IMM
DEPARTMENT OF MATHEMATICAL MODELLING

Technical University of Denmark
DK-2800 Lyngby – Denmark

J. No. **H38**
7.7.1999
HBN/ms

METHODS FOR NON-LINEAR LEAST SQUARES PROBLEMS

Kaj Madsen
Hans Bruun Nielsen
Ole Tingleff

IMM

Contents

1. Introduction and Definitions

In this booklet we consider the problem of finding an argument which gives the minimum value of a given differentiable function $F : \mathbb{R}^n \mapsto \mathbb{R}$, the socalled *objective* or *cost* function. In other words:

Definition. Global Minimizer

$$\text{Find} \quad \mathbf{x}^+ = \text{argmin}_{\mathbf{x}}\{F(\mathbf{x})\} \quad \text{where } F : \mathbb{R}^n \mapsto \mathbb{R} \tag{1.1}$$

This problem is very hard to solve in general, and the methods we give here are built to solve the simpler problem of finding a local minimizer for F, an argument vector which gives a minimum value of F inside a certain region whose size is given by δ , $\delta > 0$ and small enough:

Definition. Local minimizer

$$\text{Find} \quad \mathbf{x}^* \quad \text{so that} \quad F(\mathbf{x}) \geq F(\mathbf{x}^*) \quad \text{for} \quad \|\mathbf{x} - \mathbf{x}^*\| < \delta \tag{1.2}$$

The main subject of this booklet is the treatment of methods for a special kind of optimiziation problems where the function F has the following form

Definition. Least Squares Problem

$$\text{Find} \quad \mathbf{x}^*, \text{ a local minimizer for} \quad F(\mathbf{x}) = \tfrac{1}{2}\sum_{i=1}^{m}(f_i(\mathbf{x}))^2 \quad \text{where } f_i : \mathbb{R}^n \mapsto \mathbb{R} \text{ and } m \geq n \tag{1.3}$$

The factor $\frac{1}{2}$ has no effect on $\mathbf{x}^*$, and is introduced for convenience, see page 14.

Example 1.1. An important source of least squares problems is *data fitting*. As an example consider the *data points* $(t_1, y_1), \ldots, (t_m, y_m)$ shown below

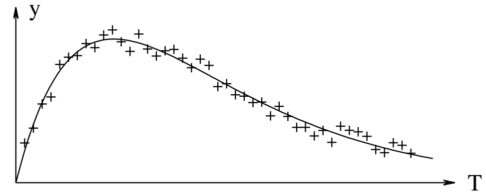


Figure 1.1. *Data points* $\{(t_i, y_i)\}$ *(marked by +) and model* $M(\mathbf{x}, t)$ *(marked by full line)*

Further, we are given a *fitting model*,

$$M(\mathbf{x}, t) = x_3 e^{x_1 t} + x_4 e^{x_2 t} .$$

The model depends on the *parameters* $\mathbf{x} = [x_1, x_2, x_3, x_4]^\top$. We assume that there exists an $\mathbf{x}^\dagger$ so that

$$y_i = M(\mathbf{x}^\dagger, t_i) + \varepsilon_i ,$$

where the $\{\varepsilon_i\}$ are (measurement) errors on the data ordinates, assumed to behave like "white noise".

For any choice of $\mathbf{x}$ we can compute the *residuals*

$$f_i(\mathbf{x}) = y_i - M(\mathbf{x}, t_i) = y_i - x_3 e^{x_1 t_i} - x_4 e^{x_2 t_i} , \quad i = 1, \ldots, m .$$

For a *least squares fit* the parameters are determined as the minimizer $\mathbf{x}^*$ of the sum of squared residuals. This is seen to be a problem of the form (1.3) with $m = 45$, $n = 4$. The graph of $M(\mathbf{x}^*, t)$ is shown by full line in Figure 1.1.

In the remainder of this introduction we shall discuss some basic concepts in optimization, and Chapter 2 is a brief review of methods for finding a local minimizer for general cost functions. For more details we refer to Frandsen et al. (1999). In Chapter 3 we give methods that are specially tuned for least squares problems.

We assume that the cost function F is so smooth that the following *Taylor expansion* is valid,[1)]

$$F(\mathbf{x}+\mathbf{h}) = F(\mathbf{x}) + \mathbf{h}^\top \mathbf{g} + \tfrac{1}{2}\mathbf{h}^\top \mathbf{H}\,\mathbf{h} + O(\|\mathbf{h}\|^3)\ , \tag{1.4a}$$

where $\mathbf{g}$ is the *gradient*,

$$\mathbf{g} \ \equiv\ \mathbf{F}'(\mathbf{x}) \ \equiv \begin{bmatrix} \dfrac{\partial F}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \dfrac{\partial F}{\partial x_n}(\mathbf{x}) \end{bmatrix}\ , \tag{1.4b}$$

and $\mathbf{H}$ is the *Hessian matrix*,

$$\mathbf{H} \ \equiv\ \mathbf{F}''(\mathbf{x}) \ \equiv \left[\frac{\partial^2 F}{\partial x_i \partial x_j}(\mathbf{x})\right]\ . \tag{1.4c}$$

If $\mathbf{x}^*$ is a local minimizer and $\|\mathbf{h}\|$ is sufficiently small, then we cannot find a point $\mathbf{x}^*{+}\mathbf{h}$ with a smaller F-value. Combining this observation with (1.4a) we see that

Necessary Condition for a Local Minimizer

$$\mathbf{x}^* \text{ is a local minimizer} \\ \Longrightarrow \\ \mathbf{g}^* \ \equiv\ \mathbf{F}'(\mathbf{x}^*) \ =\ \mathbf{0} \tag{1.5}$$

We use a special name for arguments that satisfy the necessary condition:

$\mathbf{x}_\mathrm{s}$ is a Stationary Point

$$\Longleftrightarrow \\ \mathbf{g}_\mathrm{s} \ \equiv\ \mathbf{F}'(\mathbf{x}_\mathrm{s}) \ =\ \mathbf{0} \tag{1.6}$$

1) Unless otherwise specified, $\|\cdot\|$ denotes the 2-norm, $\|\mathbf{h}\| = \sqrt{h_1^2 + \cdots + h_n^2}$.

