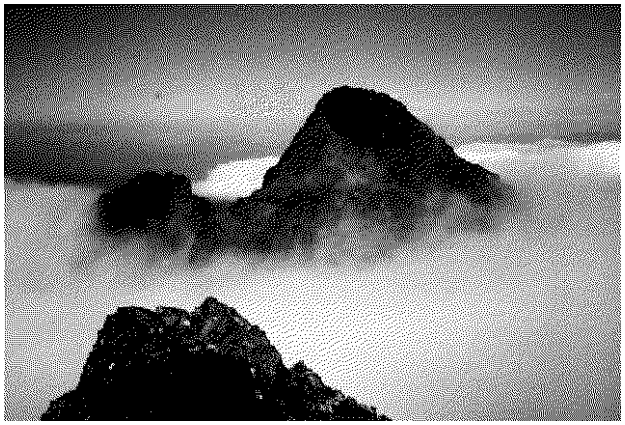
IMM
DEPARTMENT OF MATHEMATICAL MODELLING
Technical University of Denmark
DK-2800 Lyngby – Denmark

J. No. **H69**
23.8.1999

# UNCONSTRAINED OPTIMIZATION

**Poul Erik Frandsen, Kristian Jonasson**
**Hans Bruun Nielsen, Ole Tingleff**



**LECTURE NOTE**
**IMM-LEC-2**

**IMM**

## Abstract

This lecture note is intended for use in the course *04212 Optimization and Data Fitting* at the Technincal University of Denmark. It covers about 25% of the curriculum. Hopefully, the note may be useful also to interested persons not participating in that course.

The aim of the note is to give an introduction to algorithms for unconstrained optimization. We present Conjugate Gradient, Damped Newton and Quasi Newton methods together with the relevant theoretical background.

The reader is assumed to be familiar with algorithms for solving linear and nonlinear system of equations, at a level corresponding to an introductory course in numerical analysis.

The algorithms presented in the note appear in any good program library, and implementations can be found via GAMS (Guide to Available Mathematical Software) at the Internet address

http://gams.nist.gov

The examples in the note were computed in MATLAB. The programs are available via

http://www.imm.dtu.dk/~hbn/software.html

# Contents

# 1. Introduction

In this lecture note we shall discuss numerical methods for the solution of the optimization problem: For a real function of several real variables we want to find an argument vector which corresponds to a minimal function value:

| **The Optimization Problem** |
|---|
| Find $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x})$ , where $f : \mathbb{R}^n \mapsto \mathbb{R}$ |

(1.1)

The function $f$ is called the *objective function* or *cost function* and $\mathbf{x}^*$ is the *minimizer.*

In some cases we want a *maximizer* of a function. This is easily determined if we find a minimizer of the function with opposite sign.

Optimization as in (1.1) plays a very important role in many branches of science and applications: economics, operations research, network analysis, optimal design of mechanical or electrical systems, to mention but a few.

**Example 1.1.** Here we consider functions of one variable. The function

$$f(x) = (x - x^*)^2$$

has one, unique minimizer, $x^*$, see Figure 1.1.

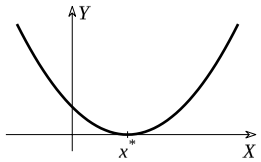



Figure 1.1: $y = (x - x^*)^2$
*One minimizer.*



The function $f(x) = -2\cos(x - x^*)$ has infinitely many minimizers: $x = x^* + 2p\pi$ , where $p$ is an integer; see Figure 1.2.

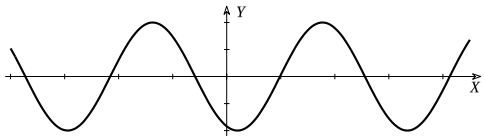



Figure 1.2: $y = -2\cos(x - x^*)$. *Many minimizers.*

The function $f(x) = 0.015(x - x^*)^2 - 2\cos(x - x^*)$ has a unique global minimizer, $x^*$. Besides that, it also has several socalled *local minimizers,* each giving the minimal function value inside a certain region, see Figure 1.3.

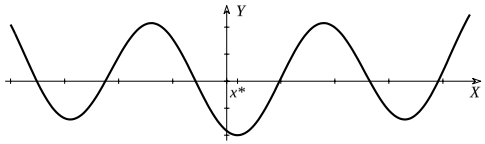



Figure 1.3: $y = 0.015(x - x^*)^2 - 2\cos(x - x^*)$
*One global minimizer and many local minimizers.*

The ideal situation for optimization computations is that the objective function has a unique minimizer. We call this the *global minimizer.*

In some cases the objective function has several (or even infinitely many) minimizers. In problems like this it may be sufficient for us to find one of these minimizers.

In many objective functions from applications we have a global minimizer and several local minimizers. It is very difficult to develop methods which can find the global minimizer with certainty in this situation. Methods for global optimization are very complicated and outside the scope of this note.

The methods described here can find a local minimizer for the objective function. When a local minimizer has been discovered, we do not know whether it is a global minimizer or one of the local minimizers. We cannot even be sure that our optimization method will find the local minimizer closest to the starting point. In order to explore several local minimizers we can try several runs with different starting points, or better still examine intermediate results produced by a global minimizer.

We end this section with an example meant to demonstrate that optimization methods based on too primitive ideas may be dangerous.

**Example 1.2.** We want the global minimizer of the function

$$f(\mathbf{x}) \;=\; (x_1 + x_2 - 2)^2 + 100(x_1 - x_2)^2 \;.$$

The idea (which we should **not** use) is the following:

"Make a series of iterations. In each iteration we keep one of the variables fixed and seek a value of the other variable so as to minimize the $f$-value". In Figure 1.4 we show the *level curves* or *contours* of $f$, i.e. curves consisting of positions with the same $f$-value. We also show the first few iterations.

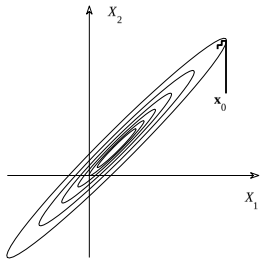



Figure 1.4: *The Method of Alternating Variables fails to determine the minimizer of a quadratic*

After some iterations the steps begin to decrease rapidly in size. They can become so small that they do not influence the **x**-values, because these are represented with a finite precision in the computer, and the progress stops completely. In many cases this happens far away from the solution. We say that the iteration is caught in *Stiefel's cage.*



The "method" is called the *method of alternating variables* and it is a classical example of a dangerous method, a method we must avoid.

---

## 1.1. Conditions for a Local Minimizer

A local minimizer for $f$ is an argument vector giving the smallest function value inside a certain region, defined by $\varepsilon$ :

**Definition**

$$\mathbf{x}^* \text{ is a } \textit{local minimizer} \text{ for } f : \mathbb{R}^n \mapsto \mathbb{R}$$
$$\Longleftrightarrow$$
$$f(\mathbf{x}^*) \le f(\mathbf{x}) \quad \text{if} \quad \|\mathbf{x}^* - \mathbf{x}\| \le \varepsilon \quad (\varepsilon > 0) \tag{1.2}$$

Most objective functions, especially those with several local minimizers, contain local maximizers and other points which satisfy a necessary condition for a local minimizer. The following theorems help us find such points and distinguish the local minimizers from the irrelevant points.

We assume that $f$ has continuous partial derivatives of second order. The first order *Taylor series* for a function of several variables gives us an approximation to the function value at a point $\mathbf{x}+\mathbf{h}$ neighbouring $\mathbf{x}$,

$$f(\mathbf{x}+\mathbf{h}) \;=\; f(\mathbf{x}) + \mathbf{h}^\top \mathbf{f}'(\mathbf{x}) + O(\|\mathbf{h}\|^2) \;, \tag{1.3}$$

where $\mathbf{f}'(\mathbf{x})$ is the *gradient* of $f$, a vector containing the first partial derivatives,

$$\mathbf{f}'(\mathbf{x}) \;\equiv\; \begin{bmatrix} \dfrac{\partial f}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \dfrac{\partial f}{\partial x_n}(\mathbf{x}) \end{bmatrix} \;. \tag{1.4}$$

We only consider vectors $\mathbf{h}$ with $\|\mathbf{h}\|$ so small that the last term in (1.3) is negligible compared with the middle term.

If our point $\mathbf{x}$ is a local minimizer it is not possible to find an $\mathbf{h}$ so that $f(\mathbf{x}+\mathbf{h}) < f(\mathbf{x})$ with $\|\mathbf{h}\|$ small enough. This together with (1.3) is the basis of

**Theorem 1.1**
**The Necessary Condition for a Local Minimum**

$\mathbf{x}^*$ is a local minimizer for $f : \mathbb{R}^n \mapsto \mathbb{R}$
$\Longrightarrow$
$\mathbf{f}'(\mathbf{x}^*) = \mathbf{0}$

The local minimizers are among the points with $\mathbf{f}'(\mathbf{x}) = \mathbf{0}$. They have a special name:

**Definition**

$\mathbf{x}_s$ is a *stationary point* for $f \iff \mathbf{f}'(\mathbf{x}_s) = \mathbf{0}$ (1.5)

The stationary points are the local maximizers, the local minimizers and "the rest". To distinguish between them, we need one extra term in the Taylor series. This is, provided that $f$ has continuous third derivatives,

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \mathbf{h}^\top \mathbf{f}'(\mathbf{x}) + \tfrac{1}{2}\mathbf{h}^\top \mathbf{f}''(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^3) \ , \quad (1.6)$$

where the *Hessian matrix* of function $f$ is a matrix containing the second partial derivatives of $f$ :

$$\mathbf{f}''(\mathbf{x}) \equiv \left[\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x})\right] \ . \quad (1.7)$$

Note that this is a symmetric matrix. For a stationary point (1.6) takes the form



$$f(\mathbf{x}_s+\mathbf{h}) = f(\mathbf{x}_s) + \tfrac{1}{2}\mathbf{h}^\top \mathbf{f}''(\mathbf{x}_s)\mathbf{h} + O(\|\mathbf{h}\|^3) \ . \quad (1.8)$$

If the $2^{nd}$ term is positive for all $\mathbf{h}$ we say that the matrix $\mathbf{f}''(\mathbf{x}_s)$ is *positive definite* (cf. Appendix A, which also has tools for checking definiteness). Further, we can take $\|\mathbf{h}\|$ so small that the error term is negligible, and it follows that $\mathbf{x}_s$ is a local minimizer.

**Theorem 1.2**
**The Sufficient Condition for a Local Minimum**

Assume that $\mathbf{x}_s$ is a stationary point, see Definition (1.5)
and that $\mathbf{f}''(\mathbf{x}_s)$ is positive definite
$\Longrightarrow$
$\mathbf{x}_s$ is a local minimizer

The Taylor series (1.6) is also the basis of the proof of the following

**Corrollary 1.3**

Assume that $\mathbf{x}_s$ is a stationary point and that $\mathbf{f}''(\mathbf{x})$ is positive semidefinite when $\mathbf{x}$ is in a neighbourhood of $\mathbf{x}_s$
$\Longrightarrow$
$\mathbf{x}_s$ is a local minimizer

The local maximizers and "the rest", which we call *saddle points,* can be characterized by the following corollary, also derived from (1.6).

**Corrollary 1.4**

Assume that $\mathbf{x}_s$ is a stationary point and that $\mathbf{f}''(\mathbf{x}_s) \neq \mathbf{0}$. Then

1) $\mathbf{f}''(\mathbf{x}_s)$ is positive definite: see Theorem 1.2.
2) $\mathbf{f}''(\mathbf{x}_s)$ is positive semidefinite:
   $\Rightarrow$ $\mathbf{x}_s$ is a local minimizer or a saddle point.
3) $\mathbf{f}''(\mathbf{x}_s)$ is neither definite nor semidefinite:
   $\Rightarrow$ $\mathbf{x}_s$ is a saddle point.
4) $\mathbf{f}''(\mathbf{x}_s)$ is negative semidefinite:
   $\Rightarrow$ $\mathbf{x}_s$ is a local maximizer or a saddle point.
5) $\mathbf{f}''(\mathbf{x}_s)$ is negative definite:
   $\Rightarrow$ $\mathbf{x}_s$ is a local maximizer.

If $\mathbf{f}''(\mathbf{x}_s) = \mathbf{0}$, then we need higher order terms in the Taylor series in order to find the local minimizers among the stationary points.

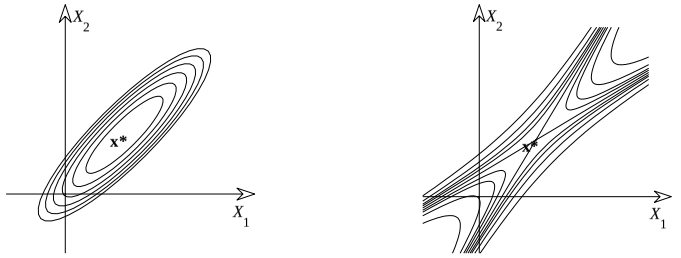
**Example 1.3.** We consider functions of two variables. Below we show the variation of the function value near a local minimizer (Figure 1.5a), a local maximizer (Figure 1.5b) and a saddle point (Figure 1.5c). It is a characteristic of a saddle point that there exists one line through $\mathbf{x}_s$, with the property that if we follow the variation of the $f$-value along the line, this "looks like" a local minimum, whereas there exists another line through $\mathbf{x}_s$, "indicating" a local maximizer.

*a) minimum* *b) maximum* *c) saddle point*

Figure 1.5: *With a 2-dimensional* $\mathbf{x}$ *we see surfaces* $z = f(\mathbf{x})$ *near a stationary point*



If we study the level curves of our function, we see curves approximately like concentric ellipses near a local maximizer or a local minimizer (Figure 1.6a), whereas the saddle points exhibit the "hyperbolae" shown in Figure 1.6b.




*a) maximum or minimum* *b) saddle point*

Figure 1.6: *The contours of a function near a stationary point*

Finally, the Taylor series (1.6) is also the basis for the following:

**Theorem 1.5**
**Second Order Necessary Condition**

$\mathbf{x}^*$ is a local minimizer
$\Longrightarrow$
$\mathbf{f}''(\mathbf{x}^*)$ is positive semidefinite

# 2. Descent Methods

All the methods in this lecture note are *iterative methods.* They produce a series of vectors

$$\mathbf{x}_0,\ \mathbf{x}_1,\ \mathbf{x}_2,\ \ldots\ , \tag{2.1a}$$

which in most cases converges under certain mild conditions. We want the series to converge towards $\mathbf{x}^*$, a local minimizer for the given objective function $f : \mathbb{R}^n \mapsto \mathbb{R}$ , i.e.

$$\mathbf{x}_k\ \rightarrow\ \mathbf{x}^* \quad \text{for} \quad k \rightarrow \infty\ , \tag{2.1b}$$

where $\mathbf{x}^*$ is a local minimizer, see definition (1.2).

In all (or nearly all) the methods there are measures which enforce the descending property

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)\ . \tag{2.2}$$

This prevents convergence to a maximizer and also makes it less probable that we get convergence to a saddle point, see Chapter 1. We talk about the *global convergence* properties of a method, i.e. convergence when the iteration starts in a position, $\mathbf{x}_0$, which is not close to a local minimizer, $\mathbf{x}^*$. We want our method to produce iterates that move steadily towards a neighbourhood of $\mathbf{x}^*$. For instance, there are methods for which it is possible to prove that any accumulation point (i.e. limit of a subseries) of $\{\mathbf{x}_k\}$ is a stationary point, see (1.5), i.e. the gradients tend to zero:

$$\mathbf{f}'(\mathbf{x}_k) \rightarrow \mathbf{0} \quad \text{for} \quad k \rightarrow \infty\ . \tag{2.3}$$



This does not exclude convergence to a saddle point or even a maximizer, but the descending property (2.2) prevents this in practice. In this "global part" of the iteration we are satisfied if the current errors do not increase except for the very first steps. Letting $\{\mathbf{e}_k\}$ denote the errors,

$$\mathbf{e}_k \equiv \mathbf{x}^* - \mathbf{x}_k\ ,$$

the requirement is

$$||\mathbf{e}_{k+1}|| < ||\mathbf{e}_k|| \quad \text{for} \quad k > K\ .$$

In the final stages of the iteration where the $\mathbf{x}_k$ are close to $\mathbf{x}^*$ we expect faster convergence. The local convergence results tell us how quickly we can get a result which agrees with $\mathbf{x}^*$ to a desired accuracy. Some methods have *linear convergence,* i.e.

$$||\mathbf{e}_{k+1}|| \leq c_1||\mathbf{e}_k|| \quad \text{with } c_1 < 1 \text{ and } \mathbf{x}_k \text{ close to } \mathbf{x}^*\ . \tag{2.4}$$

It is more desirable to have higher order of convergence, for instance *quadratic convergence* (convergence of order 2):

$$||\mathbf{e}_{k+1}|| \leq c_2||\mathbf{e}_k||^2 \quad \text{with } c_2 > 0 \text{ and } \mathbf{x}_k \text{ close to } \mathbf{x}^*\ . \tag{2.5}$$

Only a few of the methods used in the applications achieve quadratic final convergence. On the other hand we want better than linear final convergence. Many of the methods used in practice have *superlinear convergence:*

$$\frac{||\mathbf{e}_{k+1}||}{||\mathbf{e}_k||} \rightarrow 0 \quad \text{for } k \rightarrow \infty\ . \tag{2.6}$$

This is better than linear convergence though (normally) not as good as quadratic convergence.

**Example 2.1.** Consider 2 iterative methods, one with linear and one with quadratic convergence. At a given step they have both achieved the result with an accuracy of 3 decimals:

$$\|\mathbf{e}_k\| \ < \ 0.001$$

They have $c_1 = c_2 = \frac{1}{2}$ in (2.4) and (2.5) respectively. If we want an accuracy of 12 decimals, the iteration with quadratic convergence will only need 2 more steps, whereas the iteration with linear convergence will need about 30 more steps, $\left(\frac{1}{2}\right)^{30} \simeq 10^{-9}$.

---

## 2.1. Fundamental Structure of a Descent Method

**Example 2.2.** This is a 2-dimensional minimization example. A tourist has lost his way in a hilly country. It is a foggy day so he cannot see far and he has no map. He knows that his rescue is at the bottom of a nearby valley. As tools he has an altimeter, a compass and his sense of balance together with a spirit level which can tell him about the slope of the ground locally.

In order not to walk in circles he decides to use straight strides, i.e. with constant compass bearing. From what his feet tell him about the slope locally he chooses a direction and walks in that direction as long as his altimeter tells him that he gets downhill. He stops when his altimeter indicates increasing altitude, or his feet tell him that he is on an uphill slope.

Now he has to decide on a new direction and he starts his next stride. Let us hope he is saved in the end.

