end{bmatrix}$$

We assume that p* is a solution of given model, only if there exist $\lambda > 0$ and:
$(B + \lambda I)p^* = -\nabla f$
$\lambda(\Delta - ||p^*|| = 0)$

At x = [0, -1]:

$$f_k = 11, \ \nabla f_k = [-2, -20]^T, \ B_k = \begin{bmatrix} 42 & 0 \\ 0 & 20 \end{bmatrix}.$$

$$(B_k + \lambda I)p* = -\nabla f_k, \text{ thus } p* = \left[\frac{2}{42 + \lambda}, \frac{20}{20 + \lambda}\right]^T.$$

If $\lambda = 0$, $\|p^*\| = \sqrt{442}/21$. If $\lambda > 0$, $\|p^*\| < \sqrt{442}/21$.

$$p^* = \begin{cases} [1/21, 1]^T, & \sqrt{442}/21 \le \Delta \le 2. \\ \left[\frac{2}{42+\lambda}, \frac{20}{20+\lambda}\right]^T, \text{ with } \|p^*\| = \Delta, & 0 < \Delta < \sqrt{442}/21. \end{cases}$$

The same calculations can be performed for the second point.

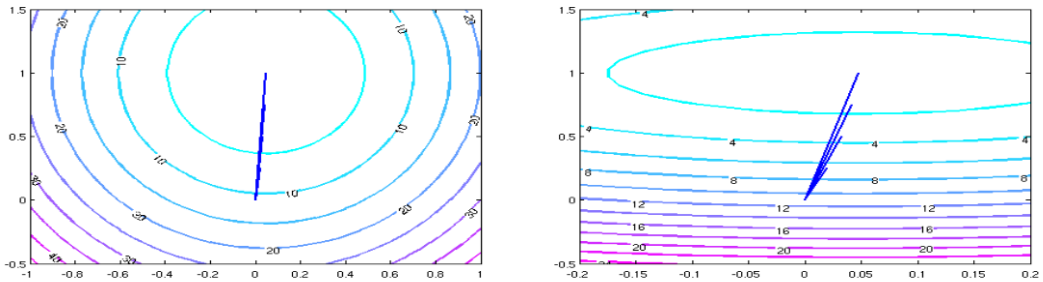


Figure 1.12 Contour and zoomed solutions for the point x = (0,-1)

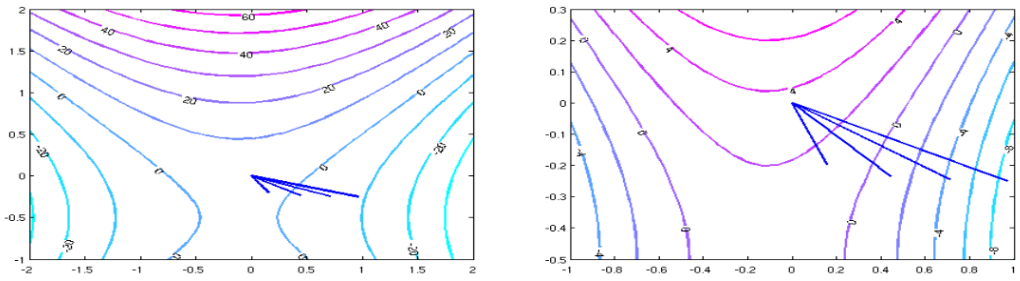


Figure 1.13 Plots for the second x point.

Problem 7: Given the Rosenbrock's function: $f(\mathbf{x}) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$, draw its contour lines, and then find its local minimizer with the Fletcher-Reeves and Polak-Ribiere methods, starting from the initial point: $x_0 = \begin{bmatrix} -1.2 & 1 \end{bmatrix}^T$.

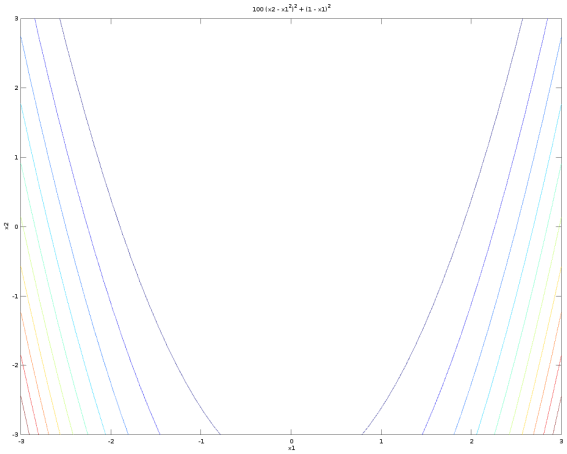


Figure 1.14 Contour plot of Rosenbrock's function.