Thus, the local minimizers are also stationary points, but so are the local maximizers. A stationary point which is neither a local maximizer nor a local minimizer is called a *saddle point*. In order to determine whether a given stationary point is a local minimizer or not, we need to include the second order term in the Taylor series (1.4a). Inserting $\mathbf{x}_\mathrm{s}$ we see that

$$F(\mathbf{x}_\mathrm{s}+\mathbf{h}) = F(\mathbf{x}_\mathrm{s}) + \tfrac{1}{2}\mathbf{h}^\top \mathbf{H}_\mathrm{s}\,\mathbf{h} + O(\|\mathbf{h}\|^3) \\ \text{with}\ \ \mathbf{H}_\mathrm{s} \equiv \mathbf{F}''(\mathbf{x}_\mathrm{s})\ . \tag{1.7}$$

From definition (1.4c) of the Hessian matrix it follows that any $\mathbf{H}$ is symmetric. If we request that $\mathbf{H}_\mathrm{s}$ is *positive definite*, then its eigenvalues are greater than some number $\delta > 0$ (see Appendix A), and

$$\mathbf{h}^\top \mathbf{H}_\mathrm{s}\,\mathbf{h} > \delta\,\|\mathbf{h}\|^2\ .$$

This shows that for $\|\mathbf{h}\|$ sufficiently small the third term on the right-hand side of (1.7) will be dominated by the second. This term is positive, so that we get

Sufficient Condition for a Local Minimizer

$$\mathbf{x}^* \text{ is a stationary point and } \mathbf{F}''(\mathbf{x}^*) \text{ is positive definite} \\ \Longrightarrow \\ \mathbf{x}^* \text{ is a local minimizer} \tag{1.8}$$

If $\mathbf{H}_\mathrm{s}$ is *negative definite*, then $\mathbf{x}_\mathrm{s}$ is a local maximizer. If $\mathbf{H}_\mathrm{s}$ is *indefinite* (i.e. it has both positive and negative eigenvalues), then $\mathbf{x}_\mathrm{s}$ is a saddle point.

2. Descent Methods

All methods for non-linear optimization are iterative: From a starting point $\mathbf{x}_0$ the method produces a series of vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots$, which (hopefully) converges against $\mathbf{x}^*$, a local minimizer for the given function, see Definition (1.2). Most methods have measures which enforce the *descending condition*

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k) \ . \tag{2.1}$$

This prevents convergence to a maximizer and also makes it less probable that we converge towards a saddle point, cf. Chapter 1. If the given function has several minimizers the result will depend on the starting point $\mathbf{x}_0$. We do not know which of the minimizers that will be found; quite often it is not the minimizer closest to $\mathbf{x}_0$.

In many cases the method produces vectors which converge towards the minimizer in 2 clearly different stages. When $\mathbf{x}_0$ is far from the solution we want the method to produce iterates which move steadily towards $\mathbf{x}^*$. In this "global stage" of the iteration we are satisfied if the errors do not increase except in the very first steps, i.e.

$$\|\mathbf{e}_{k+1}\| < \|\mathbf{e}_k\| \quad \text{for} \quad k > K \ , \tag{2.2}$$

where $\mathbf{e}_k$ denotes the current error,

$$\mathbf{e}_k \equiv \mathbf{x}_k - \mathbf{x}^* \ . \tag{2.2b}$$

In the final stage of the iteration, where $\mathbf{x}_k$ is close to $\mathbf{x}^*$, we want faster convergence. We distinguish between

Linear convergence:

$$\|\mathbf{e}_{k+1}\| \le a\|\mathbf{e}_k\| \quad \text{when } \|\mathbf{e}_k\| \text{ is small;} \quad 0 < a < 1 \ , \tag{2.3a}$$

Quadratic convergence:

$$\|\mathbf{e}_{k+1}\| = O(\|\mathbf{e}_k\|^2) \quad \text{when } \|\mathbf{e}_k\| \text{ is small} \ , \tag{2.3b}$$

Superlinear convergence:

$$\|\mathbf{e}_{k+1}\|/\|\mathbf{e}_k\| \to 0 \quad \text{for } k \to \infty \ . \tag{2.3c}$$

The methods presented in this booklet are descent methods which satisfy the descending condition (2.1) in each step of the iteration. One step from the current iterate consists in

1. Find a descent direction $\mathbf{h}_{\rm dd}$ (discussed below), and
2. find a steplength giving a good decrease in the F-value.

Thus an outline of a descent method is

Algorithm 2.4. Descent Method

```
begin
  k := 0;  x := x0;  found := false                    {Starting point}
  while not found and k < kmax
    hdd := search_direction(x)                   {From x and downhill}
    if no such h exists
      found := true                                  {x is stationary}
    else
      α := line_search(x, hdd)                 {from x in direction hdd}
      x := x + α hdd;  k := k+1                         {next iterate}
end                                         {... of descent algorithm }
```

Consider the variation of the F-value along the half line starting at $\mathbf{x}$ and with direction $\mathbf{h}$. From the Taylor series (1.4a) we see that

$$\begin{aligned} F(\mathbf{x}+\alpha\mathbf{h}) &= F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) + O(\alpha^2) \\ &\simeq F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) \quad \text{for } \alpha \text{ sufficiently small.} \end{aligned} \tag{2.5}$$

We say that $\mathbf{h}$ is a *descent direction* if $F(\mathbf{x}+\alpha\mathbf{h})$ is a decreasing function of α at $\alpha = 0$. This leads to the following

$$\boxed{\begin{array}{c}\textbf{Definition. } \mathbf{h} \textbf{ is a Descent Direction for } F \textbf{ at } \mathbf{x} \\ \Longleftrightarrow \\ \mathbf{h}^\top \mathbf{F}'(\mathbf{x}) < 0\end{array}} \qquad (2.6)$$

If no such $\mathbf{h}$ exists, then $\mathbf{F}'(\mathbf{x}) = \mathbf{0}$, showing that in this case $\mathbf{x}$ is stationary.

