

# First semester project in TMA4215

10026 and 10068

03.10.2011

## 1 Task

We consider minimization problems of the type

$$\min_{\mathbf{x} \in \mathbb{R}^n} g(\mathbf{x}), \quad g(\mathbf{x}) := -\mathbf{b}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T H \mathbf{x} + \frac{1}{12} \mathbf{x}^T C(\mathbf{x}) \mathbf{x},$$

here  $\mathbf{b} \in \mathbb{R}^n$  and,  $H$  is a  $n \times n$  symmetric and positive definite matrix and  $C(\mathbf{x})$  is a diagonal matrix with diagonal entries  $c_i x_i^2, i = 1, \dots, n$ . Here  $c_i > 0$  are the components of a vector  $\mathbf{c} \in \mathbb{R}^n$  and  $x_i$  are the components of  $\mathbf{x}$ .

## 2 Mathematical calculations

Before the problem is implemented, some properties of the problem are discussed.

### 2.1 Positive definition of $H$

Since we use only a specific  $H$ , the proof is not generally, but for  $n = 2$ . Let  $\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \in \mathbb{R}^2 \setminus \vec{0}$ , and  $H = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$ , with  $a, c > 0$ .

$$\mathbf{u}^T H \mathbf{u} = au_1^2 + cu_2^2 + 2bu_1u_2$$

If is chosen to  $b = \sqrt{a}\sqrt{c}$ , this yields to

$$= (\sqrt{a}u_1 + \sqrt{c}u_2)^2 > 0,$$

for all  $\mathbf{u}$ .

To obtain a positive definite  $H$ ,  $b$  has to be  $\sqrt{a}\sqrt{c}$ .

### 2.2 Gradient

The gradient can easily be calculated with sums.

$$\nabla g = \nabla \left( -\sum_{i=1}^n b_i x_i + \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n H_{ij} x_i x_j + \frac{1}{12} \sum_{i=1}^n c_i x_i^4 \right)$$

The two sums in the middle are divided into the diagonal element and the not diagonal elements. All not diagonal elements are there twice, because  $H$  is symmetric and therefore  $H_{ij} = H_{ji}$ .

$$\begin{aligned} &= \nabla \left( -\sum_{i=1}^n b_i x_i + \frac{1}{2} \sum_{i=1}^n H_{ii} x_i^2 + \sum_{i=1}^n \sum_{j=1}^{i-1} H_{ij} x_i x_j + \frac{1}{12} \sum_{i=1}^n c_i x_i^4 \right) \\ &= -\mathbf{b} + H\mathbf{x} + \frac{1}{3} C\mathbf{x} \end{aligned}$$

## 2.3 Hessian

The Hessian of  $g$  is easily calculable:

$$\nabla^2 g = H + C$$

## 2.4 Existence of minimum

Let  $u \in \mathbb{R}^n \setminus \vec{0}$  be an arbitrary vector, then

$$u^T (\nabla^2 g(x)) u = u^T (H + C) u = u^T H u + u^T C u = u^T H u + \sum_{i=1}^n c_i u_i^2 x_i^2.$$

Since  $H$  is positive definite,  $u^T H u > 0$  and because  $c_i > 0 \Rightarrow u^T C u > 0$ , so

$$u^T (\nabla^2 g(x)) u > 0$$

That means that the Hessian of  $g$  is positive definite and therefore strictly convex and has at most one local minimum.

## 2.5 Equivalence of steepest decent method and forward Euler method

The steepest decent method is defined as

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha \nabla g(\mathbf{x}^k).$$

The Euler's method uses the equation

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha f(\mathbf{x}^k)$$

to solve the equation  $\mathbf{x}'(t) = f(\mathbf{x}(t))$ . If  $f(\mathbf{x}^k)$  is chosen as  $-\nabla g(\mathbf{x}^k)$ , both methods return the same result.

The equation which should be solved with Euler's method is therefore

$$x'(t) = -\nabla g(x(t)).$$

## 2.6 Optimal $\alpha$ in the steepest decent method

To find the optimal  $\alpha$  for the steepest decent method,

$$g(\mathbf{x}^{(k+1)}) = g(\mathbf{x}^{(k)} - \alpha^{(k)} \nabla g(\mathbf{x}^{(k+1)}))$$

has to be minimal, so  $\frac{\partial}{\partial \alpha} g(\mathbf{x}^{(k+1)})$  has to be zero. This leads to the equation

$$\begin{aligned} & \left( \mathbf{b} \nabla g - \mathbf{x}^T \left( H + \frac{C}{3} \right) \nabla^T g \right) + ((\nabla g)(H + C) \nabla g) \alpha \\ & + \left( - \sum_{i=1}^n c_i x_i (\nabla g)_i^3 \right) \alpha^2 + \frac{1}{3} \left( \sum_{i=1}^n c_i (\nabla g)_i^4 \right) \alpha^3 = 0 \end{aligned}$$

Since the function is a cubic polynomial, the limit for  $\lim_{\alpha \rightarrow \pm\infty} f = \pm\infty$  and because the function is continuous, there has to be at least one real zero.