---

The pattern of events in the example above is the basis of the algorithms for descent methods:



**Algorithm 2.7. Descent Method**

```
begin
  k := 0;  x := x_0;  found := false                    {Starting point}
  repeat
    h_dh := search_direction(x)                         {From x and downhill }
    if no such h exists
      found := true                                     {x is stationary}
    else
      α := line_search(x, h_dh)                         {from x in direction h_dh}
      x := x + α h_dh                                   {new position }
      k := k + 1
      found := update(found)
  until  found or k>k_max
end                                                     {... of descent algorithm }
```

The search direction must be a descent direction. Then we are able to gain a smaller value of $f(\mathbf{x})$ by choosing an appropriate walking distance, and thus we can satisfy the descending condition (2.2). For details, see Sections 2.2 and 2.5 – 2.6.

As *stopping criterion* we would like to use the ideal criterion that the current error is sufficiently small

$$\|\mathbf{e}_k\| < \delta_1 \ .$$

Another ideal condition would be that the current value of $f(\mathbf{x})$ is close enough to the minimal value, i.e.

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) < \delta_2 \ .$$

Both conditions reflect the convergence $\mathbf{x}_k \to \mathbf{x}^*$. They cannot be used in real applications, however, because $\mathbf{x}^*$ and $f(\mathbf{x}^*)$ are not known. Instead we have to use approximations to these conditions:

$$\|\mathbf{x}_{k+1}-\mathbf{x}_k\| < \varepsilon_1 \quad \text{or} \quad f(\mathbf{x}_k)-f(\mathbf{x}_{k+1}) < \varepsilon_2 \ . \tag{2.8}$$

We must emphasize that even if (2.8) is fulfilled with small $\varepsilon_1$ and $\varepsilon_2$, we cannot be sure that $\|\mathbf{e}_k\|$ or $f(\mathbf{x}_k)-f(\mathbf{x}^*)$ are small.

The other type of convergence mentioned at the start of this chapter is $\mathbf{f}'(\mathbf{x}_k)\to 0$ for $k\to\infty$. This can be reflected in the stopping criterion

$$\|\mathbf{f}'(\mathbf{x}_k)\| < \varepsilon_3 \ , \tag{2.9}$$

which is included in many implementations of descent methods.

There is a good way of using the property of converging function values. The Taylor series (1.6) of $f$ at $\mathbf{x}^*$ is

$$f(\mathbf{x}_k) \simeq f(\mathbf{x}^*) + (\mathbf{x}_k-\mathbf{x}^*)^\top \mathbf{f}'(\mathbf{x}^*) + \tfrac{1}{2}(\mathbf{x}_k-\mathbf{x}^*)^\top \mathbf{f}''(\mathbf{x}^*)(\mathbf{x}_k-\mathbf{x}^*) \ .$$

Now, if $\mathbf{x}^*$ is a local minimizer, then $\mathbf{f}'(\mathbf{x}^*)=\mathbf{0}$ and $\mathbf{H}^*\equiv\mathbf{f}''(\mathbf{x}^*)$ is positive semidefinite, see Chapter 1. This gives us

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \simeq \tfrac{1}{2}(\mathbf{x}_k-\mathbf{x}^*)^\top \mathbf{H}^*(\mathbf{x}_k-\mathbf{x}^*) \ ,$$

so the stopping criterion could be

$$\tfrac{1}{2}(\mathbf{x}_{k+1}-\mathbf{x}_k)^\top \mathbf{H}_k(\mathbf{x}_{k+1}-\mathbf{x}_k) < \varepsilon_4 \quad \text{with} \quad \mathbf{x}_k \simeq \mathbf{x}^* \ . \tag{2.10}$$

Here $\mathbf{x}_k-\mathbf{x}^*$ is approximated by $\mathbf{x}_{k+1}-\mathbf{x}_k$ and $\mathbf{H}^*$ is approximated by $\mathbf{H}_k=\mathbf{f}''(\mathbf{x}_k)$.

## 2.2. Descent Directions

Now we come to the important question: "How do we find a direction which brings us downhill, a descent direction ?" A necessary condition is, that if we move from the current position to a neighbouring point in the given direction we get into a position with a smaller function value.

**Example 2.3.** Let us return to our tourist who is lost in the fog in a hilly country. By experimenting with his compass he can find out that "half" the compass bearings give strides that start uphill and that the "other half" gives strides that start downhill. Between the two halves are two strides which start off going neither uphill or downhill. These form the tangent to the level curve corresponding to his position.



The Taylor series (1.3) gives us a first order approximation to the function value in a neighbouring point to $\mathbf{x}$ in direction $\mathbf{h}$:

$$f(\mathbf{x}+\alpha\mathbf{h}) = f(\mathbf{x}) + \alpha\mathbf{h}^\top\mathbf{f}'(\mathbf{x}) + O(\alpha^2), \quad \text{with} \ \ \alpha>0 \ .$$

If $\alpha$ is not too large, then the first two terms will dominate over the last:

$$f(\mathbf{x}+\alpha\mathbf{h}) \simeq f(\mathbf{x}) + \alpha\mathbf{h}^\top\mathbf{f}'(\mathbf{x}) \ .$$

The sign of the term $\alpha\mathbf{h}^\top\mathbf{f}'(\mathbf{x})$ decides whether we start off uphill or downhill. In our space $\mathbb{R}^n$ we consider a hyperplane $\mathcal{H}$ through the current position and orthogonal to $-\mathbf{f}'(\mathbf{x})$,

$$\mathcal{H} = \{\mathbf{x}+\mathbf{h} \mid \mathbf{h}^\top\mathbf{f}'(\mathbf{x}) = 0\} \ .$$

This hyperplane divides the space in an "uphill" halfspace and a "downhill" halfspace. The halfspace we want has the vector $-\mathbf{f}'(\mathbf{x})$ pointing into it. Figure 2.1 gives the situation in $\mathbb{R}^3$.

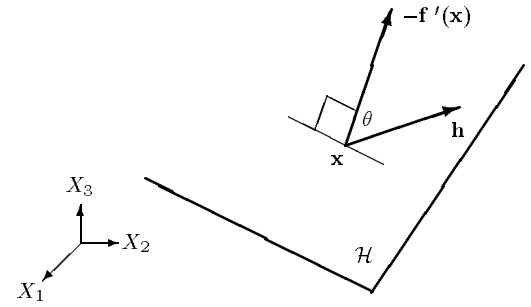



Figure 2.1: $\mathbb{R}^3$ *divided into a "downhill" and an "uphill" halfspace.*

We now define a descent direction. This is a "downhill" direction, i.e. it is inside the "good" halfspace:

**Definition**

$\mathbf{h}$ is a *descent direction* from $\mathbf{x}$ $\iff$ $\mathbf{h}^\top \mathbf{f}'(\mathbf{x}) < 0$ (2.11)

A method based on condition (2.11) is a *descent method*.

In Figure 2.1 we have a descent direction $\mathbf{h}$, satisfying (2.11). We introduce the angle between $\mathbf{h}$ and $-\mathbf{f}'(\mathbf{x})$

$$\theta = \angle(\mathbf{h}, -\mathbf{f}'(\mathbf{x})) \quad \text{with} \quad \cos\theta = \frac{-\mathbf{h}^\top \mathbf{f}'(\mathbf{x})}{\|\mathbf{h}\| \cdot \|\mathbf{f}'(\mathbf{x})\|} \quad . \tag{2.12}$$

We state a new condition on this angle,

**Definition**

An *absolute descent method* has search directions $\mathbf{h}_k$, which satisfy

$$\theta < \tfrac{\pi}{2} - \mu$$

for all $k$, with $\mu > 0$ independent of $k$ (2.13)

The discussion above is concerned with the geometry in $\mathbb{R}^3$, and is easily seen to be valid also in $\mathbb{R}^2$. If the dimension $n$ is larger than 3, we call $\theta$ "the *pseudoangle* between $\mathbf{h}$ and $-\mathbf{f}'(\mathbf{x})$". In this way we can use (2.12) and (2.13), for all $n \geq 2$.

The restriction that $\mu$ must be constant in all the steps is necessary for the global convergence result we give in the next section.

## 2.3. Descent Methods with Line Search

When a descent direction has been determined, we have to decide how long the step in this direction should be. We perform a line search as indicated in Algorithm 2.7. First, we must be sure that the descending condition (2.2) is satisfied. Next, we must guard against



the step being so short that our gain in function value diminishes. We study the variation of the objective function $f$ along the direction $\mathbf{h}$ from the current position $\mathbf{x}$

$$\varphi(\alpha) = f(\mathbf{x}+\alpha\mathbf{h}), \quad \text{with fixed } \mathbf{x} \text{ and } \mathbf{h} \ .$$

From the Taylor series (1.6) it follows that

$$\varphi(\alpha) = f(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{f}'(\mathbf{x}) + \tfrac{1}{2}\alpha^2 \mathbf{h}^\top \mathbf{f}''(\mathbf{x})\mathbf{h} + O(\alpha^3)$$

and

$$\varphi'(0) = \mathbf{h}^\top \mathbf{f}'(\mathbf{x}) \ . \tag{2.14}$$

In Figure 2.2 we show an example of the variation of $\varphi(\alpha)$ with $\mathbf{h}$ as a descent direction. The descending condition (2.2) implies that we want to stop the line search with a value $\alpha_s$ so that $\varphi(\alpha_s) < \varphi(0)$. According to (2.14) have $\varphi'(0) < 0$, but the figure shows that there is a risk that, if $\alpha$ is taken too large, then $\varphi(\alpha) > \varphi(0)$.

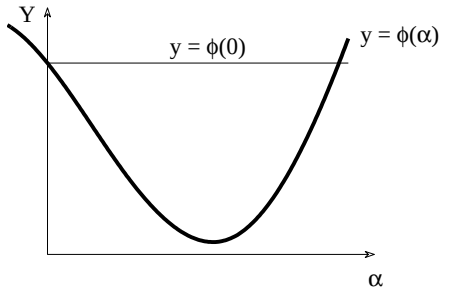



Figure 2.2: *Variation of the cost function along the search line*

To ensure that we get a useful decrease in $f$-value, we stop the search with a value $\alpha_s$ which gives a $\varphi$-value below that of the line $y = \lambda(\alpha)$, indicated in Figure 2.3. This line goes through the starting point and has a slope which is a fraction of the slope of the starting tangent to the $\varphi$-curve:

$$\varphi(\alpha_s) \le \lambda(\alpha_s) \ , \qquad \text{where}$$
$$\lambda(\alpha) = \varphi(0) + \varrho \cdot \varphi'(0) \cdot \alpha \quad \text{with} \quad 0 < \varrho < 0.5 \ . \tag{2.15}$$

The parameter $\varrho$ is normally small, 0.001 can be a good value. Condition (2.15) is needed in some convergence proofs.

We also want to ensure that the $\alpha$-value is not chosen too small. In Figure 2.3 we indicate a requirement, ensuring that the local slope is greater than the starting slope. More specificly,

$$\varphi'(\alpha_s) \ge \beta \cdot \varphi'(0) \quad \text{with } \varrho < \beta < 1 \ . \tag{2.16}$$

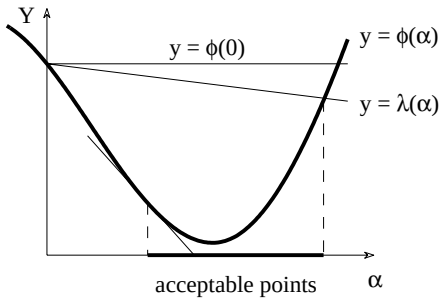



Figure 2.3: *Acceptable points according to criteria (2.15) and (2.16)*

Descent methods with line search governed by (2.15) plus (2.16) are normally convergent. Fletcher (1987), pp 26–30, has the proof of Theorem 2.1 below.

A possible outcome is that the method finds a stationary point ($\mathbf{x}_k$ with $\mathbf{f}'(\mathbf{x}_k) = \mathbf{0}$) and then it stops. Another possibility is that $f(\mathbf{x})$ is not bounded from below for $\mathbf{x}$ in the level set $\{\mathbf{x} \mid f(\mathbf{x}) < f(\mathbf{x}_0)\}$ and the method may "fall into the hole". If neither of these occur, the method converges towards a stationary point. The method being a descent method often makes it converge towards a point which is not only a stationary point but also a local minimizer.



**Theorem 2.1**

Consider an absolute descent method following Algorithm 2.7 with search directions according to (2.12) and (2.13) and with line search controlled by (2.15) and (2.16).

If $\mathbf{f}'(\mathbf{x})$ exists and is uniformly continuous on the level set $\{\mathbf{x} \mid f(\mathbf{x}) < f(\mathbf{x}_0)\}$, then for $k \to \infty$:

either $\mathbf{f}'(\mathbf{x}_k) = \mathbf{0}$ for some $k$
or $f(\mathbf{x}_k) \to -\infty$
or $\mathbf{f}'(\mathbf{x}_k) \to \mathbf{0}$

A line search as described above is often called a *soft line search* because of its liberal stopping criteria, (2.15) and (2.16). In contrast to this there are variants which we call *"exact line searches"*, exact in the sense that we seek an approximation to a local minimizer for $\varphi(\alpha)$, i.e.

$$\alpha_e = \operatorname{argmin}_{\alpha>0} f(\mathbf{x}+\alpha\mathbf{h}) \quad \text{for fixed} \quad \mathbf{x} \text{ and } \mathbf{h} \ . \tag{2.17}$$

A necessary condition on $\alpha_e$ is $\varphi'(\alpha_e) = 0$ . We have $\varphi'(\alpha) = \mathbf{h}^\top \mathbf{f}'(\mathbf{x}+\alpha\mathbf{h})$ and this shows that either $\mathbf{f}'(\mathbf{x}+\alpha_e\mathbf{h}) = \mathbf{0}$, which is a perfect result (we have found a stationary point for $f$), or if $\mathbf{f}'(\mathbf{x}+\alpha_e\mathbf{h}) \ne \mathbf{0}$, then $\varphi'(\alpha_e) = 0$ leads to:

$$\mathbf{f}'(\mathbf{x}+\alpha_e\mathbf{h}) \perp \mathbf{h} \ . \tag{2.18}$$

This shows that the exact line search will stop at a point where the local gradient is orthogonal to the search direction.

**Example 2.4.** A "divine power" with a radar set follows the movements of our wayward tourist. He has decided to continue in a given direction, until his feet or his altimeter tells him that he starts to go uphill. The "divine power" can see that he stops where the given direction is tangent to a local contour. This is equivalent to the orthogonality mentioned in (2.18).

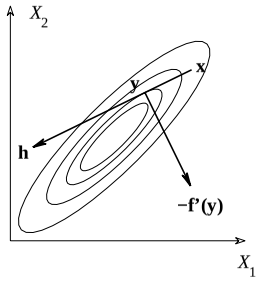



Figure 2.4: *An exact line search stops at* $\mathbf{y} = \mathbf{x}+\alpha_e\mathbf{h}$*, where the local gradient is orthogonal to the search direction*

For further details about line searches, see Sections 2.5 – 2.6. There are several disadvantages to exact line search. Firstly, it is more time consuming than soft line search. It contains iterative refinement of an approximation to the minimizer along our direction. This can take quite a lot of time. Even if an exact line search finds the solution in its first try, in some cases it will perform several steps of computation in order to check its stopping criterion. Its second disadvantage is shown in the next example.

**Example 2.5.** Our wayward tourist has determined to go by exact line searches. Walking in the given direction towards the lowest point in that direction, our tourist may feel a steep descent across his path.

This will make him want to start on a new search direction before he arrives at the bottom in his first direction.

The example hinted that it is often a good idea to use a step (in the given direction) which is shorter than the step resulting from an exact line search. This is one of the reasons behind the class of methods given in the next section, methods with no line searches.



## 2.4. Descent Methods with Trust Region

The methods in this note produce series of steps leading from the starting position to the final result, we hope. In the descent methods of this chapter and in Newton's method of Chapter 5, the directions of the steps are determined by the properties of $f(\mathbf{x})$ at the current position. Similar considerations lead us to the trust region methods, where the iteration steps are determined from the properties of a model of the objective function inside a given region. The size of the region is modified during the iteration.

The Taylor series (1.3) provides us with a linear approximation to $f$ near a given $\mathbf{x}$:

$$f(\mathbf{x}+\mathbf{h}) \simeq q(\mathbf{h}) \quad \text{with} \quad q(\mathbf{h}) = f(\mathbf{x}) + \mathbf{h}^\top \mathbf{f}'(\mathbf{x}) \ . \tag{2.19}$$

Likewise we can obtain a quadratic approximation to $f$ from the Taylor series (1.6)

$$\begin{aligned} &f(\mathbf{x}+\mathbf{h}) \simeq q(\mathbf{h}) \\ &\quad \text{with} \quad q(\mathbf{h}) = f(\mathbf{x}) + \mathbf{h}^\top \mathbf{f}'(\mathbf{x}) + \tfrac{1}{2}\mathbf{h}^\top \mathbf{f}''(\mathbf{x})\mathbf{h} \ . \end{aligned} \tag{2.20}$$

In both case $q(\mathbf{h})$ is a poor approximation to $f(\mathbf{x}+\mathbf{h})$ unless $\|\mathbf{h}\|$ is sufficiently small. These considerations lead us to determine the new iteration step as the solution to the following model problem:

$$\begin{aligned} &\mathbf{h}_{\text{tr}} = \text{argmin}_{\mathbf{h}\in\mathcal{D}}\{q(\mathbf{h})\} \\ &\quad \text{where} \quad \mathcal{D} = \{\mathbf{h} \mid \|\mathbf{h}\| \le \triangle\}, \quad \triangle > 0 \ . \end{aligned} \tag{2.21}$$

The region $\mathcal{D}$ is called the *trust region* and $q(\mathbf{h})$ is given by (2.19) or (2.20).

We use $\mathbf{h} = \mathbf{h}_{\text{tr}}$ as a candidate to our next step, and reject $\mathbf{h}$, if $f(\mathbf{x}+\mathbf{h}) \ge f(\mathbf{x})$. The gain in cost function value controls the size of the trust region for the next step: The gain is compared with the gain predicted by the approximation function, and we introduce the *gain factor*:

$$r = \frac{f(\mathbf{x}) - f(\mathbf{x}+\mathbf{h})}{q(\mathbf{0}) - q(\mathbf{h})} \quad . \tag{2.22}$$

When $r$ is small our approximation agrees poorly with $f$, and when it is large the agreement is good. Thus we let the gain factor regulate the size of the trust region for the next step (or our next attempt for this step when $r \leq 0$ and $\mathbf{h}$ is rejected).