We want to fulfil the descending property (2.1). In other words, we want $F(\mathbf{x}+\alpha\mathbf{h}) < F(\mathbf{x})$. In some methods we want to find (an approximation to) the best value of α, i.e.

$$\alpha_\mathrm{e} = \mathrm{argmin}_{\alpha>0}\{F(\mathbf{x}+\alpha\mathbf{h})\} \ . \qquad (2.7)$$

The process of finding a good value for α is called a *line search*; this is discussed in Section 2.3.

2.1. The Steepest Descent method

From (2.5) we see that when we perform a step $\alpha\mathbf{h}$ with positive α, then the relative gain in function value satisfies

$$\lim_{\alpha\to 0} \frac{F(\mathbf{x}) - F(\mathbf{x}+\alpha\mathbf{h})}{\alpha\|\mathbf{h}\|} = -\frac{1}{\|\mathbf{h}\|}\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) = -\|\mathbf{F}'(\mathbf{x})\|\cos\theta \ ,$$

where θ is the angle between the vectors $\mathbf{h}$ and $\mathbf{F}'(\mathbf{x})$. This shows that we get the greatest gain rate if $\theta = \pi$, i.e. if we use the steepest descent direction $\mathbf{h}_\mathrm{sd}$ given by

$$\mathbf{h}_\mathrm{sd} = -\mathbf{F}'(\mathbf{x}) \ . \qquad (2.8)$$

The method based on (2.8) (i.e. $\mathbf{h}_\mathrm{dd} = \mathbf{h}_\mathrm{sd}$ in Algorithm 2.4) is called the *steepest descent method* or *gradient method*. The choice of descent direction is "the best" (locally) and we could combine it with an exact line search (2.7). A method like this converges, but the final convergence is linear and often very slow. Examples in Frandsen et al. (1999) show how the steepest descent method with exact line search and finite computation accuracy can fail to find the minimizer of a second degree polynomial. However, for many problems the method has quite good performance in the initial stage of the convergence.

Considerations like this has lead to the socalled *hybrid methods*, which – as the name suggests – are based on two different methods. One which is good in the initial stage, like the gradient method, and another method which is good in the final stage, like Newton's method; see the next section. A major problem with a hybrid method is the mechanism which switches between the two methods when appropriate.

2.2. Newton's Method

We can derive this from the condition that $\mathbf{x}^*$ is a stationary point. According to (1.6) it satisfies $\mathbf{F}'(\mathbf{x}^*) = \mathbf{0}$. This is a nonlinear system of equations, and from the Taylor expansion

$$\begin{aligned}\mathbf{F}'(\mathbf{x}+\mathbf{h}) &= \mathbf{F}'(\mathbf{x}) + \mathbf{F}''(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2) \\ &\simeq \mathbf{F}'(\mathbf{x}) + \mathbf{F}''(\mathbf{x})\mathbf{h} \quad \text{for } \|\mathbf{h}\| \text{ sufficiently small}\end{aligned} \qquad (2.9)$$

we derive *Newton's method:* Find $\mathbf{h}_\mathrm{N}$ as the solutions to

$$\mathbf{H}\,\mathbf{h}_\mathrm{N} = -\mathbf{F}'(\mathbf{x}) \quad \text{with } \mathbf{H} = \mathbf{F}''(\mathbf{x}) \ , \qquad (2.10\text{a})$$

and compute the next iterate by

$$\mathbf{x} := \mathbf{x} + \mathbf{h}_\mathrm{N} \ . \qquad (2.10\text{b})$$

Suppose that $\mathbf{H}$ is positive definite, then it is nonsingular (implying that (2.10a) has a unique solution), and $\mathbf{u}^\top\mathbf{H}\,\mathbf{u} > 0$ for all nonzero $\mathbf{u}$. Thus, by multiplying with $\mathbf{h}_\mathrm{N}^\top$ on both sides of (2.10b) we get

$$0 < \mathbf{h}_\mathrm{N}^\top\mathbf{H}\,\mathbf{h}_\mathrm{N} = -\mathbf{h}_\mathrm{N}^\top\mathbf{F}'(\mathbf{x}) \ , \qquad (2.11)$$

showing that $\mathbf{h}_\mathrm{N}$ is a descent direction: it satisfies (2.6).

Newton's method is very good in the final stage of the iteration, where $\mathbf{x} \simeq \mathbf{x}^*$. We can show (see Frandsen et al. (1999)) that if the Hessian matrix at the solution is positive definite (the sufficient condition (1.8) is satisfied) and if we are at a position inside the region about $\mathbf{x}^*$ where $\mathbf{F}''(\mathbf{x})$ is positive definite, then we get quadratic convergence, see (2.3). In the opposite situation, i.e. $\mathbf{x}$ is in a region where $\mathbf{F}''(\mathbf{x})$ is negative definite everywhere, and where there is a stationary point, the "raw" Newton method (2.10) would converge (quadratically) towards this stationary point, which is a maximizer. We do not want this, and we can avoid it by requiring that all steps taken are in descent directions.