To the slope of  $\frac{\partial g}{\partial \alpha}$  is

$$\frac{\partial^2}{\partial \alpha^2} g(\mathbf{x} - \alpha \nabla g(\mathbf{x})) = \frac{\partial}{\partial \alpha} (-(\nabla g(\mathbf{x})) \nabla g(\mathbf{x} - \alpha \nabla g(\mathbf{x}))) = (\nabla g(\mathbf{x}))^T \nabla^2 g(\mathbf{x} - \alpha \nabla g(\mathbf{x})) \nabla g(\mathbf{x}) > 0,$$

because the Hessian is positive definite. With a positive slope for every  $\alpha$ , the function rising. So only one zero is possible.

### 3 Main algorithms

#### 3.1 Generation of the data

The data is generated in the function *data*.

```
1 function [ b, H, c ] = data
2     b = [ 1; 0 ];
3     c = [ 200; 400 ];
4     H = [ 200, 20 ; 20, 2 ];
5 end
```

With the calculations in chapter 2.1 can be easily seen that  $H$  is positive definite.

#### 3.2 Function, gradient and Hessian of $g$

```
1 function [ g ] = problem ( X )
2     [ b, H, c ] = data;
3     dim = size(H,1);
4     C = zeros ( dim, dim );
5     for i = 1 : dim
6         C(i,i) = c(i) * X(i) * X(i);
7     end
8     g = - b' * X + 0.5 * X' * H * X + 1/12 * X' * C * X;
9 end
```

```
1 function [ nablaG ] = grad( X )
2     [ b, H, c ] = data;
3     dim = size ( H, 1 );
4     for i = 1 : dim
5         C(i,i) = c(i) * X(i) * X(i);
6     end
7     nablaG = - b + H * X + 1/3 * C * X;
8 end
```

```
1 function [ hessG ] = hessian( X )
2     [ ~, H, c ] = data;
3     dim = size ( H, 1 );
4     C = zeros ( dim, dim );
5     for i = 1 : dim
6         C(i,i) = c(i) * X(i) * X(i);
7     end
8     hessG = H + C;
9 end
```

#### 3.3 Minimum searching algorithm

All methods are similar, only line 9 differs. One of the lines 9, 10 and 11 can be uncommented, the rest must be commented. The real implementation is more complicated and is shown in chapter 4.

```
1 X = [ 4; -1 ];
2 maxiterations = 30;
3 tol = 1e-8;
4
5 norm_old = norm ( grad ( X ) );
6 condition = 1;
7 while condition
8     maxiterations = maxiterations - 1;
9     X = X - alpha * grad ( X ); % steepest decent method with constant alpha
10    % X = X - optimalAlpha ( X ) * grad ( X ); % with optimal alpha
11    % X = X - linsolve ( hessian ( X ), grad ( X ) ); % Newton method
12    residual = norm ( grad ( X ) ) / norm_old;
13    condition = (maxiterations > 0) && ( residual > tol);
14 end
```

### 3.4 Computation of $\alpha$

The calculation for  $\alpha$  can be seen in chapter 2.6. Here only the implementation is stated.

```
1 function [ alpha_optimal ] = optimalAlpha( X )
2 [ b, H, c ] = data;
3 dim = size ( H, 2 );
4 C = zeros ( dim, dim );
5 gradg = grad ( X );
6 a3 = 0;
7 a2 = 0;
8 for i = 1 : dim
9     C(i,i) = c(i) * X(i)^2;
10     a3 = a3 + c(i) / 3 * gradg(i)^4;
11     a2 = a2 - c(i) * X(i) * gradg(i)^3;
12 end
13 a1 = gradg' * ( H + C ) * gradg;
14 a0 = b'*gradg - X' * H * gradg - 1/3 * X' * C * gradg;
15 alphas = roots( [ a3, a2, a1, a0 ] );
16 for i = 1 : 3
17     if imag ( alphas ( i ) ) == 0
18         alpha_optimal = alphas ( i );
19     end
20 end
21 end
```

## 4 Structure of the project

The project can be started by executing *main()*. In this function, the initial guess, the tolerance and the maximal number of iterations is defined. *main* first calls the function *plotmethod*, which contains a variant of the code in chapter 3.3 and returns the relative residuals, and the points  $\mathbf{x}^k$ . The change of  $\mathbf{x}$  in each iteration is calculated in *delta* for various methods. At the end of function *main*, all plots are generated using the functions *drawfunction* and *drawcurve*.