We now have the basis for

**Algorithm 2.23. Descent Method with Trust Region**

```
begin
  k := 0;  x := x_0;  △ := △_0;  found := false        {starting point}
  repeat
    k := k+1;  h_tr := Solution of model problem (2.21)
    r := gain factor (2.22)
    if r > 0.75                                        {step very good}
      △ := 2 * △                                       {larger trust region}
    if r < 0.25                                        {step not very good}
      △ := △/3                                         {smaller trust region}
    if r > 0                                           {reject step if r ≤ 0}
      x := x + h_tr
    Update found                {stopping criteria, e.g. (2.8) and (2.9)}
  until found or k>k_max
end
```

The numbers in the algorithm, 0.75, 2, 0.25 and 1/3 have been chosen from practical experience. The method is not very sensitive to minor changes in these values, but in the expressions $\triangle := p_1 * \triangle$ and $\triangle := \triangle / p_2$ the numbers $p_1$ and $p_2$ must be chosen so that the $\triangle$-values cannot oscillate.

There are versions of the trust region method where "$r$<0.25" initiates an interpolation between $\mathbf{x}$ and $\mathbf{x}$+$\mathbf{h}$ based on known values of $f$ and $\mathbf{f}'$, and/or "$r$>0.75" leads to an extrapolation along the direction $\mathbf{h}$, a line search actually. Actions like this can be rather costly,



and in his book, Fletcher (1987), p. 96, claims that the improvements in performance may be marginal. In the same reference there are theorems about the global performance of methods like 2.23.

## 2.5. Soft Line Search

Many researchers in optimization have proved their inventiveness by producing new line search methods or modifications to known methods. What we present here are useful combinations of ideas of different origin. The description is based on Madsen (1984).

In the early days of optimization the exact line searches were dominant. Now, the soft line searches are used more and more, and we rarely see new methods presented which require exact line searches.

An advantage of soft line search over exact line search is that it is the faster of the two. If the first guess on the step length is a rough approximation to the minimizer along the given direction, the linesearch will terminate immediately if some mild criteria are satisfied. The result of the exact line search is normally a good approximation to the result, and this can make descent methods with exact line search find the local minimizer in fewer iterations than used by a descent method with soft line search. Still, the extra time spent in each line search often makes the descent method with exact line search a loser.

If we are at the start of the iteration with a descent method, where $\mathbf{x}$ is far from the solution $\mathbf{x}^*$, it does not matter much that the result of the soft line search is only a rough approximation to the result; this is another point in favour of the soft line search.

The purpose of the algorithm is to find $\alpha_s$, an acceptable argument for the function

$$\varphi(\alpha) = f(\mathbf{x} + \alpha\mathbf{h}) \; .$$

The acceptability is decided by the criteria (2.15),

$$\begin{aligned} &\varphi(\alpha_s) \le \lambda(\alpha_s)\ , \qquad \text{where} \\ &\lambda(\alpha) = \varphi(0) + \varrho \cdot \varphi'(0) \cdot \alpha \quad \text{with} \ \ 0 < \varrho < 0.5 \end{aligned} \tag{2.24a}$$

and (2.16),

$$\varphi'(\alpha_s) \ge \beta \cdot \varphi'(0) \quad \text{with } \varrho < \beta < 1\ . \tag{2.24b}$$

These two criteria express the demands that $\alpha_s$ must be sufficiently small to give a useful decrease in the objective function, and sufficiently large to ensure that we have left the starting tangent of the curve $y = \varphi(\alpha)$ for $\alpha \ge 0$; cf. Figure 2.3.

The algorithm has two parts. First we find an interval $[a, b]$ that contains acceptable points, see figure 2.5:

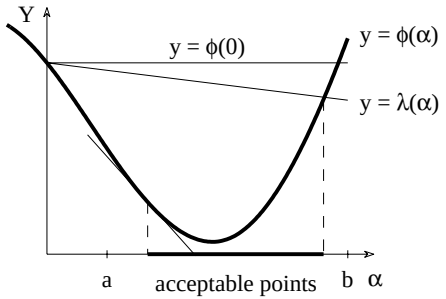



Figure 2.5: *Interval $[a, b]$ containing acceptable points*

In the second part of the algorithm we successively reduce the interval: We find a point $\alpha$ in the strict interior of $[a, b]$. If both conditions (2.24) are satisfied by this $\alpha$-value, then we are finished ($\alpha_s = \alpha$). Otherwise, the reduced interval is either $[a, b] := [a, \alpha]$ or $[a, b] := [\alpha, b]$, where the choice is made so that the reduced $[a, b]$ contains acceptable points.



**Algorithm 2.25. Soft Line Search**

```
begin
  if φ'(0) ≥ 0                                                    {1°}
    α := 0
  else
    k := 0;  γ := β * φ'(0);
    a := 0;  b := min{1, α_max}                                   {2°}
    while (φ(b) ≤ λ(b)) and (φ'(b) ≤ γ)
          and (b < α_max) and (k < k_max)
      k := k + 1;  a := b                                         {3°}
      b := min{2b, α_max}                                         {4°}
    α := b                                                        {5°}
    while ((φ(α) > λ(α)) or (φ'(α) < γ)) and (k < k_max)
      k := k + 1
      Refine α and [a, b]                                         {6°}
    if φ(α) ≥ φ(0)                                                {7°}
      α := 0
end
```

We have the following remarks:

1° If $\mathbf{x}$ is a stationary point ($\mathbf{f}'(\mathbf{x}) = \mathbf{0} \Rightarrow \varphi'(0) = 0$) or $\mathbf{h}$ is not downhill, then we do nothing.

2° The initial choice $b = 1$ is used because in many optimization methods (e.g. Newton's method in Chapter 5) $\alpha = 1$ is a very good guess in the final steps of the iteration. The upper bound $\alpha_{\max}$ must be supplied by the user. It acts as a guard against an infinite loop if $f$ is unbounded.

3° We are to the left of a minimum and update the left hand end of the interval $[a, b]$.

4° If $\alpha_{\max}$ is sufficiently large, then the series of $b$-values is $1, 2, 4, \ldots$, corresponding to an "expansion factor" of 2. Other factors could be used.

5° Initialization for second part of the algorithm.

6° See Function 2.26.

7° The algorithm may have stopped abnormally, e.g. by exceeding the permitted number $k_{\max}$ of function evaluations. If the current value of $\alpha$ does not decrease the objective function, then we return $\alpha = 0$, cf. 1°.

The following Function 2.26 receives an interval $[a, b]$ which we know contains acceptable points. It produces an $\alpha$ using interpolation. We want to be sure that the intervals have strictly decreasing widths, so we only consider the new $\alpha$ if it is inside $[a{+}d, b{-}d]$, where $d = \frac{1}{10}(b-a)$. The $\alpha$ splits $[a, b]$ into two subintervals, and we return the subinterval which must contain acceptable points.

**Function 2.26. Refine**

```
begin
  D := b − a;   c := (φ(b) − φ(a) − D ∗ φ'(a))/D²                {8°}
  if c > 0
    α := a − φ'(a)/(2c)
  else
    α := (a + b)/2
  α := max{α, a+0.1D};   α := min{α, b−0.1D}                     {9°}
  if φ(α) < λ(α)                                                 {10°}
    a := α
  else
    b := α
end
```



We have the following remarks

8° The second order polynomial

$$\psi(t) = \varphi(a) + \varphi'(a) \cdot (t{-}a) + c \cdot (t{-}a)^2$$

satisfies $\psi(a) = \varphi(a)$, $\psi'(a) = \varphi'(a)$ and $\psi(b) = \varphi(b)$. If $c > 0$, then $\psi$ has a minimum, and we let $\alpha$ be the minimizer. Otherwise we take $\alpha$ as the midpoint of $[a, b]$.

9° Ensure that $\alpha$ is in the middle 80% of the interval.

10° If $\varphi(\alpha)$ is sufficiently small, then the right hand part of $[a, b]$ contain points that satisfy **both** of the constraints (2.24). Otherwise, $[\alpha, b]$ is sure to contain acceptable points.

Finally, we give the following remarks about the implementation of the algorithm.

The function and slope values are computed as

$$\varphi(\alpha) = f(\mathbf{x}{+}\alpha\mathbf{h}), \qquad \varphi'(\alpha) = \mathbf{h}^\top \mathbf{f}'(\mathbf{x}{+}\alpha\mathbf{h}) \ .$$

The computation of $f$ and $\mathbf{f}'$ is the "expensive" part of the line search. Therefore, the function and slope values should be stored in auxillary variables for use in acceptance criteria and elsewhere, and the implementation should return the value of the objective function and its gradient to the calling programme, a descent method. They will be useful as starting function value and for the starting slope in the next linesearch (the next iteration).

## 2.6. Exact Line Search

The older methods for line search produce a value of $\alpha_{\mathrm{s}}$ which is sufficiently close to the true result, $\alpha_{\mathrm{s}} \simeq \alpha_{\mathrm{e}}$ with

$$\alpha_{\mathrm{e}} \ \equiv \ \mathrm{argmin}_{\alpha \geq 0} \ \varphi(\alpha) \ .$$

The algorithm can be similar to the soft line search in 2.25, except that the refinement loop after remark 5° is changed to

$$\begin{array}{l}\texttt{while } \big(|\varphi'(\alpha)| > \tau * |\varphi'(0)|\big) \\ \qquad \texttt{and } \big(b{-}a > \varepsilon\big) \texttt{ and } \big(k < k_{\max}\big) \\ \quad \ldots\end{array} \tag{2.27}$$

Here, $\varepsilon$ and $\tau$ indicate the level of errors tolerated; both should be small positive numbers.

An advantage of an exact line search is that (in theory at least) it can produce its results exactly, and this is needed in some theoretical convergence results concerning conjugate gradient methods, see Chapter 4.

The disadvantages are numerous. It normally takes far more time per search direction than soft line searches do. Also, as indicated in Example 2.5, it can lead to an increased number of search directions.

# 3. The Steepest Descent Method

Until now we have not answered an important question connected with algorithm 2.7: Which of the possible descent directions (see definition (2.11)) do we choose as search direction?

Our first considerations will be based purely on local first order information. Which descent direction gives us the greatest gain in function value relative to the step length? Using the first order Taylor series (1.3) we get the following approximation

$$\frac{f(\mathbf{x}) - f(\mathbf{x}+\alpha\mathbf{h})}{\alpha\|\mathbf{h}\|} \simeq -\frac{\mathbf{h}^\top \mathbf{f}'(\mathbf{x})}{\|\mathbf{h}\|} = |\mathbf{f}'(\mathbf{x})\| \cos\theta \ . \tag{3.1}$$

In the last relation we have used the definition (2.12). We see that the relative gain is greatest when the angle $\theta = 0$, i.e.

$$\mathbf{h}_{\mathrm{sd}} = -\mathbf{f}'(\mathbf{x}) \ . \tag{3.2}$$

This search direction, the negative gradient direction, is called the direction of *steepest descent.* It gives us a useful gain in function value if the step is so short that the 3rd term in the Taylor series $\big(O(\|\mathbf{h}\|^2)\big)$ is insignificant. Thus we have to stop well before we reach the minimizer along the direction $\mathbf{h}_{\mathrm{sd}}$. At the minimizer the higher order terms are big enough to have changed the slope from its negative starting value up to 0.

A descent method based on steepest descent and with a soft or an exact line search is convergent according to Theorem 2.1. If we make a method using $\mathbf{h}_{\mathrm{sd}}$ and a line search ensuring sufficiently short steps, then the global convergence will manifest itself as a very robust global

performance. The disadvantage is that the method will have linear final convergence and this will often be exceedingly slow. If we use exact line searches together with steepest descent, we invite trouble.

**Example 3.1.** We test a steepest descent method with exact line searches with the function from Example 1.2,

$$f(\mathbf{x}) \;=\; (x_1 + x_2 - 2)^2 + 100(x_1 - x_2)^2 \;.$$

Figure 3.1 gives the contours of this function.

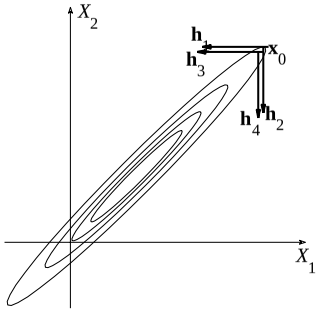



Figure 3.1: *The Steepest Descent Method fails to find the minimizer of a quadratic*

The gradient is

$$\mathbf{f}'(\mathbf{x}) = \begin{bmatrix} 2(x_1 + x_2 - 2) + 200(x_1 - x_2) \\ 2(x_1 + x_2 - 2) - 200(x_1 - x_2) \end{bmatrix} .$$

If the starting point is taken as $\mathbf{x}_0 = [3, 598/202]^\mathsf{T}$, then the first search direction is

$$\mathbf{h}_{\mathrm{sd}} = -\begin{bmatrix} 3200/202 \\ 0 \end{bmatrix} .$$

This is parallel with the $x_1$-axis. The exact line search will stop at a point where the gradient is orthogonal to this. Thus the next search direction will be parallel with the $x_2$-axis, etc. The iteration steps will be exactly as in Example 1.2. The iteration will stop far away from the solution because the steps become negligible compared with the position, when represented in the computer with a given number of digits.



The example above shows how the final linear convergence of the steepest descent method can become so slow that it makes the method completely useless when we are near the solution. We say that the iteration is caught in *Stiefel's cage.*

Still, the method is useful when we are far from the solution. It performs a little better if we make sure that the steps taken are small enough. In a version like this it is included in several modern hybrid methods, where there is a switch between two methods, one with robust global performance and one with superlinear (or even quadratic) final convergence. Under these circumstances the method of steepest descent does a very good job as the "global part" of the hybrid.

# 4. Conjugate Gradient Methods

The methods described in this chapter are the first ones that we encounter that can be called practical. They are simple and easy to implement, though perhaps not so easy to understand. Generally they are superior to the steepest descent method, but Newton's method and its relatives, that will be described in the next chapter, are usually even better. However, this is not always so, and one class of problems where conjugate gradient methods often outperform Newton-type methods are ones with very large $n$ (number of unknowns). The reason is that the Newton-type of methods rely on matrix operations, whereas conjugate gradient methods use only vectors. Ignoring sparsity, Newton's method needs $O(n^3)$ operations per iteration step, Quasi-Newton methods need $O(n^2)$, but the conjugate gradient methods use only $O(n)$ operations per iteration step. Similarly for storage: Newton-type methods require an $n \times n$ matrix to be stored, while conjugate gradient methods only need a few vectors.

The basis for the methods presented in this chapter is the following definition of *conjugate directions,* and the relevance for our problems is indicated in Example 4.1.

**Definition**

A set of directions corresponding to vectors $\{\mathbf{h}_1, \mathbf{h}_2, \ldots\}$ is *conjugate* with respect to a symmetric positive definite matrix $\mathbf{A}$

$$\Longleftrightarrow$$

$$\mathbf{h}_i^\top \mathbf{A} \mathbf{h}_j = 0 \quad \text{for all} \quad i \neq j \tag{4.1}$$



**Example 4.1.** In $\mathbb{R}^2$ we want to find the minimizer of a quadratic :

$$q(\mathbf{x}) = a + \mathbf{b}^\top \mathbf{x} + \tfrac{1}{2} \mathbf{x}^\top \mathbf{H} \mathbf{x} \, ,$$

where the matrix $\mathbf{H}$ is assumed to be positive definite. Figure 4.1 gives the contours of such a polynomial.

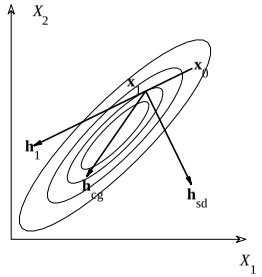



Figure 4.1: *In the 2-dimensional case, the second conjugate gradient step determines the minimizer of a quadratic*

Remember that Examples 1.2 and 3.1 showed how the methods of alternating directions and of steepest descent could be caught in Stiefel's cage and fail to find the solution $\mathbf{x}^*$.

Assume that our first step was in the direction $\mathbf{h}_1$, a descent direction. Now we have reached position $\mathbf{x}$ after an exact line search. Thus the direction $\mathbf{h}_1$ is tangent to the contour at $\mathbf{x}$. This means that $\mathbf{h}_1$ is orthogonal to the steepest descent direction $\mathbf{h}_{\text{sd}}$ at $\mathbf{x}$, i.e. $\mathbf{h}_1^\top \mathbf{h}_{\text{sd}} = 0$ :

$$\mathbf{h}_1^\top \big( (-\mathbf{q}'(\mathbf{x}) \big) = \mathbf{h}_1^\top \big( -\mathbf{b} - \mathbf{H}\mathbf{x} \big) = 0 \, .$$

Now, the minimizer satisfies $\mathbf{H}\mathbf{x}^* + \mathbf{b} = \mathbf{0}$ and inserting $\mathbf{b}$ from this we get $\mathbf{h}_1^\top \mathbf{H} (\mathbf{x}^* - \mathbf{x}) = 0$ .

This shows that if we are at $\mathbf{x}$ after an exact line search along a descent direction, $\mathbf{h}_1$, then the direction $\mathbf{x}^* - \mathbf{x}$ to the minimizer is conjugate to $\mathbf{h}_1$ with respect to $\mathbf{H}$. We can prove that the conjugate direction is a linear combination of the search direction $\mathbf{h}_1$ and the steepest descent direction, $\mathbf{h}_{\text{sd}}$, with positive coeficients, i.e. it is in the angle between $\mathbf{h}_1$ and $\mathbf{h}_{\text{sd}}$.

In the next sections we discuss conjugate gradient methods which can find the minimizer of a second degree polynomial in $n$ steps, where $n$ is the dimension of the space.

## 4.1. Quadratic models

An important tool for designing optimization methods is *quadratic modelling.* The function $f$ is approximated locally with a quadratic function $q$ of the form

$$q(\mathbf{x}) = a + \mathbf{b}^\top \mathbf{x} + \tfrac{1}{2}\mathbf{x}^\top \mathbf{H}\mathbf{x} \ , \tag{4.2}$$

where $\mathbf{H}$ is a symmetric matrix which is usually required to be positive definite.

When the modelling is direct, we simply use the minimizer of $q$ to approximate $\mathbf{x}^*$ and then repeat the process with a new approximation. This is the basis of the Newton-type methods described in Chapter 5. For the conjugate gradient methods, the model function (4.2) will be employed more indirectly.

A related concept is that of *quadratic termination,* which is said to hold for methods that find the exact minimum of the quadratic (4.2) in a finite number of steps. The steepest descent method is not quadratically terminating, but all the methods discussed in this chapter and the next are. Quadratic termination has proved to be an important idea and worth striving for in the design of optimization methods.