Now we can build a hybrid method, based on Newtons method:

```
if h_N is a descent direction
  use h_N
else
  use h_sd
```

The controlling mechanism is the descent condition, $\mathbf{h}_N^\top \mathbf{F}'(\mathbf{x}) < 0$. As shown in (2.11), this is satisfied if $\mathbf{F}''(\mathbf{x})$ is positive definite, so a sketch of the central section of this version of the algorithm is:

$$\begin{array}{l} \texttt{if } \mathbf{F}''(\mathbf{x}) \text{ is positive definite} \\ \quad \texttt{if } F(\mathbf{x}+\mathbf{h}_N) < F(\mathbf{x}) \\ \quad\quad \mathbf{x} := \mathbf{x} + \mathbf{h}_N \\ \quad \texttt{else} \\ \quad\quad \mathbf{x} := \mathbf{x} + \alpha\mathbf{h}_N \\ \texttt{else} \\ \quad \mathbf{x} := \mathbf{x} + \alpha\mathbf{h}_{sd} \end{array} \tag{2.12}$$

Here, $\mathbf{h}_{sd}$ is the steepest descent direction and α is found by a line search; see Section 2.3. This is included also with the Newton direction to make sure that the descending condition (2.1) is satisfied.

An alternative reaction when the Hessian is not positive definite, is the use of a socalled *damped Newton method:*

$$\begin{array}{l} \texttt{if } \mathbf{F}''(\mathbf{x}) \text{ is not positive definite} \\ \quad \text{Find } \mu \text{ so that } \mathbf{F}''(\mathbf{x})+\mu\mathbf{I} \text{ is positive definite} \\ \quad \text{Find } \mathbf{h}_{dN} \text{ by solving } (\mathbf{F}''(\mathbf{x})+\mu\mathbf{I})\mathbf{h}_{dN} = -\mathbf{F}'(\mathbf{x}) \end{array} \tag{2.13}$$

The step $\mathbf{h}_{dN}$ produced in this way is a descent direction and can be used instead of $\mathbf{h}_{sd}$ in the hybrid (2.12). For the indefinite case we use $\mu > 0$, and with μ large, $\mathbf{h}_{dN}$ is a short step whose direction is close to the steepest descent direction $\mathbf{h}_{sd}$, whereas we use the Newton step $\mathbf{h}_N$ in the definite case. Thus we can say that μ interpolates between these two directions. Notice that a good tool for checking a matrix for positive definiteness is Cholesky's method (see Appendix A) which, when succesful, is also used for solving the linear system in question. Thus, the check for definiteness is almost for free.

The hybrid methods indicated above can be very efficient, but they are hardly ever used. The reason is that they need an implementation of $\mathbf{F}''(\mathbf{x})$, and for complicated application problems this is not available. Instead we have to make do with a socalled *Quasi-Newton method,* based on series of matrices which gradually approach $\mathbf{H}^* = \mathbf{F}''(\mathbf{x}^*)$, or $(\mathbf{H}^*)^{-1}$ or a factorization of $\mathbf{H}^*$. In Section 3.3 we present such a method. See also Frandsen et al. (1999).

2.3. Line Search

Given a point $\mathbf{x}$ and a descent direction $\mathbf{h}$. The next iteration step is a move from $\mathbf{x}$ in direction $\mathbf{h}$. To find out, how far to move, we study the variation of the given function along the half line from $\mathbf{x}$ in the direction $\mathbf{h}$:

$$\phi(\alpha) = F(\mathbf{x}+\alpha\mathbf{h}) , \quad \mathbf{x} \text{ and } \mathbf{h} \text{ fixed}, \ \alpha \geq 0 . \tag{2.14}$$

An example of the behaviour of $\phi(\alpha)$ is shown in Figure 2.1 below.

Our $\mathbf{h}$ being a descent direction ensures that

$$\phi'(0) = \mathbf{h}^\top \mathbf{F}'(\mathbf{x}) < 0 ,$$

indicating that if α is sufficiently small, we satisfy the descending condition (2.1), which is equivalent with

$$\phi(\alpha) < \phi(0) . \tag{2.15}$$

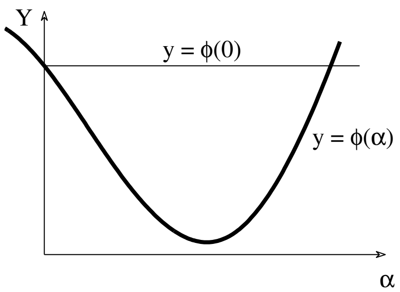


Figure 2.1. *Variation of the cost function along the search line*

Often, we are given an initial guess on α, e.g. $\alpha = 1$ with Newton's method. Figure 2.1 illustrates that three different situations can arise

1° α is so small that the gain in value of the objective function is very small. We should increase α.

2° α is too large: $\phi(\alpha) \geq \phi(0)$. We must decrease α in order to satisfy the descent condition (2.1).

3° α is close to the minimizer[1)] of $\phi(\alpha)$. We happily accept this α-value.

An *exact line search* is an iterative process producing a series $\alpha_1, \alpha_2 \ldots$. The aim is to find the true minimizer α_e definied in (2.7), and the algorithm stops when the iterate α_s satisfies

$$|\phi'(\alpha_\mathrm{s})| \leq \tau\, |\phi'(0)| ,$$

where τ is a small, positive number. In the iteration we can use approximations to the variation of $\phi(\alpha)$ based on the computed values of

1) More precisely: the smallest local minimizer of ϕ. If we increase α beyond the interval shown in Figure 2.1, then it may well happen that we get close to another local minimum for F.

$$\phi(\alpha_k) = F(\mathbf{x}{+}\alpha_k\mathbf{h}) \quad \text{and} \quad \phi'(\alpha_k) = \mathbf{h}^\top \mathbf{F}'(\mathbf{x}{+}\alpha_k\mathbf{h}) .$$

See Sections 2.5 – 2.6 in Frandsen et al. (1999) for details.