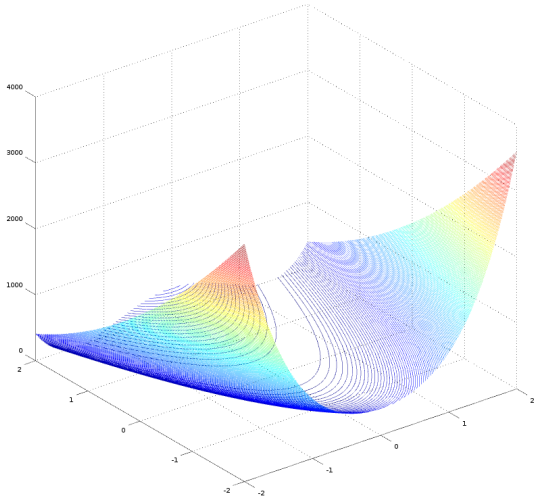


Figure 1.15 Contour plot of Rosenbrock's function in 3D version.

The both methods were tested along with given results, but also with limited iterations (again the fall into endless loop was experienced):

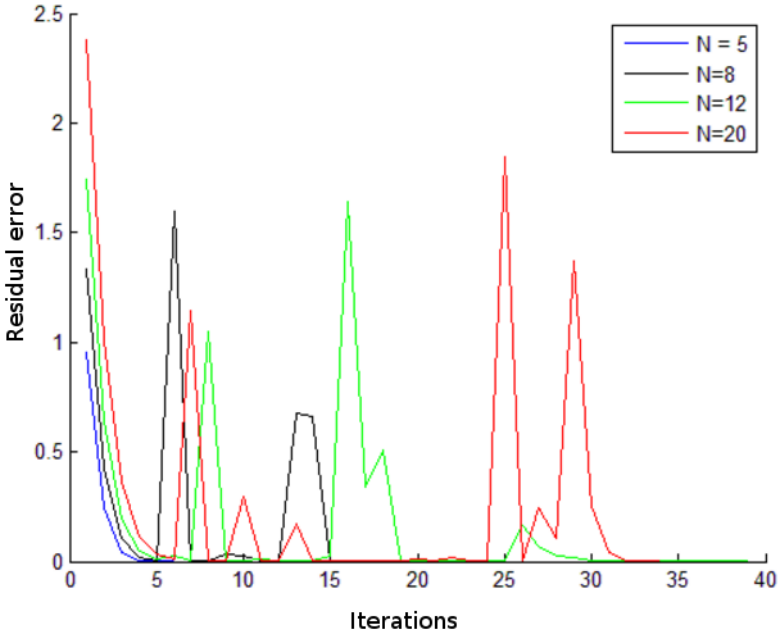
```
Fletcher-Reeves algorithm

solution =
1.0560   1.1234
Elapsed time is 2.1183 seconds.


Polak-Ribiere algorithm
solution =
1.0075   1.1003
Elapsed time is 1.821183 seconds.
```

Problem 8: Using algorithms such as: the basic gradient descent, quasi-Newton, and CG, solve the system of linear equations: $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} = [a_{ij}] \in \Re^{N \times N}$, with $a_{ij} = \frac{1}{i + j - 1}$, is the Hilbert matrix, and the right-hand side is given by $\mathbf{b} = [1, \ldots, 1]^T \in \Re^N$. Start the iterations from $\mathbf{x}_0 = \mathbf{0}$. If it is not specified by the algorithm, use the square Euclidean distance to measure the residual error. Try dimensions $N = 5, 8, 12, 20$ and report the number of iterations required to reduce the residual error below 10^{-6}. Then apply the preconditioning with the incomplete Cholesky factorization, and carry out the similar tests. Compare the results.

The matrix A is recognized as a Hilbert matrix, therefore the corresponding Octave function was used to initialize it.
The error was set to $1e-6$ and then the CG algorithm was used to perform needed calculations and the result are plotted on the image below.