Because of the importance of quadratic models we now take a closer look at the quadratic function (4.2). It is not difficult to see that its gradient at $\mathbf{x}$ is given by

$$\mathbf{q}'(\mathbf{x}) = \mathbf{H}\mathbf{x} + \mathbf{b} \tag{4.3}$$

and for all $\mathbf{x}$ the Hessian is

$$\mathbf{q}''(\mathbf{x}) = \mathbf{H} \ . \tag{4.4}$$



If $\mathbf{H}$ is positive definite, then $q$ has a single minimizer at $\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$. If $n$=2, then the contours of $q$ are ellipses with centers at $\mathbf{x}^*$. The shape and orientation of the ellipses are determined by the eigenvalues and eigenvectors of $\mathbf{H}$. For $n$=3 this generalizes to ellipsoids, and in higher dimensions we get $(n-1)$-dimensional hyperellipsoids. It is of course possible to define quadratic functions with a non-positive definite Hessian, but then there is no longer a single minimizer.

Finally, a useful fact is that multiplication by $\mathbf{H}$ maps differences in $\mathbf{x}$-values to differences in the corresponding gradients:

$$\mathbf{H}(\mathbf{x} - \mathbf{z}) = \mathbf{q}'(\mathbf{x}) - \mathbf{q}'(\mathbf{z}) \ . \tag{4.5}$$

## 4.2. Structure of a Conjugate Gradient Method

Let us have another look at Figure 3.1 where the slow convergence of the steepest descent method is demonstrated. An idea for a possible cure is to take a linear combination of the previous search direction and the current steepest descent direction to get a direction toward the solution. This gives a method of the following type.

**Algorithm 4.6. Conjugate Gradient Method**

```
begin
  x := x0;  k := 0;  found := false;  γ := 0;  hcg := 0                {1°}
  repeat
    hprev := hcg;  hcg := −f′(x) + γ ∗ hprev
    if f′(x)^T hcg ≥ 0                                                 {2°}
      hcg := −f′(x)
    α := line_search(x, hcg);  x := x + α hcg                          {3°}
    γ := ···                                                           {4°}
    k := k+1;  found := ···                                            {4°}
  until found or k > kmax
end
```

We have the following remarks:

1° Initialization.

2° In most cases the vector $\mathbf{h}_{\text{cg}}$ is downhill. This is not guaranteed, e.g./ if we use a soft line search, so we use this modification to ensure that each step is downhill.

3° New iterate.

4° The formula for $\gamma$ is characteristic for the method. This is discussed in the next sections.

5° We recommend to stop if one of the criteria

$$\|\mathbf{f}'(\mathbf{x})\|_\infty \leq \varepsilon_1 \tag{4.7a}$$

$$\|\alpha \mathbf{h}_{\text{cg}}\|_2 \leq \varepsilon_2(\varepsilon_2 + \|\mathbf{x}\|_2) \tag{4.7b}$$

is satisfied, cf. (2.8) and (2.9).

In the next theorem we show that a method employing conjugate search directions and exact line searches is very good for minimizing quadratics. In Theorem 4.2 (in Section 4.3) we show that, if $f$ is quadratic and the line searches are exact, then a proper choice of $\gamma$ gives conjugate search directions.

**Theorem 4.1**

Use Algorithm 4.6 with exact line searches on a quadratic like (4.2) with $\mathbf{x} \in \mathbb{R}^n$. The iterates are $\mathbf{x}_1$, $\mathbf{x}_2$, ... with the iteration steps $\mathbf{h}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$ corresponding to conjugate directions. Then

1° The search directions $\mathbf{h}_{\text{cg}}$ are downhill.

2° The local gradient $\mathbf{f}'(\mathbf{x}_k)$ is orthogonal to $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k$.

3° The algorithm terminates after at most $n$ steps.

**Proof:** We examine the inner product in (2.11) and insert the expression for $\mathbf{h}_{\text{cg}}$



$$\begin{aligned}\mathbf{f}'(\mathbf{x})^\top \mathbf{h}_{\text{cg}} &= -\mathbf{f}'(\mathbf{x})^\top \mathbf{f}'(\mathbf{x}) + \gamma \mathbf{f}'(\mathbf{x})^\top \mathbf{h}_{\text{prev}} \\ &= -\|\mathbf{f}'(\mathbf{x})\|_2^2 \leq 0 \quad .\end{aligned} \tag{4.8}$$

The second term in the first line is zero for any choice of $\gamma$ since exact line searches terminate when the local gradient is orthogonal to the search direction. Thus, $\mathbf{h}_{\text{cg}}$ is downhill (unless $\mathbf{x}$ is a stationary point i.e. unless $\mathbf{f}'(\mathbf{x}) = \mathbf{0}$), and we have proven 1°.

Next, the exact line searches guarantee that

$$\mathbf{h}_i^\top \mathbf{f}'(\mathbf{x}_i) = 0, \quad i = 1, \ldots, k \tag{4.9}$$

and by means of (4.5) we see that for $j < k$,

$$\begin{aligned}\mathbf{h}_j^\top \mathbf{f}'(\mathbf{x}_k) &= \mathbf{h}_j^\top \big(\mathbf{f}'(\mathbf{x}_j) + \mathbf{f}'(\mathbf{x}_k) - \mathbf{f}'(\mathbf{x}_j)\big) \\ &= 0 + \mathbf{h}_j^\top \mathbf{H}(\mathbf{x}_k - \mathbf{x}_j) \\ &= \mathbf{h}_j^\top \mathbf{H}(\mathbf{h}_k + \ldots + \mathbf{h}_{j+1}) = 0 \quad .\end{aligned}$$

Here, we have exploited that the directions $\{\mathbf{h}_i\}$ are conjugate with respect to $\mathbf{H}$, and we have proven 2°.

Finally, $\mathbf{H}$ is non-singular, and it is easy to show that this implies that a set of conjugate vectors is linearly independent. Therefore $\{\mathbf{h}_1, \ldots, \mathbf{h}_n\}$ span the entire $\mathbb{R}^n$, and $\mathbf{f}'(\mathbf{x}_n)$ must be zero. □

We remark that if $\mathbf{f}'(\mathbf{x}_k) = \mathbf{0}$ for some $k \leq n$, then the solution has been found and Algorithm 4.6 stops.

What remains is to find a clever way to determine $\gamma$. The approach used is to determine $\gamma$ in such a way that the resulting method will work well for minimizing quadratic functions. The success of the method for quadratics is then used as a justification for applying it on more general functions. This makes sense because Taylor's formula shows that smooth functions are locally well approximated by quadratics.

## 4.3. The Fletcher–Reeves Method

The following formula for $\gamma$ was the first one to be suggested:

$$\gamma = \frac{\mathbf{f}'(\mathbf{x})^\top \mathbf{f}'(\mathbf{x})}{\mathbf{f}'(\mathbf{x}_{\text{prev}})^\top \mathbf{f}'(\mathbf{x}_{\text{prev}})} \quad , \tag{4.10}$$

where $\mathbf{x}_{\text{prev}}$ is the previous iterate.

Algorithm 4.6 with this choice for $\gamma$ is called the *Fletcher–Reeves method* after the people who invented it in 1964.

**Theorem 4.2**

Apply the Fletcher–Reeves method with exact line searches to the quadratic function (4.2). If $\mathbf{f}'(\mathbf{x}_k) \neq \mathbf{0}$ for $k=1,\ldots,n$, then the search directions $\mathbf{h}_1,\ldots,\mathbf{h}_n$ are conjugate with respect to $\mathbf{H}$.

**Proof:** See Appendix B. □

According to Theorem 4.1 this implies that the Fletcher–Reeves method with exact line searches used on quadratics will terminate in at most $n$ steps.

Point 1° in Theorem 4.1 shows that a conjugate gradient method with exact linesearches produces descent directions. Al-Baali (1985) proves that this is also the case for the Fletcher–Reeves method with soft line searches satisfying certain mild conditions. We return to this result in Theorem 4.3 below.

## 4.4. The Polak–Ribière Method

An alternative formula for $\gamma$ is

$$\gamma = \frac{\big(\mathbf{f}'(\mathbf{x}) - \mathbf{f}'(\mathbf{x}_{\text{prev}})\big)^\top \mathbf{f}'(\mathbf{x})}{\mathbf{f}'(\mathbf{x}_{\text{prev}})^\top \mathbf{f}'(\mathbf{x}_{\text{prev}})} \quad , \tag{4.11}$$



Algorithm 4.6 with this choice of $\gamma$ is called the *Polak–Ribière Method*. It dates from 1971 (and again it is named after the inventors). For quadratics, (4.11) is equivalent to (4.10) (because then $\mathbf{f}'(\mathbf{x}_{\text{prev}})^\top \mathbf{f}'(\mathbf{x}) = 0$, see (B.6) in Appendix B).

For general functions, however, the two methods differ, and through the years experience has shown (4.11) to be superior to (4.10). Of course the search directions are still downhill for exact line searches combined with the Polak–Ribière Method. For soft line search there is however no result parallel to that of Al-Baali for the Fleetcher–Reeves Method. In fact M.J.D. Powell has constructed an example where the method fails to converge even with exact line search (see Nocedal (1992) p. 213). The succes of the Polak–Ribière formula is therefore not so easily explained by theory.

**Example 4.2. (Resetting).** A possibility that has been proposed, is to reset the search direction $\mathbf{h}$ to the steepest descent direction $\mathbf{h}_{\text{sd}}$ every $n$ iterations. The rationale behind this is the $n$-step quadratic termination property. If we enter a neighbourhood of the solution where $f$ behaves like a quadratic, resetting will ensure quick convergence. Another apparent advantage of resetting is that it will guarantee global convergence (by Theorem 2.1). However, practical experience has shown that the profit of resetting is doubtful.

In connection with this we remark that the Polak–Ribière method has a kind of inbuilt resetting. Should we encounter a step away from the solution with very little progress, so that $\|\mathbf{x}-\mathbf{x}_{\text{prev}}\|$ is small compared with $\|\mathbf{f}'(\mathbf{x}_{\text{prev}})\|$, then $\|\mathbf{f}'(\mathbf{x})-\mathbf{f}'(\mathbf{x}_{\text{prev}})\|$ will also be small and therefore $\gamma$ is small, and $\mathbf{h}_{\text{cg}} \simeq \mathbf{h}_{\text{sd}}$ in this situation. Also, the modification before the line search in Algorithm 4.6 may result in an occasional resetting.

## 4.5. Convergence Properties

In Theorem 4.1 we saw that the search directions $\mathbf{h}_{\text{cg}}$ of a conjugate gradient method are descent directions and thus the $\theta$ of (2.12) satisfies $\theta < \pi/2$. There is no guarantee, however, that the $\mu$ of (2.13) will stay constant, and Theorem 2.1 is therefore not directly applicable.

For many years it was thought that to guarantee convergence of a conjugate gradient method it would be necessary to use a complicated ad hoc line search, and perhaps make some other changes to the method. But in 1985 Al-Baali managed to prove global convergence using a traditional soft line search:

**Theorem 4.3**

Let the line search used in Algorithm 4.6 satisfy (2.15) and (2.16) with parameter values $\varrho < \beta < 0.5$. Then there is a $c > 0$ such that for all $k$

$$\mathbf{f}'(\mathbf{x})^{\top}\mathbf{h}_{\text{cg}} \leq -c\|\mathbf{f}'(\mathbf{x})\|_2^2$$

and

$$\lim_{k\to\infty} \|\mathbf{f}'(\mathbf{x})\|_2 = 0$$

**Proof:** See Al-Baali (1985). □

In Example 4.2 we saw that resetting will ensure global convergence for any conjugate gradient method. The importance of this result is however of more theoretical than practical value.

Let us finally remark on the rate of convergence. Crowder and Wolfe (1972) show that, for exact line searches, conjugate gradient methods have a linear convergence rate, as defined in (2.4). This should be contrasted with the superlinear convergence rate that holds for Quasi-Newton methods and the quadratic convergence rate that Newton's method possesses.



## 4.6. Other Methods and further reading

Over the years there have been proposed numerous other conjugate gradient formulae and amendments to the Fletcher–Reeves and Polak–Ribière method. We only give a short summary here, and refer the interested reader to the book by Fletcher (1987) and the paper by Nocedal (1992) for details and further information.

A possible amendment to the Polak–Ribière method is to choose $\gamma = \max(\gamma^{\text{PR}}, 0)$ where $\gamma^{\text{PR}}$ is the $\gamma$ of (4.11). With this choice of $\gamma$ it is possible to guarantee global convergence with inexact line searches. See p. 213 in Nocedal (1992) for further discussion and references.

The conjugate gradient methods belong to a class of methods sometimes referred to as conjugate direction methods. Other examples of these may be found in Fletcher (1987).

Finally we want to mention two classes of methods that have received much attention in recent years. The first class is called limited memory Quasi-Newton methods, and the second class is truncated Newton methods or inexact Newton methods. These are not conjugate direction methods, but they are also aimed at solving large problems. See pages 233–234 in Nocedal (1992) for some discussion and further references.

## 4.7. The CG Method for Linear Systems

We cannot part with conjugate gradient methods without mentioning that they can of course be used to minimize the quadratic function (4.2) itself. But by (4.3) this is equivalent to solving the positive definite linear system

$$\mathbf{H}\mathbf{x} = -\mathbf{b}\ .$$

When used in this way the exact steplength $\alpha$ may be calculated directly and no line search is needed. It is not difficult to see that

$$\alpha = \frac{-\mathbf{h}_{\rm cg}^\top \mathbf{H}(\mathbf{x}+\mathbf{b})}{\mathbf{h}_{\rm cg}^\top \mathbf{H}\mathbf{h}_{\rm cg}} \quad .$$

The Fletcher–Reeves and the Polak–Ribière formulae are equivalent in this setting, and the resulting method is called the *conjugate gradient method for linear systems.* Its study is a whole subject in itself, within the field of numerical linear algebra.

One situation where this method may be preferrable is when the system to be solved is large and sparse. Since the conjugate gradient method only needs matrix-vector multiplications it can then be much cheaper than a direct method, e.g. Gaussian elimination.

## 4.8. Implementation

To implement a conjugate gradient algorithm in a computer program, some decisions must be made. Of course we need to choose a formula for $\gamma$. Here the Polak–Ribière formula is recommended.

We also need to specify the exactness of the line search. For Newton-type methods it is usually recommended that the line search be quite soft, so for the line search in Algorithm 2.25 it is common to choose the parameter values $\varrho = 0.01$ and $\beta = 0.9$. For conjugate gradient methods experience dictates that a line search with stricter tolerances be used, say $\varrho = 0.01$ and $\beta = 0.1$. In addition we have to specify the stopping criterion. Here (2.9) is recommended. We do not have acces to $\mathbf{f}''(\mathbf{x}_k)$ and therefore cannot use (2.10). For methods with a fast convergence rate, (2.8) may be quite satisfactory, but its use for conjugate gradient methods must be discouraged because their final convergence rate is only linear.

Finally some remarks on the storage of vectors. The Fletcher–Reeves method may be implemented using three $n$-vectors of storage, `x`, `g` and `h`. If these contain $\mathbf{x}$, $\mathbf{f}'(\mathbf{x})$ and $\mathbf{h}_{\rm prev}$ at the beginning of the current iteration step, we may overwrite `h` with $\mathbf{h}_{\rm cg}$ and during the line search we overwrite `x` with $\mathbf{x}+\alpha\mathbf{h}_{\rm cg}$ and `g` with $\mathbf{f}'(\mathbf{x}+\alpha\mathbf{h}_{\rm cg})$.



Before overwriting the gradient, we find $\mathbf{f}'(\mathbf{x})^\top\mathbf{f}'(\mathbf{x})$ for use in the denominator in (4.10) on the next iteration. For the Polak–Ribière method we need acces to $\mathbf{f}'(\mathbf{x})$ and $\mathbf{f}'(\mathbf{x}_{\rm prev})$ simultaneously, and thus four vectors are required, say `x`, `g`, `gnew` and `h`.

**Example 4.3.** *Rosenbrock's function,*

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 \ + \ (1-x_1)^2 \ ,$$

is widely used for testing optimization algorithms. Figure 4.2 shows level curves for this function (and illustrates, why it is sometimes called the *"banana function"*).

The function has one minimizer $\mathbf{x}^* = [1,\ 1]^\top$ with $f(\mathbf{x}^*) = 0$, and there is a "valley" with sloping bottom following the parabola $x_2 = x_1^2$. Most optimization algorithms will try to follow this valley. Thus, we will need a considerable amount of iteration steps if we take $\mathbf{x}_0$ in the 2nd quadrant.

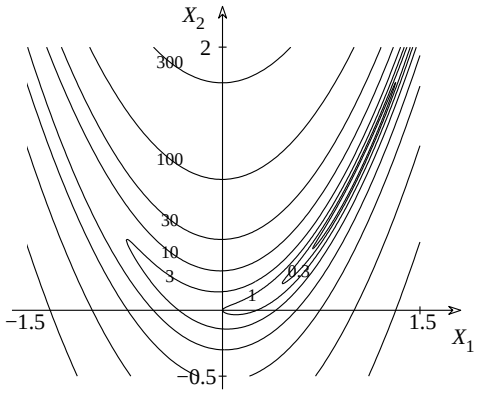



Figure 4.2: *Contours of Rosenbrock's function*

Below we give the number of iteration steps and evaluations of $f(\mathbf{x})$ and $\mathbf{f}'(\mathbf{x})$ when applying Algorithm 4.6 on this function. In all cases

we use the starting point $\mathbf{x}_0 = [-1.2,\ 1]^\mathsf{T}$, and stopping criteria given by $\varepsilon_1 = 10^{-8}$, $\varepsilon_2 = 10^{-12}$ in (4.7). In case of exact line search we use $\tau = 10^{-6}$, $\varepsilon = 10^{-6}$ in (2.27), while we take $\beta = 10^{-1}$, $\varrho = 10^{-2}$ in Algorithm 2.25 for soft line search.

| Method | Line search | # it. steps | # fct. evals |
|---|---|---|---|
| Fletcher–Reeves | exact | 118 | 1429 |
| Fletcher–Reeves | soft | 249 | 628 |
| Polak–Ribière | exact | 24 | 266 |
| Polak–Ribière | soft | 45 | 130 |

Thus, in this case the Polak–Ribière method with soft line search performs best. Below we give the iterates (cf. Figure 4.2) and the values of $f(\mathbf{x}_k)$ and $\|\mathbf{f}'(\mathbf{x}_k)\|_\infty$; note the logarithmic ordinate axis.

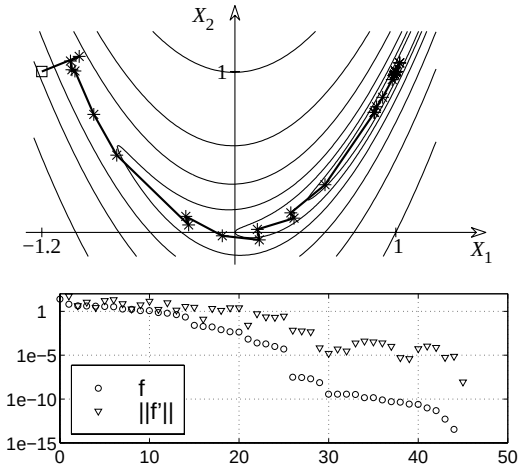



Figure 4.3: *Polak–Ribière method with soft line search applied to Rosenbrock's function.*
*Top: iterates* $\mathbf{x}_k$. *Bottom:* $f(\mathbf{x}_k)$ *and* $\|\mathbf{f}'(\mathbf{x}_k)\|_\infty$.