An exact line search can waste a lot of computing time: When $\mathbf{x}$ is far from $\mathbf{x}^*$, the search direction $\mathbf{h}$ may be far from the direction $\mathbf{x}^* - \mathbf{x}$, and there is no need to find the true minimum of ϕ very accurately. This is the background for the socalled *soft line searches,* where we accept an α-value if it does not fall in the categories 1° or 2° listed above. We use a stricter version of the descending condition (2.1), viz.

$$\phi(\alpha_\mathrm{s}) \leq \phi(0) + \varrho \cdot \phi'(0) \cdot \alpha \quad \text{with} \quad 0 < \varrho < 0.5 . \tag{2.16a}$$

This ensures that we are not in case 2°. Case 1° corresponds to the point $(\alpha, \phi(\alpha))$ being too close to the starting tangent, and we supplement with the condition

$$\phi'(\alpha_\mathrm{s}) \geq \beta \cdot \phi'(0) \quad \text{with} \ \varrho < \beta < 1 . \tag{2.16b}$$

If the starting guess on α satisfies both these criteria, then we accept it as α_s. Otherwise, we have to iterate as outlined for exact line search. Details can be seen in Section 2.5 of Frandsen et al. (1999).

3. Non-Linear Least Squares Problems

In the remainder of this booklet we shall discuss methods for nonlinear least squares problems. Given a vector function $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$ with $m \geq n$. We want to minimize $\|\mathbf{f}(\mathbf{x})\|$, or equivalently to find

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}}\{F(\mathbf{x})\} , \tag{3.1a}$$

where

$$F(\mathbf{x}) = \tfrac{1}{2}\sum_{i=1}^{m}(f_i(\mathbf{x}))^2 = \tfrac{1}{2}\|\mathbf{f}(\mathbf{x})\|^2 = \tfrac{1}{2}\mathbf{f}(\mathbf{x})^\top\mathbf{f}(\mathbf{x}) . \tag{3.1b}$$

Least squares problems can be solved by general optimization methods, but we shall present special methods that are more efficient. In many cases they achieve better than linear convergence, sometimes even quadratic convergence, even though they do not need implementation of second derivatives.

In the description of the methods in this chapter we shall need formulae for derivatives of F: Provided that $\mathbf{f}$ has continuous second partial derivatives, we can write its *Taylor series* as

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) = \mathbf{f}(\mathbf{x}) + \mathbf{J}_f(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2) , \tag{3.2a}$$

where $\mathbf{J}_f \in \mathbb{R}^{m\times n}$ is the *Jacobian matrix* containing the first partial derivatives of the function components,

$$(\mathbf{J}_f(\mathbf{x}))_{ij} = \frac{\partial f_i}{\partial x_j}(\mathbf{x}) . \tag{3.2b}$$

As regards $F : \mathbb{R}^n \mapsto \mathbb{R}$, it follows from the first formulation in (3.1b),

that[1)]

$$\frac{\partial F}{\partial x_j}(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})\frac{\partial f_i}{\partial x_j}(\mathbf{x}) . \tag{3.3}$$

Thus, the gradient (1.4b) is

$$\mathbf{F}'(\mathbf{x}) = \mathbf{J}_f(\mathbf{x})^\top\mathbf{f}(\mathbf{x}) . \tag{3.4a}$$

We shall also need the Hessian matrix of F. From (3.3) we see that the element in position (j,k) is

$$\frac{\partial^2 F}{\partial x_j \partial x_k}(\mathbf{x}) = \sum_{i=1}^{m}\left(\frac{\partial f_i}{\partial x_j}(\mathbf{x})\frac{\partial f_i}{\partial x_k}(\mathbf{x}) + f_i(\mathbf{x})\frac{\partial^2 f_i}{\partial x_j \partial x_k}(\mathbf{x})\right) ,$$

showing that

$$\mathbf{F}''(\mathbf{x}) = \mathbf{J}_f(\mathbf{x})^\top\mathbf{J}_f(\mathbf{x}) + \sum_{i=1}^{m} f_i(x)\mathbf{f}_i''(\mathbf{x}) . \tag{3.4b}$$

Example 3.1. The simplest case of (3.1) is when $\mathbf{f}(\mathbf{x})$ has the form

$$\mathbf{f}(\mathbf{x}) = \mathbf{b} - \mathbf{A}\mathbf{x} ,$$

where the vector $\mathbf{b} \in \mathbb{R}^m$ and matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ are given. We say that this is a *linear least squares problem.* In this case $\mathbf{J}_f(\mathbf{x}) = -\mathbf{A}$ for all $\mathbf{x}$, and from (3.4a) we see that

$$\mathbf{F}'(\mathbf{x}) = -\mathbf{A}^\top(\mathbf{b} - \mathbf{A}\mathbf{x}) .$$

This is zero for $\mathbf{x}^*$ determined as the solution to the socalled *normal equations,*

$$(\mathbf{A}^\top\mathbf{A})\mathbf{x}^* = \mathbf{A}^\top\mathbf{b} . \tag{3.5}$$

The problem can be written in the form

$$\mathbf{A}\mathbf{x}^* \simeq \mathbf{b} ,$$

and alternatively we can solve it via *orthogonal transformation:* Find an orthogonal matrix $\mathbf{Q}$ so that

1) If we had not used the factor $\frac{1}{2}$ in the definition (1.3), we would have got an annoying factor of 2 in a lot of expressions.

$$\mathbf{Q}^\top \mathbf{A} = \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix} ,$$

where $\mathbf{R} \in \mathbb{R}^{n\times n}$ is upper triangular. The solution is found by back substitution in the system[2)]

$$\mathbf{R}\mathbf{x}^* = (\mathbf{Q}^\top \mathbf{b})_{1:n} .$$

This is the method employed in MATLAB. It is more accurate than the solution via the normal equations.

As the title of the booklet suggests, we assume that $\mathbf{f}$ is nonlinear, and shall not discuss linear problems in detail. We refer to Chapter 6 in Nielsen (1996) or Section 5.2 in Golub and Van Loan (1989).