To count the time a process needs, there are the functions *meantime*, which makes many experiments and *fastmethod*, which is a faster variant of *plotmethod*.

## 5 Results

With the program presented above, several experiments can be done. If not other stated,  $(4, -1)^T$  is used as initial guess,  $10^{-8}$  as tolerance and the number of iterations is limited to 30. A first look at the function in figure 5 with the defined values suggest the minimum is around  $(0, 0)^T$ .

### 5.1 Steepest decent method with constant $\alpha$

If the initial guess is far away from the solution, the gradient at this point is very large. To get  $\mathbf{x}^2$ , this gradient has to be multiplied by  $\alpha$ . If  $\alpha$  is not small enough, the value of  $g$  for the next iteration gets infinity and the program breaks down. With  $\alpha = 0.0001$ , the algorithm works, but the minimum is only slowly approached. The tolerance is reached after 744ms and 25 473 iterations, so in most cases the limiting factor is the maximal number of iterations.

In figure 5 the steepest decent method with constant  $\alpha$  approaches the minimum, but does not reach it for the maximum number of iterations is limited to 30.

### 5.2 Steepest decent method with optimal $\alpha$

The algorithm works better than with constant  $\alpha$ , but when the relative residual drops below  $10^{-3}$ , the relative residual rises and falls alternately, but with a dropping drift. It reaches the tolerance for the relative residual after 26 iterations and needs 4.33ms, which is much faster than with a constant  $\alpha$ .

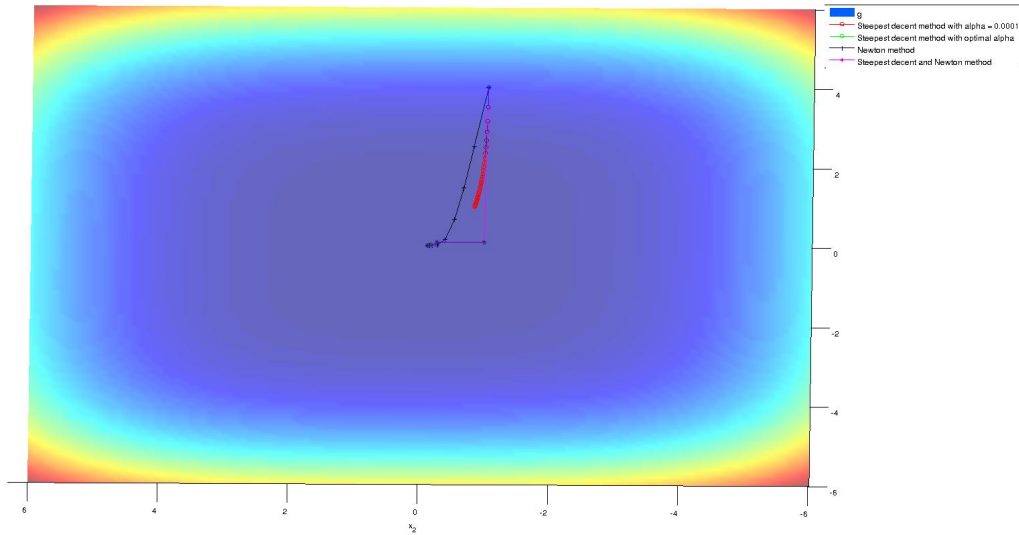


Figure 1: The function  $g$  as a three dimensional plot. The points of for various iterations are drawn, too.

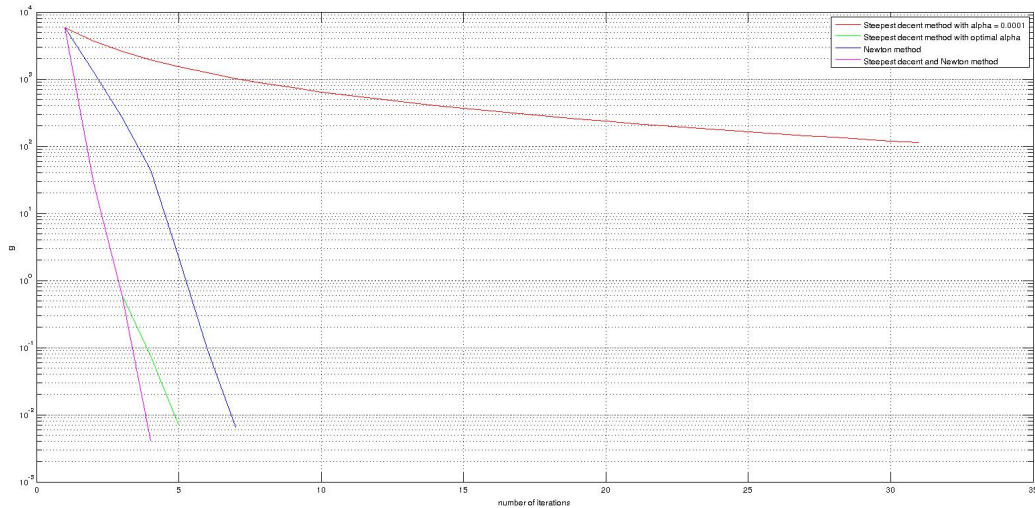


Figure 2: The value of  $g$  as a function of iterations for various methods.