# 5. Newton-Type Methods

In this chapter we consider a class of methods for unconstrained optimization which are based on Newton's method. This class is called Quasi-Newton methods. In order to explain these methods we first describe Newton's method for unconstrained optimization in detail. Newton's method leads to another kind of methods known as Damped Newton Methods, which will also be presented.

Finally we get to the Quasi-Newton methods. This class includes some of the best methods on the market for solving the unconstrained optimization problem.

## 5.1. Newton's Method

Newton's method forms the basis of all Quasi-Newton methods. It is widely used for solving systems of non-linear equations, and until recently it was also widely used for solving unconstrained optimization problems. As it will appear, the two problems are closely related.

**Example 5.1.** In Example 1.2 we saw the method of alternating directions fail to find the minimizer of a simple quadratic in two dimensions and in Example 3.1 we saw the steepest descent method fail on the same quadratic. In Chapter 4 we saw that the conjugate gradient methods finds the minimizer of a quadratic in $n$ steps ($n$ being the dimension of the space), in two steps in Example 4.1.

Newton's method can find the minimizer of a quadratic in $n$-dimensional space in one step. This follows from equation (5.2) below.

Figure 5.1 gives the contours of our 2-dimensional quadratic together with (an arbitrary) $\mathbf{x}_0$. $\mathbf{x}_1$ and the minimizer $\mathbf{x}^*$, marked by $*$.

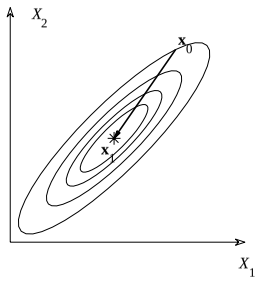



Figure 5.1: *Newton's method finds the minimizer of a quadratic in the very first step*

In order to derive *Newton's method* in the version used in optimization, we shall once again consider the truncated Taylor expansion of the cost function at the current iterate $\mathbf{x}$ :

$$f(\mathbf{x}+\mathbf{h}) \simeq q(\mathbf{h}) \ , \tag{5.1a}$$

where $q(\mathbf{h})$ is the quadratic model of $f$ in the vicinity of $\mathbf{x}$,

$$q(\mathbf{h}) = f(\mathbf{x}) + \mathbf{h}^\top \mathbf{f}'(\mathbf{x}) + \tfrac{1}{2}\mathbf{h}^\top \mathbf{f}''(\mathbf{x})\mathbf{h} \ . \tag{5.1b}$$

The idea now is to minimize the model $q$ at the current iterate. If $\mathbf{f}''(\mathbf{x})$ is positive definite, then $q$ has a unique minimizer at a point where the gradient of $q$ equals zero, i.e. where

$$\mathbf{f}'(\mathbf{x}) + \mathbf{f}''(\mathbf{x})\mathbf{h} \ = \ \mathbf{0} \ . \tag{5.2}$$

Hence, in Newton's method the new iteration step is obtained as the solution to the system (5.2) as shown in the following algorithm.

**Algorithm 5.3. Newton's Method**

```
begin
  x := x_0;                                   {Initialisation}
  repeat
    Solve  f''(x)h_N = -f'(x)                 {find step}
    x := x + h_N                              {... and next iterate}
  until stopping criteria satisfied
end
```



Newton's method is well defined as long as $\mathbf{f}''(\mathbf{x})$ remains non-singular. Also, (5.2) shows that the step is downhill if the Hessian is positive definite:

$$\mathbf{h}_{\mathrm{N}}^\top \mathbf{f}''(\mathbf{x})\mathbf{h}_{\mathrm{N}} > 0 \implies \mathbf{h}_{\mathrm{N}}^\top \mathbf{f}'(\mathbf{x}) < 0 \tag{5.4}$$

proving that $\mathbf{h}_{\mathrm{N}}$ is downhill, see Definition (2.11). Further, if $\mathbf{f}''(\mathbf{x})$ stays positive definite in all the steps and if the starting point is sufficiently close to a minimizer, then the method usually converges rapidly towards such a solution. More precisely the following theorem holds:

**Theorem 5.1**

If one of the iterates, $\mathbf{x}$, is sufficiently close to a local minimizer $\mathbf{x}^*$ and $\mathbf{f}''(\mathbf{x}^*)$ is positive definite, then Newton's method is well defined for all the following steps, and it converges quadratically towards $\mathbf{x}^*$.

**Proof:** See e.g. Fletcher (1987). □

**Example 5.2.** We shall use Newton's method to find the minimizer of the following function

$$\begin{aligned} f(\mathbf{x}) = \ & 0.5 * x_1^2 * (x_1^2/6 + 1) \\ & + x_2 * \mathrm{Arctan}(x_2) - 0.5 * \ln(x_2^2 + 1) \ . \end{aligned} \tag{5.5}$$

We need the derivatives of first and second order for this function:

$$\mathbf{f}'(\mathbf{x}) = \begin{bmatrix} x_1^3/3 + x_1 \\ \mathrm{Arctan}(x_2) \end{bmatrix}, \qquad \mathbf{f}''(\mathbf{x}) = \begin{bmatrix} x_1^2 + 1 & 0 \\ 0 & 1/(1+x_2^2) \end{bmatrix} .$$

We can see in Figure 5.2 that in a region around the minimizer the function looks very well-behaved and extremely simple to minimize.

Table 5.1 gives results of the iterations with the starting point $\mathbf{x}_0^\top = [1,\ 0.7]$. According to Theorem 5.1 we expect quadratic convergence.

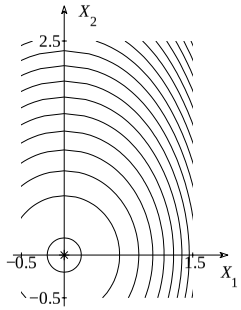



Figure 5.2: *Contours of the function (5.5). The level curves are symmetric across both axes*

If the factor $c_2$ in (2.5) is of the order of magnitude 1, then the column of $\mathbf{x}_k^\top$ would show the number of correct digits doubled in each iteration step, and the $f$-values and step lengths would be squared in each iteration step. The convergence is faster than this; actually for any starting point $\mathbf{x}_0^\top = [u,\ v]$ with $|v| < 1$ we will get *cubic convergence*; see the next example.

| $k$ | $\mathbf{x}_k^\top$ | $f$ | $\|\mathbf{f}'\|$ | $\|\mathbf{h}_k\|$ |
|---|---|---|---|---|
| 0 | [1.00000000000000, 0.70000000000000] | 8.11e-01 | 1.47e+00 | |
| 1 | [0.33333333333333, -0.20998168693992] | 7.85e-02 | 4.03e-01 | 1.13e+00 |
| 2 | [0.02222222222222, 0.00611895804438] | 2.66e-04 | 2.31e-02 | 3.79e-01 |
| 3 | [0.00000731234690, -0.00000015273477] | 2.67e-11 | 7.31e-06 | 2.30e-02 |
| 4 | [0.00000000000000, 0.00000000000000] | 3.40e-32 | 2.61e-16 | 7.31e-06 |
| 5 | [0.00000000000000, 0.00000000000000] | 0.00e+00 | 0.00e+00 | 2.61e-16 |

Table 5.1: *Newton's method on (5.5).* $\mathbf{x}_0^\top = [1,\ 0.7]$

Until now, everything which has been said about Newton's method seems very promising: It is very simple and if the conditions of Theorem 5.1 are satisfied, then the rate of convergence is excellent. Nevertheless, due to a series of drawbacks the basic version of the method is not suitable for a general purpose optimization algorithm.

The first and by far the most severe drawback is the methods lack of global convergence.



**Example 5.3.** With the starting point $\mathbf{x}_0^\top = [1,\ 2]$ the Newton method behaves very badly:

| $k$ | $\mathbf{x}_k^\top$ | $f$ | $\|\mathbf{f}'\|$ | $\|\mathbf{h}_k\|$ |
|---|---|---|---|---|
| 0 | [1.00000000000000, 2.00000000000000] | 1.99e+00 | 1.73e+00 | |
| 1 | [0.33333333333333, -3.53574358897045] | 3.33e+00 | 1.34e+00 | 5.58e+00 |
| 2 | [0.02222222222222, 13.95095908692750] | 1.83e+01 | 1.50e+00 | 1.75e+01 |
| 3 | [0.00000731234690, -2.7934406653e+02] | 4.32e+02 | 1.57e+00 | 2.93e+02 |
| 4 | [0.00000000000000, 1.2201699892e+05] | 1.92e+05 | 1.57e+00 | 1.22e+05 |
| 5 | [0.00000000000000, -2.3386004198e+10] | 3.67e+10 | 1.57e+00 | 2.34e+10 |

Table 5.2: *Newton's method on (5.5).* $\mathbf{x}_0^\top = [1,\ 2]$

Clearly, the sequence of iterates moves rapidly away from the solution (the first component converges, whereas the second increases in size with alternating sign) even though $\mathbf{f}''(\mathbf{x})$ is positive definite for any $\mathbf{x} \in \mathbb{R}^2$.

The reader is encouraged to investigate what happens in detail. Hint: The Taylor expansion for Arctan(0+$h$) is

$$\text{Arctan}(0+h) = \begin{cases} h - \frac{1}{3}h^3 + \frac{1}{5}h^5 - \frac{1}{7}h^7 + \cdots & \text{for } |h| < 1 \\ \text{sign}(h)\left(\frac{\pi}{2} - \frac{1}{h} + \frac{1}{3h^3} - \frac{1}{5h^5} + \cdots\right) & \text{for } |h| > 1\,. \end{cases}$$

The next point to discuss is that $\mathbf{f}''(\mathbf{x})$ may not be positive definite when $\mathbf{x}$ is far from the solution. In this case the sequence may be heading towards a saddle point or a maximizer since the iteration is identical to the one used for solving the non-linear system of equations $\mathbf{f}'(\mathbf{x}) = \mathbf{0}$. Any stationary point of $f$ is a solution to this system. Also, $\mathbf{f}''(\mathbf{x})$ may be ill-conditioned or singular so that the linear system (5.2) cannot be solved without considerable errors in $\mathbf{h}_{\text{N}}$. Such ill-conditioning may be detected by a well designed matrix factorization (e.g. a Cholesky factorization as described in Appendix A), but it still leaves the question of what to do in case ill-conditioning occurs.

The final major drawback is of a more practical nature but basically just as severe as the ones already discussed. Algorithm 5.3

requires the analytic second order derivatives. These may be difficult to determine even though they are known to exist. Further, in case they can be obtained, users tend to make erroneous implementations of the derivatives (and later blame a consequential malfunction on the optimization algorithm). Also, in large scale problems the calculation of the Hessian may be costly since $\frac{1}{2}n(n+1)$ function evaluations are needed.

Below, we summarize the advantages and disadvantages of Newton's method discussed above. They are the key to the development of more useful algorithms, since they point out properties to be retained and areas where improvements and modifications are required.

**Advantages and disadvantages of Newton's method for unconstrained optimization problems**

**Advantages**

1° Quadratically convergent from a good starting point if $\mathbf{f}''(\mathbf{x}^*)$ is positive definite.

2° Simple and easy to implement.

**Disadvantages**

1° Not globally convergent for many problems.

2° May converge towards a maximum or saddle point of $f$.

3° The system of linear equations to be solved in each iteration may be ill-conditioned or singular.

4° Requires analytic second order derivatives of $f$.

Table 5.3: *Pros and Cons of Newton's Method*

## 5.2. Damped Newton Method

Despite the fact that disadvantage no. 4 in Table 5.3 often makes it impossible to use any of the modified versions of Newton's method, we shall still discuss these, because some important ideas have been



introduced when they were developed. Further, in case second order derivatives are obtainable, modified Newton methods may be used succesfully. Hence, for the methods discussed in this subsection it is still assumed, that second order analytic derivatives of $f$ are available.

The more efficient modified Newton methods are constructed as either explicit or implicit hybrids between the original Newton method and the method of steepest descent. The idea is that the Algorithm in some way should take advantage of the safe, global convergence properties of the steepest descent method whenever Newton's method gets into trouble. On the other hand the quadratic convergence of Newton's method should be obtained when the iterates get close enough to $\mathbf{x}^*$, provided that the Hessian is positive definite.

The first modification which comes to mind is a Newton method with line search in which the Newton step is used as a search direction i.e. $\mathbf{h}_{\mathrm{N}} = -[\mathbf{f}''(\mathbf{x})]^{-1}\mathbf{f}'(\mathbf{x})$. Such a method is obtained if the step $\mathbf{x} := \mathbf{x}+\mathbf{h}_{\mathrm{N}}$ in 5.3 is substituted by

$$\alpha := \mathrm{line\_search}(\mathbf{x}, \mathbf{h}_{\mathrm{N}}); \quad \mathbf{x} := \mathbf{x} + \alpha\mathbf{h}_{\mathrm{N}} \tag{5.6}$$

This will work fine as long as $\mathbf{f}''(\mathbf{x})$ is positive definite since in this case $\mathbf{h}_{\mathrm{N}}$ is a descent direction, cf. (5.4).

The main difficulty thus arises when $\mathbf{f}''(\mathbf{x})$ is not positive definite. The Newton step can still be computed if $\mathbf{f}''(\mathbf{x})$ is non-singular, and one may search along $\pm\mathbf{h}_{\mathrm{N}}$ where the sign is chosen in each iteration to ensure a descent direction. However, this rather primitive approach is questionable since the quadratic model $q(\mathbf{h})$ will not even possess a unique minimum.

A much more appealing modification is a hybrid method where we keep the line search and use a steepest descent direction in case the Hessian is not positive definite. (This is the so-called Goldstein and Price (1960) modification). In order to ensure global convergence towards a stationary point, one must also demand that possible Newton directions shall satisfy the angle test (2.13) in order for the method to

be an absolute descent method. Unfortunately, according to Fletcher (1987), such a method frequently behaves like the steepest descent method itself due to the fact that the second order information is ignored in many of the steps.

The last class of modifications of the original Newton method to be considered here is often refered to as the Damped Newton methods. These are also considered to be the most successful in general. In order to derive the framework of these methods, Newton's method and a reformulated version of the steepest descent method are shown together here. ($\mathbf{I}$ is the identity matrix).

| **Steepest Descent** | **Newton's method** |
|---|---|
| Solve $\mathbf{I}\,\mathbf{h}_{\mathrm{sd}} = -\mathbf{f}'(\mathbf{x})$ | Solve $\mathbf{f}''(\mathbf{x})\mathbf{h}_{\mathrm{N}} = -\mathbf{f}'(\mathbf{x})$ |
| $\alpha := \mathrm{line\_search}(\mathbf{x}, \mathbf{h}_{\mathrm{sd}})$ | |
| $\mathbf{x} := \mathbf{x} + \alpha\mathbf{h}_{\mathrm{sd}}$ | $\mathbf{x} := \mathbf{x} + \mathbf{h}_{\mathrm{N}}$ |

Table 5.4: *Steepest Descent and Newton's method*

The approach in a *Damped Newton method* is to combine the two methods by adding a multiple of the identity matrix to $\mathbf{f}''(\mathbf{x})$. Hence, the framework for this type of method is

**Damped Newton step**

Solve $(\mathbf{f}''(\mathbf{x}) + \mu\mathbf{I})\,\mathbf{h}_{\mathrm{dN}} = -\mathbf{f}'(\mathbf{x}) \qquad (\mu \geq 0)$

Adjust $\mu$

If $\mathbf{x} + \mathbf{h}_{\mathrm{dN}}$ is acceptable, then $\mathbf{x} := \mathbf{x} + \mathbf{h}_{\mathrm{dN}}$ (5.7)

As it is easily seen, this type of method is a compromise between the two underlying methods. If $\mu$ is large then $\mathbf{h}_{\mathrm{dN}}$ will be very close to the steepest descent direction, whereas a small $\mu$ yields an $\mathbf{h}_{\mathrm{dN}}$ which is close to the Newton direction $\mathbf{h}_{\mathrm{N}}$. Since second order information is not neglected, methods of the this type are normally more effective than the one by Goldstein and Price.



In (5.7) there is no line search but a new type of parameter has appeared. We must decide how $\mu$ should be chosen. Furthermore, we must consider if it is wise to leave out the line search. Damped Newton type methods with line search have been used (see e.g. Luenberger (1974)). However, as we shall see below, present techniques for choosing $\mu$ makes the line search obsolete (this is also presented in Luenberger (1974)).

There are several schemes for dynamical updating of $\mu$. In *Levenberg–Marquardt* type methods $\mu$ is updated in each iteration step. Given the present value of the parameter, the Cholesky factorization of $\mathbf{f}''(\mathbf{x})+\mu\mathbf{I}$ is employed to check for positive definiteness, and $\mu$ is increased if the matrix is not significantly positive definite. Otherwise, the solution $\mathbf{h}_{\mathrm{dN}}$ is easily obtained via the factorization. Note, that increasing $\mu$ in the case where $\mathbf{f}''(\mathbf{x})+\mu\mathbf{I}$ is not positive definite corresponds to changing the quadratic model so that it has a unique minimizer.

With the procedure above the direction found is sure to be downhill, but this is not enough to ensure global convergence. We must also include measures that ensure that the length of the step is appropriate, so that the method is descending. (Consider what would happen if this was the only modification and such a method were used to minimize the tricky function (5.5)). Also, the procedure only provides mechanisms to increase $\mu$. There is no way to reduce it and thereby take advantage of the rapid convergence of the Newton method.

As in a trust region method we can investigate the value of the cost function at the trial point, i.e. $f(\mathbf{x}+\mathbf{h}_{\mathrm{dN}})$. If it is sufficiently below $f(\mathbf{x})$, then the point $\mathbf{x}+\mathbf{h}_{\mathrm{dN}}$ is chosen as the next iterate. Otherwise, $\mathbf{x}$ is still the current iterate, and $\mu$ is increased. It is not sufficient to check whether $f(\mathbf{x} + \mathbf{h}_{\mathrm{dN}}) < f(\mathbf{x})$. In order to prove convergence for the whole procedure one needs to test whether the actual decrease in $f$-value is larger than some small portion of the decrease predicted by the quadratic model (5.1), i.e. if

$$r \equiv \frac{f(\mathbf{x}) - f(\mathbf{x}+\mathbf{h})}{q(\mathbf{0}) - q(\mathbf{h})} > \delta \,, \tag{5.8}$$

where $\delta$ is a small positive number (typically $\delta \simeq 10^{-3}$).