Example 3.2. In Example 1.1 we saw a nonlinear least squares problem arising from data fitting. Another application is in the solution of nonlinear systems of equations,

$$\mathbf{f}(\mathbf{x}^*) = \mathbf{0} , \quad \text{where} \quad \mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n .$$

We can use *Newton-Raphson's method:* From an initial guess $\mathbf{x}_0$ we compute $\mathbf{x}_1, \mathbf{x}_2, \ldots$ by the algorithm, which is based on seeking $\mathbf{h}$ so that $\mathbf{f}(\mathbf{x}+\mathbf{h}) = \mathbf{0}$ and ignoring the term $O(\|\mathbf{h}\|^2)$ in (3.2a),

$$\begin{aligned} &\text{Solve } \; \mathbf{J}_f(\mathbf{x}_k)\mathbf{h}_k = -\mathbf{f}(\mathbf{x}_k) \; \text{ for } \mathbf{h}_k \\ &\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h}_k \; . \end{aligned} \tag{3.6}$$

Here, the Jacobian matrix $\mathbf{J}_f$ is given by (3.2b). If $\mathbf{J}_f(\mathbf{x}^*)$ is nonsingular, then the method has quadratic final convergence, i.e. if $d_k = \|\mathbf{x}_k - \mathbf{x}^*\|$ is small, then $\|\mathbf{x}_{k+1} - \mathbf{x}^*\| = O(d_k^2)$. However, if $\mathbf{x}_k$ is far from $\mathbf{x}^*$, then we risk to get even further away, but as in Section 2.2 we can supply the method with a line search: We are seeking a zero for $\mathbf{f}(\mathbf{x})$. This is a minimizer of the function F defined by (3.1),

$$F(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{f}(\mathbf{x})\|^2 ,$$

with $F(\mathbf{x}^*) = 0$ and $F(\mathbf{x}) > 0$ if $\mathbf{f}(\mathbf{x}) \neq \mathbf{0}$. We get a *robust method* by modifying (3.6) to

$$\begin{aligned} &\text{Solve } \; \mathbf{J}_f(\mathbf{x}_k)\mathbf{h}_k = -\mathbf{f}(\mathbf{x}_k) \; \text{ for } \mathbf{h}_k \\ &\alpha_k = \operatorname{argmin}_{\alpha>0}\{\tfrac{1}{2}\|\mathbf{f}(\mathbf{x}_k+\alpha\mathbf{h}_k)\|^2\} \\ &\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{h}_k \; . \end{aligned}$$

Here (an approximation to) α_k is found as outlined in Section 2.3.

The method may fail if the Jacobian matrix has singularities in the neighbourhood of $\mathbf{x}^*$. As an example consider the following problem, taken from Powell (1970),

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} x_1 \\ \frac{10x_1}{x_1+0.1} + 2x_2^2 \end{bmatrix} ,$$

with $\mathbf{x}^* = \mathbf{0}$ as the only solution.

If we take $\mathbf{x}_0 = [\,3,\ 1\,]^\top$ and use the above algorithm with exact line search, then the iterates converge to $\mathbf{x}_c \simeq [\,1.8016,\ 0\,]^\top$, which is **not** a solution. To explain this behaviour, we look at the Jacobian matrix

$$\mathbf{J}_f(\mathbf{x}) = \begin{bmatrix} 1 & 0 \\ (x_1+0.1)^{-2} & 4x_2 \end{bmatrix} .$$

This is singular for $x_2 = 0$, and for $\mathbf{x}_k$ close to the X_1-axis the vector $\mathbf{h}_k = \mathbf{J}_f(\mathbf{x}_k)^{-1}\mathbf{f}(\mathbf{x}_k)$ will have large components. Then α_k will be very small, and we get stuck at the current position. A rigorous proof is given by Powell (1970).

An alternative approach is to reformulate the problem so that we aim directly at minimizing F, instead of just using it in the line search. By itself this does not cure the problems associated with singular Jacobian matrices, but it allows us to use all the "tools" we are going to present in this chapter.

[2)] An expression like $\mathbf{u}_{p:q}$ is used to denote the subvector with elements u_i, $i = p, \ldots, q$. The ith row and jth column of a matrix $\mathbf{A}$ is denoted $\mathbf{A}_{i,:}$ and $\mathbf{A}_{:,j}$, respectively.

3.1. The Gauss-Newton Method

This method is the basis of the very efficient method we will describe in the next section. It is based on implemented first derivatives of the components of the vector function. In special cases it can give quadratic convergence as the Newton-method does for general optimization, see Frandsen et al. (1999). As we shall see in an example, there is a risk of convergence towards a non stationary point if we incorporate an exact line search into it.

The basis of the Gauss-Newton Method is a linear approximation to the components of $\mathbf{f}$ (a *linear model* of $\mathbf{f}$) in the neighbourhood of $\mathbf{x}$: For small $\|\mathbf{h}\|$ we see from the Taylor expansion (3.2) that

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) \simeq \boldsymbol{\ell}(\mathbf{h}) \equiv \mathbf{f}(\mathbf{x}) + \mathbf{J}_f(\mathbf{x})\mathbf{h} . \tag{3.7a}$$

Inserting this in the definition (3.1) of F we see that

$$\begin{aligned} F(\mathbf{x}+\mathbf{h}) \simeq L(\mathbf{h}) &\equiv \tfrac{1}{2}\boldsymbol{\ell}(\mathbf{h})^\top \boldsymbol{\ell}(\mathbf{h}) \\ &= \tfrac{1}{2}\mathbf{f}^\top\mathbf{f} + \mathbf{h}^\top \mathbf{J}_f^\top \mathbf{f} + \tfrac{1}{2}\mathbf{h}^\top \mathbf{J}_f^\top \mathbf{J}_f \mathbf{h} \\ &= F(\mathbf{x}) + \mathbf{h}^\top \mathbf{J}_f^\top \mathbf{f} + \tfrac{1}{2}\mathbf{h}^\top \mathbf{J}_f^\top \mathbf{J}_f \mathbf{h} \end{aligned} \tag{3.7b}$$