### 5.3 Newton method

At the beginning the Newton method reaches the minimum more slowly than the steepest decent method with optimal  $\alpha$ . This can be seen well in figure 5, where the Newton method has a smaller step size at the beginning than the steepest decent method with optimal  $\alpha$ . After the relative residual drop below  $10^{-3}$ , the Newton method converges faster, which can be seen in figure 5. It reaches the tolerance after 1.73 ms and 11 iterations and is therefore faster than the steepest decent method with optimal  $\alpha$ .

If the initial guess for the minimum is far away from the minimum, the Newton method converges more slowly than the steepest decent method with optimal  $\alpha$ . So the Newton method is not always faster than the steepest decent method with optimal  $\alpha$ , only if the initial guess is near the minimum.

### 5.4 Steepest decent method and Newton method

First the steepest decent method is applied and after a certain tolerance limit, the algorithm switches to the Newton method.

To find this tolerance limit, the total time for the algorithm is watched with various tolerance limits.

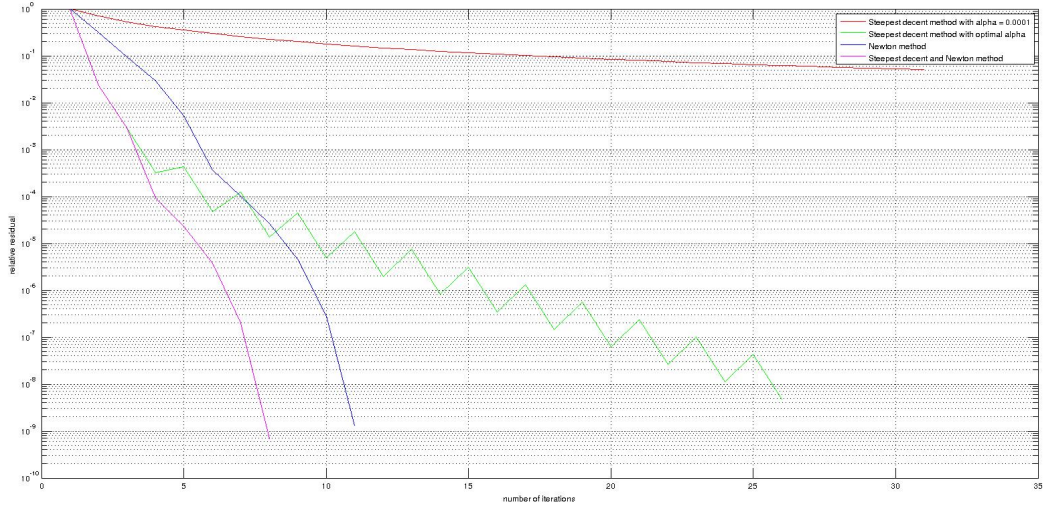


Figure 3: The relative residuals as a function of iterations for various methods.

The program is run 1000 times for each limit and the mean is given in table 1. To run the function with minimal time, the steepest decent method is applied till the relative residual is below  $10^{-2}$  and the Newton method is applied till the default tolerance ( $10^{-8}$ ) is reached. In figure 5 it can be seen that the steepest

tolerance limit	$10^{-1}$	$10^{-2}$	$10^{-3}$	$10^{-4}$	$10^{-5}$	$10^{-6}$
time [ms]	1.73	1.72	1.79	1.92	4.24	4.27

Table 1: Times for the mixture of steepest decent and Newton method for various changing conditions.

decent method and this algorithm start together. In both other plots it can be seen that at a certain point the methods differ.

The algorithm is not much faster than the Newton method alone, which is because the initial guess is in the near of the minimum. A larger difference between the times can be obtained with a point far from the minimum.

We decided to keep the number of iterations constant, because 20 iterations is not much.

## 6 Summary

It is shown that the function has at least one minimum and different numerical methods for finding the minimum are implemented and tested. The results, their convergence and the time they need are compared. It can be seen that the best results can be obtained with a combination with the steepest decent method to get in the near of the minimum and the Newton method to get faster to the minimum, once there is a gross approximation.