We recognize $r$ as the *gain factor*, (2.22). It is also used to monitor $\mu$: If $r$ is close to one, then one may expect the model to be a good approximation of $f$ in the neighbourhood of $\mathbf{x}$. Thus the influence of Newton's method should be increased by decreasing $\mu$. If, on the other hand, the actual decrease of $f$ is much smaller than expected, then $\mu$ must be increased in order to adjust the method more towards steepest descent. It is important to note that in this case the length of $\mathbf{h}_{\mathrm{dN}}$ is reduced, since for $\mu$ large $\mathbf{h}_{\mathrm{dN}} \simeq \frac{-1}{\mu}\mathbf{f}'(\mathbf{x})$. Thus $\mu$ acts as a kind of step size regulator, besides its control over the step direction, and we have a further analogy to trust region methods. The reader is referred to Fletcher (1987) or Moré and Sorenson (1982) for a treatment of this subject.

We could use an updating strategy similar to the one employed in Algorithm 2.23,

$$\begin{aligned} &\texttt{if } r > 0.75 \\ &\quad \mu := \mu/3 \\ &\texttt{if } r < 0.25 \\ &\quad \mu := \mu * 2 \end{aligned} \tag{5.9}$$

However, the discontinuous changes in $\mu$ when $r$ is close to 0.25 or 0.75 can cause a "flutter" that slows down convergence. Therefore, we recommend to use the equally simple strategy given by

$$\begin{aligned} &\texttt{if } \ r > 0 \\ &\quad \mu := \mu * \max\{\tfrac{1}{3}, 1 - (2r-1)^3\} \\ &\texttt{else} \\ &\quad \mu := \mu * 2 \end{aligned} \tag{5.10}$$

The two strategies are illustrated below and are further discussed in Nielsen (1999) and Section 3.2 of Madsen et *al.* (1999).



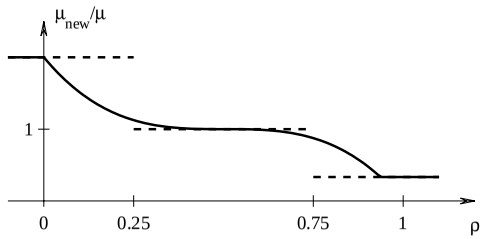



Figure 5.3: *Updating of $\mu$ by (5.9) (dasheded line) and by (5.10) (full line).*

The method is summarized in

**Algorithm 5.11. Damped Newton Method**

```
begin                                   {Levenberg–Marquardt type damped Newton}
  x := x_0;  μ := μ_0;  found := false;  k := 0;          {Initialisation}
  repeat
    while f''(x)+μI not pos. def.                          {using ...}
      μ := 2μ
    Solve (f''(x)+μI) h_dN = -f'(x)                        {... Cholesky}
    Compute gain factor r by (5.8)
    if r > δ                                               {f decreases}
      x := x + h_dN                                        {new iterate}
      μ := μ * max{1/3, 1 - (2r-1)^3}                      {... and μ}
    else
      μ := μ * 2                                           {increase μ but keep x}
    k := k+1;  Update found                                {see (5.12)}
  until found or k > k_max
end                                     {of LM type damped Newton}
```

Similar to (4.7) we can use the stopping criteria

$$\|\mathbf{f}'(\mathbf{x})\|_\infty \le \varepsilon_1 \quad \texttt{or} \quad \|\mathbf{h}_{\mathrm{dN}}\|_2 \le \varepsilon_2(\varepsilon_2 + \|\mathbf{x}\|_2) \,. \tag{5.12}$$

The simplicity of the original Newton method has disappeared in the attempt to obtain global convergence, but this type of method does perform well in general.

**Example 5.4.** Table 5.5 illustrates the performance of Algorithm 5.11 when applied to the tricky function (5.5). We use $\mu_0 = 1$ and $\varepsilon_1 = 10^{-8}$, $\varepsilon_2 = 10^{-12}$ in (5.12).

| $k$ | $\mathbf{x}_k^\top$ | $f$ | $\|\mathbf{f}'\|_\infty$ | $r$ | $\mu$ |
|---|---|---|---|---|---|
| 0 | [ 1.0000000000, 2.0000000000] | 1.99e+00 | 1.33e+00 | 0.999 | 1.00e+00 |
| 1 | [ 0.5555555556, 1.0773760685] | 6.63e-01 | 8.23e-01 | 0.872 | 3.33e-01 |
| 2 | [ 0.1824004456, 0.0441028668] | 1.77e-02 | 1.84e-01 | 1.010 | 1.96e-01 |
| 3 | [ 0.0323940533, 0.0071966616] | 5.51e-04 | 3.24e-02 | 1.000 | 6.54e-02 |
| 4 | [ 0.0020074933, 0.0004414865] | 2.11e-06 | 2.01e-03 | 1.000 | 2.18e-02 |
| 5 | [ 0.0000428275, 0.0000094174] | 9.61e-10 | 4.28e-05 | 1.000 | 7.27e-03 |
| 6 | [ 0.0000003089, 0.0000000679] | 5.00e-14 | 3.09e-07 | 1.000 | 2.42e-03 |
| 7 | [ 0.0000000007, 0.0000000002] | 3.05e-19 | 7.46e-10 | | |

Table 5.5: *Algorithm 5.11 applied to (5.5).* $\mathbf{x}_0^\top = [1,\ 2]$, $\mu_0 = 1$

The solution is found without problems, and the columns with $f$ and $\|\mathbf{f}'\|$ show superlinear convergence, as defined in (2.6).

**Example 5.5.** We have used Algorithm 5.11 on Rosenbrock's function from Example 4.3. We use the same starting point, $\mathbf{x}_0 = [-1.2,\ 1]^\top$, and with $\mu_0 = 1$, $\varepsilon_1 = 10^{-10}$, $\varepsilon_2 = 10^{-12}$ we found the solution after 29 iteration steps. The performance is illustrated below

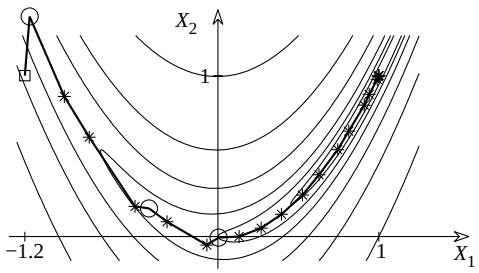



Figure 5.4a: *Damped Newton Method on Rosenbrock's function. Iterates*



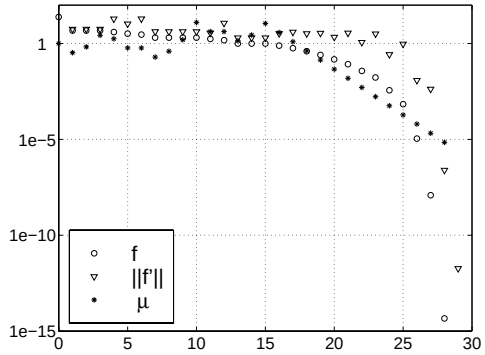



Figure 5.4b: $f(\mathbf{x}_k)$, $\|\mathbf{f}'(\mathbf{x}_k)\|_\infty$ *and* $\mu$.

The three circles in Figure 5.4a indicates points, where the iterations stalls, i.e. the current $\mathbf{x}$ is not changed, but $\mu$ is updated. After passing the bottom of the parabola, the damping parameter $\mu$ is decreased in each step. As in the previous example we achieve superlinear final convergence.

## 5.3. Quasi–Newton Methods

The modifications discussed in the previous section make it possible to overcome the first three of the main disadvantages of Newton's method shown in Table 5.3: The damped Newton method is globally convergent, ill-conditioning may be avoided, and minima are rapidly located. However, no means of overcoming the fourth disadvantage has been considered: The user must still supply formulae and implementations of the second derivatives of the cost function.

In *Quasi–Newton methods* (from latin, quasi: nearly) the idea is to use matrices which approximate the Hessian matrix or its inverse, instead of the Hessian matrix or its inverse in Newton's equation (5.2). The matrices are normally named

$$\mathbf{B} \simeq \mathbf{f}''(\mathbf{x}) \qquad \text{and} \qquad \mathbf{D} \simeq \mathbf{f}''(\mathbf{x})^{-1} \ . \tag{5.13}$$

The matrices can be produced in many different ways ranging from very simple techniques to highly advanced schemes, where the approximation is built up and adjusted dynamically on the basis of information about the first derivatives, obtained during the iteration. These advanced Quasi–Newton methods, developed in the period from 1959 and up to the present days, are some of the most powerful methods for solving unconstrained optimization problems.

Possibly the simplest and most straight-forward Quasi–Newton method is obtained if the elements of the Hessian matrix are approximated by finite differences: In each coordinate direction, $\mathbf{e}_i$ ($i$=1,...,$n$), a small increment $\delta_i$ is added to the corresponding element of $\mathbf{x}$ and the gradient in this point is calculated. The $i^{\text{th}}$ column of a matrix $\mathbf{B}$ is calculated as the *difference approximation* $\big(\mathbf{f}'(\mathbf{x}+\delta_i\mathbf{e}_i) - \mathbf{f}'(\mathbf{x})\big)/\delta_i$. After this, the symmetric matrix $\mathbf{B} := \frac{1}{2}(\mathbf{B}+\mathbf{B}^\top)$ is formed.

If the $\{\delta_i\}$ are chosen appropriately, this is a good approximation to $\mathbf{f}''(\mathbf{x})$ and may be used in a damped Newton method. However, the alert reader will notice that this procedure requires $n$ extra evaluations of the gradient in each iteration – an affair that may be very costly. Further, there is no guaranty that $\mathbf{B}$ is positive (semi-)definite.

In the advanced Quasi–Newton methods these extra gradient evaluations are avoided. Instead we use updating formulae where the $\mathbf{B}$ or $\mathbf{D}$ matrices (see 5.13) are determined from information about the iterates, $\mathbf{x}_1, \mathbf{x}_2, \ldots$ and the gradients of the cost function, $\mathbf{f}'(\mathbf{x}_1), \mathbf{f}'(\mathbf{x}_2), \ldots$ gathered during the iteration steps. Thus, in each iteration step the $\mathbf{B}$ (or $\mathbf{D}$) matrix is changed so that it finally converges towards $\mathbf{f}''(\mathbf{x}^*)$ (or respectively $\mathbf{f}''(\mathbf{x}^*)^{-1}$), $\mathbf{x}^*$ being the minimizer.

## 5.4. Quasi–Newton with Updating Formulae

We begin this subsection with a short discussion on why approximations to the inverse Hessian are preferred rather than approximations to the Hessian itself: First, the computational labor in the updating



is the same no matter which of the matrices we update. Second, if we have an approximate inverse, then the search direction is found simply by multiplying the approximation with the negative gradient of $f$. This is an $O(n^2)$ process whereas the solution of the linear system with $\mathbf{B}$ as coefficient matrix is an $O(n^3)$ process.

A third possibility is to use approximations to the Cholesky factor of the Hessian matrix, determined at the start of the iteration and updated in the iteration. Using these, we can find the solution of the system (5.2) in $O(n^2)$ operations. This technique is beyond the scope of the present notes, but the details can be found in Dennis and Schnabel (1984). Further, we remark that early experiments with updating formulae indicated that the updating of an approximation to the inverse Hessian might become unstable. According to Fletcher (1987), recent research indicates that this needs not be the case.

A classical Quasi–Newton method with updating always includes a line search. Alternatively, updating formulae have been used in trust region methods. Basically, these two different approaches (line search or trust region) define two classes of methods. In this section we shall confine ourselves to the line search approach.

With these comments the framework may be presented:

**Framework 5.14 for iteration step**
**Quasi–Newton with Updating and Line Search**

$\mathbf{B}$ (or $\mathbf{D}$) is the current approximation to $\mathbf{f}''(\mathbf{x})$ (or $\mathbf{f}''(\mathbf{x})^{-1}$)
Solve $\mathbf{B}\mathbf{h}_{\text{qN}} = -\mathbf{f}'(\mathbf{x})$ (or compute $\mathbf{h}_{\text{qN}} := -\mathbf{D}\,\mathbf{f}'(\mathbf{x})$)
Line search along $\mathbf{h}_{\text{qN}}$ giving $\mathbf{h}_{\text{qN}} := \alpha\mathbf{h}_{\text{qN}}$; $\mathbf{x}_{\text{new}} = \mathbf{x} + \mathbf{h}_{\text{qN}}$
Update $\mathbf{B}$ to obtain $\mathbf{B}_{\text{new}}$ (or $\mathbf{D}$ to $\mathbf{D}_{\text{new}}$)

In what follows the requirements to the updating and the techniques needed shall be presented.

## 5.5. The Quasi–Newton Condition

The first and most important requirement which an updating formula must satisfy, is the socalled *Quasi–Newton condition,* which may be derived in several ways. The condition is also referred to as the *Sequant condition,* because it is closely related to the secant method for non-linear equations with one unknown.

Let $\mathbf{x}$ and $\mathbf{D}$ be the current iterate and approximation to $\mathbf{f}''(\mathbf{x})^{-1}$. Given these, the first parts of the iteration step in the framework 5.14 can be performed yielding $\mathbf{h}_{\mathrm{qN}}$ and hence $\mathbf{x}_{\mathrm{new}}$. The objective is to calculate $\mathbf{D}_{\mathrm{new}}$ by a correction of $\mathbf{D}$. The correction must contain some information about the second derivatives. Clearly, this information is only approximate. It is based on the gradients of $f$ at the two points. Now, consider the Taylor expansion of $\mathbf{f}'$ around $\mathbf{x}+\mathbf{h}_{\mathrm{qN}}$ :

$$\mathbf{f}'(\mathbf{x}) = \mathbf{f}'(\mathbf{x}+\mathbf{h}_{\mathrm{qN}}) - \mathbf{f}''(\mathbf{x}+\mathbf{h}_{\mathrm{qN}})\mathbf{h}_{\mathrm{qN}} + \cdots \quad . \tag{5.15}$$

If $f$ were a quadratic function, then the higher order terms would vanish, and the equation would be exact. Since $f$ is well approximated by a quadratic near $\mathbf{x}_{\mathrm{new}} = \mathbf{x}+\mathbf{h}_{\mathrm{qN}}$, and since the higher order terms are difficult to handle, they are neglected. With these comments and the notation

$$\mathbf{y} = \mathbf{f}'(\mathbf{x}_{\mathrm{new}}) - \mathbf{f}'(\mathbf{x}) \ , \tag{5.16}$$

equation (5.15) leads to the relation, similar to (4.5),

$$\mathbf{y} = \mathbf{f}''(\mathbf{x}_{\mathrm{new}})\mathbf{h}_{\mathrm{qN}} \ .$$

Therefore, we require that $\mathbf{D}_{\mathrm{new}}$ should satisfy

$$\mathbf{D}_{\mathrm{new}}\mathbf{y} = \mathbf{h}_{\mathrm{qN}} \ . \tag{5.17a}$$

This is the *Quasi–Newton condition.* The same arguments lead to the alternative formulation of the Quasi–Newton condition,

$$\mathbf{B}_{\mathrm{new}}\mathbf{h}_{\mathrm{qN}} = \mathbf{y} \ . \tag{5.17b}$$



The Quasi–Newton condition only supplies $n$ conditions on the matrix $\mathbf{D}_{\mathrm{new}}$ (or $\mathbf{B}_{\mathrm{new}}$) but it has $n^2$ elements. Therefore additional conditions are needed to get a well defined method.

In the Quasi–Newton methods that we describe, the $\mathbf{D}$ (or $\mathbf{B}$) matrix is updated in each iteration step. We produce $\mathbf{D}_{\mathrm{new}}$ (or $\mathbf{B}_{\mathrm{new}}$) by adding a correction term to the present $\mathbf{D}$ (or $\mathbf{B}$). An important requirement to the updating is that it must be simple and fast to perform and yet effective. This can be obtained with a recursive relation between successive approximations,

$$\mathbf{D}_{\mathrm{new}} = \mathbf{D} + \mathbf{W} \ ,$$

where $\mathbf{W}$ is a correction matrix. In nearly all methods used in practice, $\mathbf{W}$ is a *rank-one matrix*

$$\mathbf{D}_{\mathrm{new}} = \mathbf{D} + \mathbf{a}\mathbf{b}^{\top}$$

or a *rank-two matrix*

$$\mathbf{D}_{\mathrm{new}} = \mathbf{D} + \mathbf{a}\mathbf{b}^{\top} + \mathbf{u}\mathbf{v}^{\top} \ ,$$

where $\mathbf{a}, \mathbf{b}, \mathbf{u}, \mathbf{v} \in \mathbb{R}^n$. Hence $\mathbf{W}$ is an *outer product* of two vectors or a sum of two such products. Often $\mathbf{a}$ equals $\mathbf{b}$, and $\mathbf{u} = \mathbf{v}$; this is a simple way of ensuring that $\mathbf{W}$ is symmetric.

## 5.6. Broyden's Rank-One Formula

Tradition calls for a presentation of the simplest of all updating formulas which was initially described by Broyden (1965). It was not the first updating formula but we present it here to illustrate some of the ideas and techniques used to establish updating formulae.

First, consider rank-one updating of the matrix $\mathbf{B}$ :

$$\mathbf{B}_{\mathrm{new}} = \mathbf{B} + \mathbf{a}\mathbf{b}^{\top} \ .$$

The vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ are chosen so that they satisfy the Quasi–

Newton condition (5.17b),

$$\left(\mathbf{B} + \mathbf{a}\mathbf{b}^\top\right)\mathbf{h}_{\mathrm{qN}} = \mathbf{y} \tag{5.18a}$$

and – in an attempt to keep information already in $\mathbf{B}$ – Broyden demands that for all $\mathbf{v}$ orthogonal to $\mathbf{h}_{\mathrm{qN}}$ we get $\mathbf{B}_{\mathrm{new}}\mathbf{v} = \mathbf{B}\mathbf{v}$, i.e.

$$\left(\mathbf{B} + \mathbf{a}\mathbf{b}^\top\right)\mathbf{v} = \mathbf{B}\mathbf{v} \quad \text{for all } \mathbf{v} \mid \mathbf{v}^\top\mathbf{h}_{\mathrm{qN}} = 0\ . \tag{5.18b}$$

These conditions are satisfied if we take $\mathbf{b} = \mathbf{h}_{\mathrm{qN}}$ and the vector $\mathbf{a}$ determined by

$$(\mathbf{h}_{\mathrm{qN}}^\top\mathbf{h}_{\mathrm{qN}})\mathbf{a} = \mathbf{y} - \mathbf{B}\mathbf{h}_{\mathrm{qN}}\ .$$

This results in *Broyden's rank-one formula* for updating the approximation to the Hessian:

$$\mathbf{B}_{\mathrm{new}} = \mathbf{B} + \frac{1}{\mathbf{h}_{\mathrm{qN}}^\top\mathbf{h}_{\mathrm{qN}}}(\mathbf{y} - \mathbf{B}\mathbf{h}_{\mathrm{qN}})\mathbf{h}_{\mathrm{qN}}^\top\ . \tag{5.19}$$

A formula for updating an approximation to the inverse Hessian may be derived in the same way and we obtain

$$\mathbf{D}_{\mathrm{new}} = \mathbf{D} + \frac{1}{\mathbf{y}^\top\mathbf{y}}(\mathbf{h}_{\mathrm{qN}} - \mathbf{D}\mathbf{y})\mathbf{y}^\top\ . \tag{5.20}$$

The observant reader will notice the symmetry between (5.19) and (5.20). This is further discussed in Section 5.10.