(with $\mathbf{f} = \mathbf{f}(\mathbf{x})$ and $\mathbf{J}_f = \mathbf{J}_f(\mathbf{x})$). The *Gauss-Newton step* $\mathbf{h}_{\mathrm{GN}}$ minimizes $L(\mathbf{h})$,

$$\mathbf{h}_{\mathrm{GN}} = \mathrm{argmin}_{\mathbf{h}}\{L(\mathbf{h})\} .$$

It is easily seen that the gradient and the Hessian matrix of L are

$$\mathbf{L}'(\mathbf{h}) = \mathbf{J}_f^\top \mathbf{f} + \mathbf{J}_f^\top \mathbf{J}_f \mathbf{h}, \qquad \mathbf{L}''(\mathbf{h}) = \mathbf{J}_f^\top \mathbf{J}_f . \tag{3.8}$$

Comparison with (3.4a) shows that $\mathbf{L}'(\mathbf{0}) = \mathbf{F}'(\mathbf{x})$. Further, we see that the matrix $\mathbf{L}''(\mathbf{h})$ is independent of $\mathbf{h}$. It is symmetric and if $\mathbf{J}_f$ has *full rank,* i.e. if the columns are linearly independent, then $\mathbf{L}''(\mathbf{h})$ is also positive definite, cf. Appendix A. This implies that $L(\mathbf{h})$ has a unique minimizer, which can be found by solving

$$(\mathbf{J}_f^\top \mathbf{J}_f)\mathbf{h}_{\mathrm{GN}} = -\mathbf{J}_f^\top \mathbf{f} . \tag{3.9}$$

This is a descent direction for F since

$$\mathbf{h}_{\mathrm{GN}}^\top \mathbf{F}'(\mathbf{x}) = \mathbf{h}_{\mathrm{GN}}^\top (\mathbf{J}_f^\top \mathbf{f}) = -\mathbf{h}_{\mathrm{GN}}^\top (\mathbf{J}_f^\top \mathbf{J}_f)\mathbf{h}_{\mathrm{GN}} < 0 . \tag{3.10}$$

Thus, we can use $\mathbf{h}_{\mathrm{GN}}$ for $\mathbf{h}_{\mathrm{dh}}$ in Algorithm 2.4. The typical step is

$$\begin{aligned} &\text{Solve } (\mathbf{J}_f^\top \mathbf{J}_f)\mathbf{h}_{\mathrm{GN}} = -\mathbf{J}_f^\top \mathbf{f} \\ &\mathbf{x} := \mathbf{x} + \alpha \mathbf{h}_{\mathrm{GN}} \end{aligned} \tag{3.11}$$

where α is found by line search. The classical Gauss–Newton method uses $\alpha = 1$ in all steps. The method with line search can be shown to have guaranteed convergence, provided that

a) $\{\mathbf{x} \mid F(\mathbf{x}) \le F(\mathbf{x}_0)\}$ is bounded, and

b) the Jacobian $\mathbf{J}_f(\mathbf{x})$ has full rank in all steps.

In chapter 2 we saw that Newton's method for optimization has quadratic convergence. This is normally not the case with the Gauss-Newton method. To see this, we compare the search directions used in the two methods,

$$\mathbf{F}''(\mathbf{x})\mathbf{h}_{\mathrm{N}} = -\mathbf{F}'(\mathbf{x}) \quad \text{and} \quad \mathbf{L}''(\mathbf{h})\mathbf{h}_{\mathrm{GN}} = -\mathbf{L}'(\mathbf{0}) .$$

We already remarked at (3.8) that the two right-hand sides are identical, but from (3.4b) and (3.8) we see that the coefficient matrices differ:

$$\mathbf{F}''(\mathbf{x}) = \mathbf{L}''(\mathbf{h}) + \sum_{i=1}^{m} f_i(x)\mathbf{f}_i''(\mathbf{x}) . \tag{3.12}$$

Therefore, if $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$, then $\mathbf{L}''(\mathbf{h}) \simeq \mathbf{F}''(\mathbf{x})$ for $\mathbf{x}$ close to $\mathbf{x}^*$, and we get quadratic convergence also with the Gauss-Newton method. We can expect superlinear convergence if the functions $\{f_i\}$ have small curvatures or if the $\{|f_i(\mathbf{x}^*)|\}$ are small, but in general we must expect linear convergence. It is remarkable that the value of $F(\mathbf{x}^*)$ controls the convergence speed.

Example 3.3. Consider the simple problem with $n = 1$, $m = 2$ given by

$$\mathbf{f}(x) = \begin{bmatrix} x+1 \\ \lambda x^2 + x - 1 \end{bmatrix} . \quad F(x) = \tfrac{1}{2}(x+1)^2 + \tfrac{1}{2}(\lambda x^2 + x - 1)^2 .$$

It follows that

$$F'(x) = 2\lambda^2 x^3 + 3\lambda x^2 - 2(\lambda - 1)x ,$$

so $x = 0$ is a stationary point for F. Now,

$$F''(x) = 6\lambda^2 x^2 + 6\lambda x - 2(\lambda - 1) .$$

This shows that if $\lambda < 1$, then $\mathbf{F}''(0) > 0$, so $x = 0$ is a local minimizer – actually, it is the global minimizer.

The Jacobian matrix is

$$\mathbf{J}_f(x) = \begin{bmatrix} 1 \\ 2\lambda x + 1 \end{bmatrix} ,$$

and the classical Gauss-Newton method from x_k gives

$$x_{k+1} = x_k - \frac{\lambda^2 x_k^3 + 1.5\lambda x_k^2 - (\lambda - 1) x_k}{1 + \lambda x_k} .$$

Now, if $\lambda \neq 0$ and x_k is close to zero, then

$$x_{k+1} \;=\; x_k + (\lambda - 1) x_k (1 - \lambda x_k) + O(x_k^2) \;=\; \lambda x_k + O(x_k^2) .$$

Thus, if $|\lambda| < 1$, we have linear convergence. convergence, linear If $\lambda < -1$, then the classical Gauss-Newton method cannot find the minimizer. E.g. with $\lambda = -2$ and $x_0 = 0.1$ we get

k	x_k
0	0.1000
1	−0.2425
2	0.4046
3	−4.7724
4	44.2975
⋮	⋮

Finally, if $\lambda = 0$, then

$$x_{k+1} = x_k - x_k = 0 ,$$

i.e. we find the solution in one step. The reason is that in this case $\mathbf{f}$ is a linear function.