Chapter 2

Listings of algorithms

2.1 Coded selected algorithms

Algorithm - Gradient Descent algorithm

```
function [x0, k] = gdescent_backtrack (fun,x0,err,a)

rho = 0.5;
c = 0.0001;

df = 1;
xk = x0;
k = 1;
while ((norm(df) > err))
    [f,df] = fun(x0);
    a = backtracking(fun,x0,-df, rho, c);
    if(mod(k,100)==0)
    iteration = k
    a
    endif
    x0 = x0 - a*df;
    k = k+1;
end

endfunction
```

Algorithm - Conjugate Gradient algorithm

```
function [x, n, res,rx] = conjugate_gradient(A, f, x0, it, err)

x = x0;
r = f - A*x0;
p = r;
r1 = r'*r;
n = 0;

normf = norm(f);
```

```
10
11 while (norm(r)/normf > err)   % tolerance condition
12     a = A*p;
13     alpha = r1/(a'*p);
14
15     x = x + alpha*p;
16     r = r - alpha*a;
17
18     r2 = r'*r;
19     beta = r2/r1;
20     p = r + beta * p;
21
22     r1 = r2;
23     n = n + 1;
24
25     res(n) = norm(f - A*x);
26     rx(n) = norm(x - x0);
27     if (n == it)
28         break
29     endif
30 endwhile
31 endfunction
```

Algorithm - Polak-Ribiere algorithm

```
1  function [x,f] = ribiere(fun, x, iter)
2  [f,df] = fun(x);
3
4  p = -df;
5  alpham = 0.01;
6  k = 0;
7
8  while ( k<iter && f!=0 )
9      a = strongwolfe(fun,p,x,alpham);
10
11     [f,df] = fun(x);
12     x = x + a*p;
13     [f1,df1] = fun(x);
14
15     B = dot(df1',(df1-df)) / norm(df)^2;
16     p = -df1 + B*p;
17     df=df1;
18     f=f1;
19
20     k = k+1;
21 endwhile
22 endfunction
```

Algorithm - Fletcher-Reeves algorithm

```
function [x,f] = fletcherReeves(fun, x, iter)

[f,df] = fun(x);
p = -df;

alpham = 0.01;
k = 0;

while ( k<iter && f!=0 )
    a = strongwolfe(fun,p,x,alpham);

    [f,df] = fun(x);
    x = x + a*p;
    [f1,df1] = fun(x);

    B = dot(df1',df1) / dot(df',df);
    p = -df1 + B*p;
    df=df1;
    f=f1;

    k = k+1;

endwhile
endfunction
```

Strategy - Strongwolfe algorithm [4]

```
function alphas = strongwolfe(f,d,x0,alpham)

alpha0 = 0;
alphap = alpha0;
alphas = 0;

c1 = 1e-4;
c2 = 0.5;

alphax = alpham;

[fx0,gx0] = feval(f,x0)
fxp = fx0;
gxp = gx0;
i=1;


while (i<1000)
    xx = x0 + alphax*d;
    [fxx,gxx] = feval(f,xx);

    if (fxx > dot(fx0 + c1*alphax*gx0,fx0 + c1*alphax*gx0) || ((i > 1)
       && (fxx >= fxp)))
        alphas = zoom(f,x0,d,alphap,alphax);
        return;
    end

    if ( dot(abs(gxx),abs(gxx)) <= dot(-c2*gx0,-c2*gx0) )
        alphas = alphax;
        return;
    end

    if ( dot(gxx,gxx) >= 0 )
        alphas = zoom(f,x0,d,alphax,alphap);
        return;
    end

    alphap = alphax;
    fxp = fxx;
    gxp = gxx;

    alphax = alphax + (alpham-alphax);
    i = i+1;
    end

    if (alphas==0)
    alphas = alphax;
endwhile
endfunction
```

Strategy - Backtracking line search [4]

```
function a = backtracking(fun,x,d, rho, c)
a = 1;

[fk, gk] = fun(x);
xx = x;
x = x + a*d;
fk1 = fun(x);

while (fk1 > (fk + c*a*(gk'*d))),

a = a*rho;
x = xx + a*d;
fk1 = fun(x);

end
```

Bibliography

[1] J. Nocedal, S. J. Wright, Numerical Optimization, Springer, 1999,

[2] Zdunek R., Optimization Methods - lecture slides.

[3] Mathworks webpage, "Unconstrained Optimization Algorithms", https://www.mathworks.com/help/optim/ug/unconstrained-nonlinear-optimization-algorithms.html

[4] Nonlinear Programming course webpage, North Carolina State University, http://www4.ncsu.edu/ kksivara/ma706/

[5] University of Chicago, CG convergence lecture (for Fletcher-Reeves algorithm), http://www.mcs.anl.gov/ anitescu/CLASSES/2011/LECTURES/STAT310-2011-lec6.pdf