Now, given some initial approximation $\mathbf{D}_0$ (or $\mathbf{B}_0$) (the choice of which shall be discussed later), we can use (5.19) or (5.20) to generate the sequence needed in the framework. However, two important features of the Hessian (or its inverse) would then be disregarded: We wish both matrices $\mathbf{B}$ and $\mathbf{D}$ to be symmetric and positive definite. This is not the case for (5.19) and (5.20), and thus the use of Broyden's formula may lead to steps which are not even downhill, and convergence towards saddle points or maxima will often occur. Hence these formulae are never used for unconstrained optimization.



Broyden's rank-one formula was developed for solving systems of non-linear equations. Further, the formulae have several other applications, e.g. in methods for least squares and minimax optimization.

## 5.7. Symmetric Updating

Since $\mathbf{f}''(\mathbf{x})^{-1}$ is symmetric, it is natural to require $\mathbf{D}$ to be so. If at the same time rank-one updating is required, the basic recursion must have the form

$$\mathbf{D}_{\mathrm{new}} = \mathbf{D} + \mathbf{u}\mathbf{u}^\top\ . \tag{5.21a}$$

The Quasi–Newton condition (5.17a) determines $\mathbf{u}$ uniquely: Substituting (5.21) into (5.17a) and letting $\mathbf{h}$ denote $\mathbf{h}_{\mathrm{qN}}$ yields

$$\mathbf{h} = \mathbf{D}\mathbf{y} + \mathbf{u}\mathbf{u}^\top\mathbf{y} \quad\Longleftrightarrow\quad \mathbf{h} - \mathbf{D}\mathbf{y} = (\mathbf{u}^\top\mathbf{y})\mathbf{u}\ , \tag{5.21b}$$

implying that

$$(\mathbf{h} - \mathbf{D}\mathbf{y})(\mathbf{h} - \mathbf{D}\mathbf{y})^\top = (\mathbf{u}^\top\mathbf{y})^2\mathbf{u}\mathbf{u}^\top\ . \tag{5.21c}$$

The factor $(\mathbf{u}^\top\mathbf{y})^2$ is found simply by taking the inner product with $\mathbf{y}$ on both sides of (5.21b):

$$\mathbf{y}^\top(\mathbf{h} - \mathbf{D}\mathbf{y}) = (\mathbf{u}^\top\mathbf{y})\mathbf{u}^\top\mathbf{y} = (\mathbf{u}^\top\mathbf{y})^2\ . \tag{5.21d}$$

By combining (5.21a–d) we get the *SR1 formula* (symmetric rank-one updating formula)

$$\mathbf{D}_{\mathrm{new}} = \mathbf{D} + \frac{1}{\mathbf{y}^\top\mathbf{u}}\mathbf{u}\mathbf{u}^\top \qquad \text{with } \mathbf{u} = \mathbf{h} - \mathbf{D}\mathbf{y}\ . \tag{5.22}$$

It may be shown that if $\mathbf{h} = \mathbf{D}\mathbf{y}$, then $\mathbf{D}_{\mathrm{new}} = \mathbf{D}$ is the only solution to the problem of finding a symmetric rank-one update which satisfies (5.17a). If, however, $\mathbf{y}^\top\mathbf{u} = 0$ while at the same time $\mathbf{h} \neq \mathbf{D}\mathbf{y}$, then there is no solution, and the updating breaks down. Thus, in case the denominator becomes small we simply set $\mathbf{D}_{\mathrm{new}} = \mathbf{D}$ and avoid division by zero.

The SR1 formula has some interesting properties. The most important is that a Quasi–Newton method without line search based on SR1 will minimize a quadratic function with positive definite Hessian in at most $n+1$ iteration steps, provided the search directions are linearly independent and $\mathbf{y}^\top\mathbf{u}$ remains positive. Further, in this case $\mathbf{D}_{\text{new}}$ equals $\mathbf{f}''(\mathbf{x}^*)^{-1}$ after $n+1$ steps. This important property is called *quadratic termination,* cf. Section 4.1.

The SR1 formula has only been used very little in practice. This is due to the fact that $\mathbf{y}^\top\mathbf{u}$ may vanish, whereby numerical instability is introduced or the updating breaks down.

A similar derivation gives the SR1 formula for approximations to $\mathbf{f}''(\mathbf{x})$ :

$$\mathbf{B}_{\text{new}} = \mathbf{B} + \frac{1}{\mathbf{h}^\top\mathbf{v}}\mathbf{v}\mathbf{v}^\top \qquad \text{with} \quad \mathbf{v} = \mathbf{y} - \mathbf{B}\mathbf{h} \ ,$$

and similar comments can be made.

## 5.8. Preserving Positive Definiteness

Consider Newton's equation (5.2) or a Quasi–Newton equation based on 5.13). The step is determined by

$$\mathbf{G}\mathbf{h} = -\mathbf{f}'(\mathbf{x}) \ , \tag{5.23}$$

where $\mathbf{G} = \mathbf{f}''(\mathbf{x})$ (Newton) or – in the case of Quasi–Newton, $\mathbf{G} = \mathbf{B}$ or $\mathbf{G} = \mathbf{D}^{-1}$. Now, remember definition (2.11): $\mathbf{h}$ is downhill if $\mathbf{h}^\top\mathbf{f}'(\mathbf{x}) < 0$. Taking the inner product with $(-\mathbf{h})$ on both sides of (5.23) we see that

$$\mathbf{h}^\top\mathbf{f}'(\mathbf{x}) = -\mathbf{h}^\top\mathbf{G}\,\mathbf{h} \ ,$$

and this is negative if $\mathbf{G}$ is positive definite.



**Conclusion 5.24**

Newton's equation (5.2) or the Quasi–Newton equation made from (5.13) produces a downhill direction if the coefficient matrix is positive definite.

If we use $\mathbf{D} = \mathbf{I}$ (the identity matrix) in all the steps in the Quasi–Newton framework 5.14, then the method of steepest decent appears. As discussed in Chapter 3 this method has good global convergence properties, but the final convergence is often very slow. If, on the other hand, the iterates are near the solution $\mathbf{x}^*$, a Newton method (and also a Quasi–Newton method with good Hessian approximations) will give good performance, close to quadratic convergence. Thus a good strategy for the updating would be to use $\mathbf{D}$ close to $\mathbf{I}$ in the initial iteration step and then successively let $\mathbf{D}$ approximate $\mathbf{f}''(\mathbf{x})^{-1}$ better and better towards the final phase. This will make the iteration start like the steepest descent and end up somewhat like Newton's method. If in addition, the updating preserves positive definiteness for all coefficient matrices, all steps will be downhill and a reasonable rate of convergence can be expected, since $\mathbf{f}''(\mathbf{x})^{-1}$ is positive (semi-)definite at a minimizer.

## 5.9. The DFP Formula

One of the first updating formulae was proposed by Davidon in 1959. This formula actually has the capability of preserving positive definiteness. The formula was later developed by Fletcher and Powell in 1963, and it is called the *DFP formula.* A proper derivation of this formula is very lengthy, so we confine ourselves to the less rigorous presentation given by Fletcher (1987).

The first observation is that a greater flexibility is allowed for with a rank-two updating formulae, simply because more terms may be adjusted. A symmetric rank-two formula can be written as

$$\mathbf{D}_{\text{new}} = \mathbf{D} + \mathbf{u}\mathbf{u}^\top + \mathbf{v}\mathbf{v}^\top \ ,$$

which inserted in the Quasi–Newton condition (5.17a) gives

$$\mathbf{h} = \mathbf{D}\mathbf{y} + \mathbf{u}\mathbf{u}^\top\mathbf{y} + \mathbf{v}\mathbf{v}^\top\mathbf{y} \ .$$

Since the second updating term has been included, there is no unique determination of $\mathbf{u}$ and $\mathbf{v}$. Fletcher points out that an obvious choice is to try

$$\mathbf{u} = \alpha\mathbf{h} \ , \qquad \mathbf{v} = \beta\mathbf{D}\mathbf{y} \ .$$

Then the Quasi–Newton condition will be satisfied if $\mathbf{u}^\top\mathbf{y} = 1$ and $\mathbf{v}^\top\mathbf{y} = -1$, and this yields the formula

**DFP Updating**

$$\mathbf{D}_{\text{new}} = \mathbf{D} + \frac{1}{\mathbf{h}^\top\mathbf{y}}\,\mathbf{h}\mathbf{h}^\top - \frac{1}{\mathbf{y}^\top\mathbf{v}}\,\mathbf{v}\mathbf{v}^\top \ , \tag{5.25}$$

where

$$\mathbf{h} = \mathbf{x}_{\text{new}} - \mathbf{x}, \quad \mathbf{y} = \mathbf{f}'(\mathbf{x}_{\text{new}}) - \mathbf{f}'(\mathbf{x}), \quad \mathbf{v} = \mathbf{D}\mathbf{y} \ .$$

This was the dominating formula for more than a decade and it was found to work well in practice. In general it is more efficient than the conjugate gradient method (see Chapter 4). Traditionally it has been used in Quasi–Newton methods with exact line search, but it may also be used with soft line search as we shall see in a moment. A method like this has the following important properties:

On quadratic objective functions with positive definite Hessian:

a) it terminates in at most $n$ iterations with $\mathbf{D}_{\text{new}} = \mathbf{f}''(\mathbf{x}^*)^{-1}$,

b) it generates conjugate directions,

c) it generates conjugate gradients if $\mathbf{D}_0 = \mathbf{I}$ ,

provided that the method uses exact line searches.

On general functions:



d) it preserves positive definite $\mathbf{D}$-matrices if $\mathbf{h}_{\text{qN}}{}^\top\mathbf{y} > 0$ in all steps,

e) it gives superlinear final convergence,

f) it gives global convergence for strictly convex objective functions provided that the line searches are exact.

Here we have a method with superlinear final convergence (defined in (2.6)). Methods with this property are very useful because they finish the iteration with fast convergence. Also, in this case

$$\|\mathbf{x}^* - \mathbf{x}_{\text{new}}\| \ll \|\mathbf{x}^* - \mathbf{x}\| \quad \text{for} \quad k \to \infty \ ,$$

implying that $\|\mathbf{x}_{\text{new}} - \mathbf{x}\|$ can be used to estimate the distance from $\mathbf{x}$ to $\mathbf{x}^*$.

**Example 5.6.** The proof of property d) in the above list is instructive, and therefore we give it here:

Assume that $\mathbf{D}$ is positive definite. Then its *Cholesky factor* exists: $\mathbf{D} = \mathbf{C}\mathbf{C}^\top$, and for any non-zero $\mathbf{z} \in \mathbb{R}^n$ we use (5.25) to find

$$\mathbf{z}^\top\mathbf{D}_{\text{new}}\mathbf{z} = \mathbf{x}^\top\mathbf{D}\mathbf{z} + \frac{(\mathbf{z}^\top\mathbf{h})^2}{\mathbf{h}^\top\mathbf{y}} - \frac{(\mathbf{z}^\top\mathbf{D}\mathbf{y})^2}{\mathbf{y}^\top\mathbf{D}\mathbf{y}} \ .$$

We introduced $\mathbf{a} = \mathbf{C}^\top\mathbf{z}$, $\mathbf{b} = \mathbf{C}^\top\mathbf{y}$ and $\theta = \angle(\mathbf{a}, \mathbf{b})$, cf. (2.12), and get

$$\begin{aligned}\mathbf{z}^\top\mathbf{D}_{\text{new}}\mathbf{z} &= \mathbf{a}^\top\mathbf{a} - \frac{(\mathbf{a}^\top\mathbf{b})^2}{\mathbf{b}^\top\mathbf{b}} + \frac{(\mathbf{z}^\top\mathbf{h})^2}{\mathbf{h}^\top\mathbf{y}} \\ &= \|\mathbf{a}\|^2\left(1 - \cos^2\theta\right) + \frac{(\mathbf{z}^\top\mathbf{h})^2}{\mathbf{h}^\top\mathbf{y}} \ .\end{aligned}$$

If $\mathbf{h}^\top\mathbf{y} > 0$, then both terms on the right-hand side are non-negative. The first term vanishes only if $\theta = 0$, i.e. when $\mathbf{a}$ and $\mathbf{b}$ are proportional, which implies that $\mathbf{z}$ and $\mathbf{y}$ are proportional, $\mathbf{z} = \beta\mathbf{y}$ with $\beta \neq 0$. In this case the second term becomes $(\beta\mathbf{y}^\top\mathbf{h})^2/\mathbf{h}^\top\mathbf{y}$ which is positive due to the basic assumption. Hence, $\mathbf{z}^\top\mathbf{D}_{\text{new}}\mathbf{z} > 0$ for any non-zero $\mathbf{z}$ and $\mathbf{D}_{\text{new}}$ is positive definite.

The essential condition $\mathbf{h}^\top \mathbf{y} > 0$ is called the *curvature condition* because it can be expressed as

$$\mathbf{h}^\top \mathbf{f}'_{\text{new}} > \mathbf{h}^\top \mathbf{f}' \ . \tag{5.26}$$

Notice, that if the line search slope condition (2.16) is satisfied then (5.26) is also satisfied since $\mathbf{h}^\top \mathbf{f}' = \varphi'(0)$ and $\mathbf{h}^\top \mathbf{f}'_{\text{new}} = \varphi'(\alpha_s)$, where $\varphi(\alpha)$ is the line search function defined in section 2.1.

The DFP formula with exact line search works well in practice and has been used widely. When the soft line search methods were introduced, however, the DFP formula appeared less favorable because it does not always work well with a soft line search. There is another rank-two updating formula which works better, and the DFP formula only has theoretical importance today. The corresponding formula for updating approximations to the Hessian itself is rather long, and we omit it here.

At this point we shall elaborate on the importance of using soft line search in Quasi Newton methods. The number of iteration steps will usually be larger with the soft line search when compared to an exact line search, but the total number of function evaluations needed to minimize $f$ will be considerably smaller. Clearly, the purpose of using soft line search is to be able to take the steps which are proposed by the Quasi Newton method directly. In this way we can avoid a noticeable number of function evaluations in each iteration step for the determination of the exact minimum of $f$ along the line. Further, in the final iterations, the approximations to the second order derivatives are usually remarkably good and the Quasi–Newton method obtains a fine rate of convergence (see below).

## 5.10. The BFGS Formulae

The final updating formulae to be discussed in these notes are known as the *BFGS formulae,* and they are the most popular of all the updating formulae, described in the literature. As it is the case with the



DFP formula, the BFGS formulae are difficult to derive directly from the requirements. However, they arrive in a funny way through the concept of *duality* which will be discussed briefly here: Remember the Quasi–Newton conditions (5.17):

$$\mathbf{D}_{\text{new}}\mathbf{y} = \mathbf{h} \qquad \text{and} \qquad \mathbf{B}_{\text{new}}\mathbf{h} = \mathbf{y} \ .$$

The second equation has the same form as the first, except that $\mathbf{y}$ and $\mathbf{h}$ are interchanged and $\mathbf{D}_{\text{new}}$ is replaced by $\mathbf{B}_{\text{new}}$. This implies that any updating formula for $\mathbf{D}$ which satisfies (5.17a) can be transformed into an updating formula for $\mathbf{B}$. Further, any formula for $\mathbf{D}$ has a dual formula for $\mathbf{B}$ which is found by the substitution $\mathbf{D} \leftrightarrow \mathbf{B}$ and $\mathbf{h} \leftrightarrow \mathbf{y}$ . Performing this operation on the DFP formula (5.25) yields the following updating formula, which was discovered independently by Broyden, Fletcher, Goldfarb and Shanno in 1970:

**BFGS Updating**

$$\mathbf{B}_{\text{new}} = \mathbf{B} + \frac{1}{\mathbf{h}^\top \mathbf{y}} \mathbf{y}\mathbf{y}^\top - \frac{1}{\mathbf{h}^\top \mathbf{u}} \mathbf{u}\mathbf{u}^\top \ , \tag{5.27}$$

where

$$\mathbf{h} = \mathbf{x}_{\text{new}} - \mathbf{x}, \quad \mathbf{y} = \mathbf{f}'(\mathbf{x}_{\text{new}}) - \mathbf{f}'(\mathbf{x}), \quad \mathbf{u} = \mathbf{B}\mathbf{h} \ .$$

This updating formula is very useful (Dennis and Schnabel (1984) say "It is the best"), and it outperforms the DFP formula. The reader is referred to Nocedal (1992) for an excellent explanation why this is the case. If we perform the dual operation on the BFGS update we return to the DFP updating, as we expected. The BFGS formula produces $\mathbf{B}$ which converges to $\mathbf{f}''(\mathbf{x}^*)$ and the DFP formula produces $\mathbf{D}$ which converges to $\mathbf{f}''(\mathbf{x}^*)^{-1}$.

Alternatively, we can find another set of matrices $\{\mathbf{D}\}$ which has the same convergence, although it is different from the $\mathbf{D}$-matricess produced by DFP. The BFGS formula is a rank two update, and there are formulae which give the corresponding update for $\mathbf{B}^{-1}$ :

**BFGS Updating for D**

$$\mathbf{D}_{\text{new}} = \mathbf{D} + a\mathbf{h}\mathbf{h}^\top - b\left(\mathbf{h}\mathbf{v}^\top + \mathbf{v}\mathbf{h}^\top\right) ,$$

where

$$\mathbf{h} = \mathbf{x}_{\text{new}} - \mathbf{x}, \quad \mathbf{y} = \mathbf{f}'(\mathbf{x}_{\text{new}}) - \mathbf{f}'(\mathbf{x}), \quad \mathbf{v} = \mathbf{D}\mathbf{y},$$

$$b = \frac{1}{\mathbf{h}^\top\mathbf{y}} , \quad a = b\big(1 + b(\mathbf{y}^\top\mathbf{v})\big) .$$

(5.28)

The BFGS formulae are always used together with a soft line search and as discussed above the procedure should be initiated with the full Quasi–Newton step in each iteration step, i.e. the initial $\alpha$ in 2.25 should be one. Experiments show that it should be implemented with a very loose line search; typical values for the parameters in (2.24) are $\varrho = 10^{-4}$ and $\beta = 0.9$.

The properties a) – f) of the DFP formula also hold for the BFGS formulae. Moreover, Powell has proved a better convergence result for the latter formulae namely that they will also converge with a soft line search on convex problems. Unfortunately, convergence towards a stationary point has not been proved for neither the DFP nor the BFGS formulae on general non-linear functions – no matter which type of line search. Still, BFGS with soft line search is known as the method which never fails to come out with a stationary point.

## 5.11. Quadratic Termination

We indicated above that there is a close relationship between the DFP-update and the BFGS-updates. Still, their performances are different with the DFP update performing poorly with soft line searches. Broyden suggested to combine the two sets of formulae:

**Broyden's One Parameter family**

$$\mathbf{D}_{\text{new}} = \mathbf{D} + \sigma\mathbf{W}_{\text{DFP}} + (1-\sigma)\mathbf{W}_{\text{BFGS}} ,$$

where $0 \le \sigma \le 1$ and $\mathbf{W}_{\text{DFP}}$ and $\mathbf{W}_{\text{BFGS}}$ are the updating terms in (5.25) and (5.28), respectively.

(5.29)



The parameter $\sigma$ can be adjusted during the iteration, see Fletcher (1987) for details. He remarks that $\sigma = 0$, "clean" BFGS updating is quite often the best.

We want to state a result for the entire Broyden family, a result which consequently is true for both DFP and BFGS. The result is concerned with *quadratic termination*:

The Broyden One Parameter Updating formula gives quadratic termination for all values of $\sigma$ $(0 \le \sigma \le 1)$, provided that $\mathbf{D}_0$ is positive definite.

This implies that a Quasi–Newton method with exact line searches determines the minimizer of a positive definite quadratic after no more than $n$ iteration steps ($n$ being the dimension of the space).

(5.30)

The basis of all the updating in this chapter is the Quasi–Newton conditions (5.17a–b). This corresponds to a linear interpolation in the gradient of the cost function. If the cost function is quadratic, then its gradient is linear in $\mathbf{x}$, and so is its approximation. When the Quasi–Newton condition has been enforced in $n$ steps, the two linear functions agree in $n+1$ positions in $\mathbb{R}^n$, and consequently the two functions are identical. Iterate no. $n+1$, $\mathbf{x}_{\text{new}}$, makes the gradient of the approximation equal to zero, and so it also makes the gradient of the cost function equal to zero; it solves the problem. The proviso that the quadratic and $\mathbf{D}_0$ must be positive definite, ensures that $\mathbf{x}_{\text{new}}$ is not only a stationary point, but also a minimizer.

## 5.12. Implementation of a Quasi–Newton Method

In this section we shall discuss some details of the implementation and finally show the Quasi–Newton algorithm with the different parts assembeled.

Based on the above discussion we have chosen a BFGS updating formula, and for the reasons given p. 62, an update of the inverse Hes-

sian has been chosen. For student exercises and preliminary research this update is adequate, but even though $\mathbf{D}$ in theory stays positive definite, the rounding errors may cause ill conditioning and even indefiniteness. For professional codes updating of a *factorization* of the Hessian is recommended such that the effect of round off errors can be treated properly. In the present context a less advanced remedy is described which is to omit the updating if the curvature condition (5.26) does not hold, since in this case the new $\mathbf{D}$ would not be positive definite. Actually, Dennis and Schnabel (1984) recommend that the updating is skipped if

$$\mathbf{h}^{\mathsf{T}}\mathbf{y} \leq \varepsilon_{\mathrm{M}}^{1/2}\, \|\mathbf{h}\|_2\, \|\mathbf{y}\|_2 \;, \tag{5.31}$$

where $\varepsilon_{\mathrm{M}}$ is the *machine precision.* As a final remark on the updating formula we shall warn against implementing (5.28) with $O(n^3)$ operations – a frequent error.

We shall assume the availability of a soft line search such as Algorithm 2.25. It is important to notice that all the function evaluations take place during the line search. Hence, the values of $f$ and $\mathbf{f}'$ at the new point are recieved from the line search subprogram. In the next iteration step these values are returned to the subprogram such that $f$ and $\mathbf{f}'$ for $\alpha = 0$ are ready for the next search. Sometimes the gradient needs not be calculated as often as $f$. In a production code the line search should only calculate $f$ respectively $\mathbf{f}'$ whenever they are needed.

The choice of the initial approximation to the inverse Hessian, $\mathbf{D}_0$, must also be discussed. Traditionally it is recommended to use $\mathbf{D}_0 = \mathbf{I}$, the identity matrix. This $\mathbf{D}_0$ is of course positive definite and the first step will be in the steepest descent direction.

Finally, we outline an algorithm for a Quasi–Newton method. Actually, the curvature condition (5.26) needs not be tested because it is incorporated in the soft line search as stopping criterion (2.24b).



**Algorithm 5.32**
**Quasi–Newton Method with BFGS–Updating**

```
begin
  x := x_0;  D := D_0;  k := 0;                          {Initialisation}
  while ||f'(x)|| > ε and k < k_max
    h_qN := D(−f'(x))                                    {Quasi–Newton equation}
    α := soft_line_search(x, h_qN)                       {Algorithm 2.25}
    x_new := x + α h_qN;  k := k+1
    if h_qN^T f'(x_new) > h_qN^T f'(x)                   {Condition (5.26)}
      Update D                                           {using (5.28)}
    x := x_new
end
```

**Example 5.7.** We consider Rosenbrock's function from Examples 4.3 and 5.5. As in the former, we have tried different updating formulae and different line search methods. The line search parameters were chosen as in Example 4.3.

With the starting point $\mathbf{x}_0 = [-1.2,\ 1]^{\mathsf{T}}$, the following numbers of iteration steps and evaluations of $f(\mathbf{x})$ and $\mathbf{f}'(\mathbf{x})$ are needed to satisfy the stopping criterion $\|\mathbf{f}'(\mathbf{x})\| \leq 10^{-10}$,

| Update by | Line search | # it. steps | # fct. evals |
|---|---|---|---|
| DPF | exact | 23 | 295 |
| DPF | soft | 31 | 93 |
| BFGS | exact | 23 | 276 |
| BFGS | soft | 29 | 68 |

The results are as expected: BFGS combined with soft line search needs the smallest number of function evaluations to find the solution.

Below we give the iterates (cf. Figures 4.2, 4.3 and 5.4) and the values of $f(\mathbf{x}_k)$ and $\|\mathbf{f}'(\mathbf{x}_k)\|_\infty$. As with the Damped Newton Method we have superlinear final convergence.

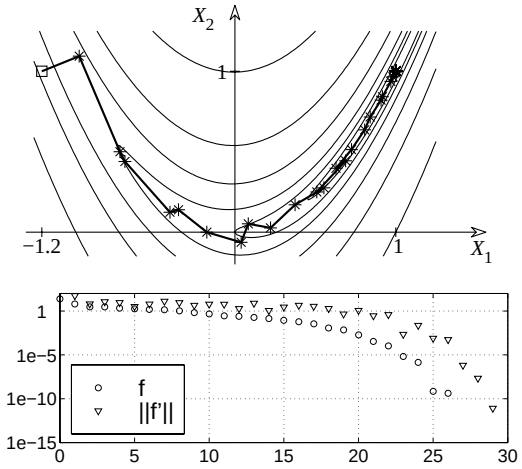



Figure 5.5: *BFGS with soft line search applied to Rosenbrock's function. Top: iterates* $\mathbf{x}_k$. *Bottom:* $f(\mathbf{x}_k)$ *and* $\|\mathbf{f}'(\mathbf{x}_k)\|_\infty$.

The number of iteration steps is about the same as in Example 5.5, while the number of function evaluations is almost four times as big. Note, however, that with Algorithm 5.32 each evaluation involves $f(\mathbf{x})$ and $\mathbf{f}'(\mathbf{x})$, while each evalution in the Damped Newton Method also involves the Hessian $\mathbf{f}''(\mathbf{x})$. For many problems this is not available. If it is, it may be costly: we need to compute $\frac{1}{2}n(n+1$ elements in the symmetric matrix $\mathbf{f}''(\mathbf{x})$, while $\mathbf{f}'(\mathbf{x})$ has $n$ elements only.

# Appendix

## A. Symmetric, Positive Definite Matrices

A matrix $\mathbf{A}\in\mathbb{R}^{n\times n}$ is symmetric if $\mathbf{A}=\mathbf{A}^\top$, i.e. if $a_{ij}=a_{ji}$ for all $i,j$.

**Definition**

The symmetric matrix $\mathbf{A}\in\mathbb{R}^{n\times n}$ is

*positive definite* $\iff$ for all $\mathbf{x}\in\mathbb{R}^n,\ \mathbf{x}\neq\mathbf{0}:\ \mathbf{x}^\top\mathbf{A}\,\mathbf{x}>0$

*positive semidefinite* $\iff$ for all $\mathbf{x}\in\mathbb{R}^n,\ \mathbf{x}\neq\mathbf{0}:\ \mathbf{x}^\top\mathbf{A}\,\mathbf{x}\geq 0$ (A.1)

Such matrices play an important role in optimization, and some useful properties are listed in

**Theorem A**

Let $\mathbf{A}\in\mathbb{R}^{n\times n}$ be symmetric and let $\mathbf{A}=\mathbf{LU}$, where $\mathbf{L}$ is a unit lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix. Then

1° (All $u_{ii}>0,\ i=1,\dots,n$) $\iff$ ($\mathbf{A}$ is positive definite) .

If $\mathbf{A}$ is positive definite, then

2° The LU-factorization is numerically stable.

3° $\mathbf{U}=\mathbf{DL}^\top$ with $\mathbf{D}=\mathrm{diag}(u_{ii})$.

4° $\mathbf{A}=\mathbf{CC}^\top$, the *Cholesky factorization.* $\mathbf{C}\in\mathbb{R}^{n\times n}$ is a lower triangular matrix.

**Proof:** See e.g. Golub and Van Loan (1989) or Nielsen (1996). □



A *unit lower triangular matrix* $\mathbf{L}$ is characterized by $\ell_{ii}=1$ and $\ell_{ij}=0$ for $j>i$. Note, that the LU-factorization $\mathbf{A}=\mathbf{LU}$ is made without pivoting (which, by the way, could destroy the symmetry). Also note that points 3°–4° give the following relation between the LU- and the Cholesky-factorization

$$\mathbf{A}=\mathbf{L}\,\mathbf{U}=\mathbf{LDL}^\top=\mathbf{CC}^\top \tag{A.2a}$$

with

$$\mathbf{C}=\mathbf{LD}^{1/2}\ ,\qquad \mathbf{D}^{1/2}=\mathrm{diag}(\sqrt{u_{ii}})\ . \tag{A.2b}$$

The Cholesky factorization with test for positive definiteness can be implemented as follows. (This algorithm does not rely on (A.2), but is derived directly from 4° in Theorem A).

**Algorithm (A.3). Cholesky factorization**

```
begin
  k := 0;  posdef := true                          {Initialisation}
  while posdef and k < n
    k := k+1
    d := a_kk − Σ_{j=1}^{k−1} (c_kj)^2
    if d > 0                                       {test for pos. def.}
      c_kk := √d                                   {diagonal element}
      for i := k+1,...,n                           {subdiagonal elements}
        c_ik := ( a_ik − Σ_{j=1}^{k−1} c_ij c_kj ) / c_kk
    else
      posdef := false
end
```

The "cost" of this algorithm is $O(n^3)$ operations.

This algorithm can e.g. be used in Algorithm 5.11. Actually it is the cheapest way to check positive-definiteness.

The solution to the system

$$\mathbf{Ax}=\mathbf{b}$$

can be computed via the Cholesky factorization: Inserting $\mathbf{A} = \mathbf{C}\mathbf{C}^\top$ we see that the system splits into

$$\mathbf{C}\mathbf{z} = \mathbf{b} \qquad \text{and} \qquad \mathbf{C}^\top \mathbf{x} = \mathbf{z} \ .$$

The two triangular systems are solved by forward- and back-substitution, respectively.

**Algorithm (A.4). Cholesky solve**

```
begin
  for k := 1,...,n-1,n                                   {forward}
    z_k := ( b_k - sum_{j=1}^{k-1} c_kj z_j ) / c_kk
  for k := n,n-1,...,1                                   {back}
    x_k := ( z_k - sum_{j=k+1}^{n} c_jk x_j ) / c_kk
end
```

The "cost" of this algorithm is $O(n^2)$ operations.

## B. Proof of Theorem 4.2

We shall use induction to show that for $j = 1, \ldots, n$:

$$\mathbf{h}_i^\top \mathbf{H} \mathbf{h}_j = 0 \quad \text{for all } i < j \ . \tag{B.1}$$

We use the notation $\mathbf{g}_i = \mathbf{f}'(\mathbf{x}_i)$ and define the search directions by $\mathbf{h}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$. Then (4.5) leads to

$$\mathbf{H}\mathbf{h}_r = \mathbf{g}_r - \mathbf{g}_{r-1} \ , \tag{B.2}$$

and 4.6 and (4.10) combine to

$$\mathbf{h}_{r+1} = \alpha_{r+1} \left( -\mathbf{g}_r + \gamma_r \alpha_r^{-1} \mathbf{h}_r \right) \quad \text{with} \quad \gamma_r = \frac{\mathbf{g}_r^\top \mathbf{g}_r}{\mathbf{g}_{r-1}^\top \mathbf{g}_{r-1}} \tag{B.3}$$

and $\alpha_{r+1}$ found by exact line search. Finally, we remind the reader of (4.9) and (4.8)

$$\mathbf{h}_r^\top \mathbf{g}_r = 0 \qquad \text{and} \qquad \alpha_{r+1}^{-1} \mathbf{h}_{r+1}^\top \mathbf{g}_r = -\mathbf{g}_r^\top \mathbf{g}_r \ . \tag{B.4}$$



Now, we are ready for the induction:

For $j$=1, (B.1) is trivially satisfied, there is no $\mathbf{h}_i$ vector with $i$<1.

Next, assume that (B.1) holds for all $j = 1, \ldots, k$. Then it follows from the proof of Theorem 4.1 that

$$\mathbf{g}_k^\top \mathbf{h}_i = 0 \quad \text{for} \ \ i = 1, \ldots, k \ . \tag{B.5}$$

If we insert (B.3), we see that this implies

$$0 = \mathbf{g}_k^\top \left( -\mathbf{g}_{i-1} + \gamma_{i-1} \alpha_{i-1}^{-1} \mathbf{h}_{i-1} \right) = -\mathbf{g}_k^\top \mathbf{g}_{i-1} \ .$$

Thus, the gradients at the iterates are orthogonal,

$$\mathbf{g}_k^\top \mathbf{g}_i = 0 \quad \text{for} \ \ i = 1, \ldots, k-1 \ . \tag{B.6}$$

Now, we will show that (B.1) also holds for $j = k+1$ :

$$\begin{aligned} \alpha_{k+1}^{-1} \mathbf{h}_i^\top \mathbf{H}\, \mathbf{h}_{k+1} &= \mathbf{h}_i^\top \mathbf{H} \left( -\mathbf{g}_k + \gamma_k \alpha_k^{-1} \mathbf{h}_k \right) \\ &= -\mathbf{g}_k^\top \mathbf{H} \mathbf{h}_i + \gamma_k \alpha_k^{-1} \mathbf{h}_i^\top \mathbf{H}\, \mathbf{h}_k \\ &= -\mathbf{g}_k^\top \left( \mathbf{g}_i - \mathbf{g}_{i-1} \right) + \gamma_k \alpha_k^{-1} \mathbf{h}_i^\top \mathbf{H}\, \mathbf{h}_k \ . \end{aligned}$$

For $i < k$ each term is zero according to (B.1) for $j \le k$ and (B.5).
For $i = k$ also the term $\mathbf{g}_k^\top \mathbf{g}_{k-1} = 0$, and we get

$$\begin{aligned} \alpha_{k+1}^{-1} \mathbf{h}_k^\top \mathbf{H}\, \mathbf{h}_{k+1} &= -\mathbf{g}_k^\top \mathbf{g}_k + \gamma_k \alpha_k^{-1} \mathbf{h}_k^\top \left( \mathbf{g}_k - \mathbf{g}_{k-1} \right) \\ &= -\mathbf{g}_k^\top \mathbf{g}_k + \gamma_k \left( 0 + \mathbf{g}_{k-1}^\top \mathbf{g}_{k-1} \right) \ = \ 0 \ . \end{aligned}$$

In the first reformulation we use both relations in (B.4), and next we use the definition of $\gamma_k$ in (B.3).

Thus, we have shown that (B.1) also holds for $j = k+1$ and thereby finished the proof. □

# References


1. M. Al-Baali (1985): "Descent Property and Global Convergence of the Fletcher-Reeves method with inexact linesearch", IMA Journal on Num. Anal. **5**, pp 121–124.
2. C. G. Broyden (1965): "A class of methods for solving nonlinear simultaneous equations", Maths. Comp. **19**, pp 577–593.
3. H. P. Crowder & P. Wolfe (1972): "Linear Convergence of the Conjugate Gradient Method", IBM J. Res. Dev. **16**, pp 431–433.
4. J. E. Dennis, Jr. & R.B. Schnabel (1983): "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice Hall.
5. R. Fletcher (1980): "Practical Methods of Optimization", vol 1, John Wiley.
6. R. Fletcher (1987): "Practical Methods of Optimization", vol 2, John Wiley.
7. A. A. Goldstein & J.F. Price (1967): "An Effective algorithm for Minimization", Num. Math. **10**, pp 184–189.
8. G. Golub & C.F. Van Loan (1989): "Matrix Computations", John Hopkins Univ. Press.
9. D. Luenberger (1984): "Linear and Nonlinear Programming", Addison-Wesley.
10. K. Madsen (1984): "Optimering uden bibetingelser" (in Danish), Hæfte 46, Numerisk Institut, DTH.
11. K. Madsen, H.B. Nielsen & O. Tingleff (1999): "Methods for Non-Linear Least Squares Problems", IMM, DTU. Available at http://www.imm.dtu.dk/~hbn/publ/H38.ps.gz





12. J. J. Moré & D.C. Sorenson (1982): "Newtons Method", Argone Nat. Lab. Report ANL-82-8.
13. H.B. Nielsen (1996): "Numerisk Lineær Algebra" (in Danish), IMM, DTU.
14. H.B. Nielsen (1999): "Damping Parameter in Marquardt's Method". IMM, DTU. Report IMM-REP-1999-05. Available at http://www.imm.dtu.dk/~hbn/publ/TR9905.ps.Z
15. J. Nocedal (1992): "Theory of Algorithms for Unconstrained Optimization", in "Acta Numerica 1992", Cambridge Univ. Press.

# INDEX