Example 3.4. For the data fitting problem from Example 1.1 the ith row of the Jacobian matrix is

$$\mathbf{J}_f(\mathbf{x})_{i,:} = \begin{bmatrix} -x_3 t_i e^{x_1 t_i} & -x_4 t_i e^{x_2 t_i} & -e^{x_1 t_i} & -e^{x_2 t_i} \end{bmatrix} .$$

If the problem is *consistent* (i.e. $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$), then the Gauss-Newton method with line search will have quadratic final convergence, provided that x_1^* is significantly different from x_2^*. If $x_1^* = x_2^*$, then $\operatorname{rank}(\mathbf{J}_f(\mathbf{x}^*)) \le 2$, and the Gauss-Newton method fails.

If one or more measurement errors are large, then $\mathbf{f}(\mathbf{x}^*)$ has some large components, and this may slow down the convergence.

In MATLAB we can give a very compact function for computing $\mathbf{f}$ and $\mathbf{J}_f$: Suppose that `x` holds the currrent iterate and that the $m \times 2$ array `ty` holds the coordinates of the data points. The following function returns `f` and `J` containing $\mathbf{f}(\mathbf{x})$ and $\mathbf{J}_f(\mathbf{x})$, respectively.

```
function  [f, J] = fitexp(x, ty)
  t = ty(:,1); y = ty(:,2);
  E = exp(t * [x(1), x(2)]);
  f = y - E*[x(3); x(4)];
  J = -[x(3)*t.*E(:,1), x(4)*t.*E(:,2), E];
```

Example 3.5. Consider the problem from Example 3.2, $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$ with $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$. If we use Newton-Raphson's method to solve this problem, the typical iteration step is

$$\text{Solve} \;\; \mathbf{J}_f(\mathbf{x})\mathbf{h}_{\mathrm{NR}} = -\mathbf{f}(\mathbf{x}); \quad \mathbf{x} := \mathbf{x} + \mathbf{h}_{\mathrm{NR}} .$$

The Gauss-Newton method applied to the minimization of $F(\mathbf{x}) = \frac{1}{2}\mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$ has the typical step

$$\text{Solve} \;\; (\mathbf{J}_f(\mathbf{x})^\top \mathbf{J}_f(\mathbf{x}))\mathbf{h}_{\mathrm{GN}} = -\mathbf{J}_f(\mathbf{x})^\top \mathbf{f}(\mathbf{x}); \quad \mathbf{x} := \mathbf{x} + \mathbf{h}_{\mathrm{GN}} .$$

Note, that $\mathbf{J}_f(\mathbf{x})$ is a square matrix, and we assume that it is nonsingular. Then $(\mathbf{J}_f(\mathbf{x})^\top)^{-1}$ exists, and it follws that $\mathbf{h}_{\mathrm{GN}} = \mathbf{h}_{\mathrm{NR}}$. Therefore, when applied to Powells problem from Example 3.2, the Gauss-Newton method will have the same troubles as discussed for Newton-Raphson's method in that example.

These examples show that the Gauss-Newton method may fail, both with and without a line search. Still, in many applications it gives quite good performance, though it normally only has linear convergence as opposed to the quadratic convergence from Newton's method with implemented second derivatives.

In Section 3.2 we give a method with superior global performance, and in Section 3.3 we give modifications to the method so that we achieve superlinear final convergence.

3.2. Marquardt's Method

In section 2.2 we suggested a hybrid method with better global performance than Newton's method. One problem with the latter is that if we are far from a local minimizer, the Hessian matrix may be indefinite or even negative difinite. Thus the Newton step $\mathbf{h}_{\rm N}$ is perhaps not a descent direction, and in that case it would be better to use the steepest descent direction $\mathbf{h}_{\rm sd}$.

In Section 3.1 we saw that the Gauss-Newton step $\mathbf{h}_{\rm GN}$ is well-defined only if $\mathbf{J}_f(\mathbf{x})$ has full rank. In that case $\mathbf{h}_{\rm GN}$ is a descent direction.

Both Newton's method and the Gauss-Newton method may suggest steps that are so long that the non-linearity of the components of $\mathbf{f}$ gives a value of F, which is larger than the one we are about to leave. The reason is that the linear models behind the two methods, (2.9) and (3.11), are good approximations only for small values of $\|\mathbf{h}\|$.

Levenberg (1944) and later Marquardt (1963) suggested a method where the step $\mathbf{h}_{\rm M}$ is computed by the following modification of the system (3.9) defining $\mathbf{h}_{\rm GN}$:

$$(\mathbf{J}_f^\top \mathbf{J}_f + \mu \mathbf{I})\mathbf{h}_{\rm M} = -\mathbf{g} \quad \text{with} \quad \mathbf{g} = \mathbf{J}_f^\top \mathbf{f} \text{ and } \mu \geq 0 \ . \tag{3.13}$$

Here, $\mathbf{J}_f = \mathbf{J}_f(\mathbf{x})$ and $\mathbf{f} = \mathbf{f}(\mathbf{x})$. The damping parameter μ has several effects:

a) For all $\mu > 0$ the coefficient matrix is positive definite, and this ensures that $\mathbf{h}_{\rm M}$ is a descent direction, cf. (3.10).

b) For large values of μ we get

$$\mathbf{h}_{\rm M} \simeq -\frac{1}{\mu}\mathbf{g} = -\frac{1}{\mu}\mathbf{F}'(\mathbf{x}) \ ,$$

i.e. a short step in the steepest descent direction.

c) If μ is very small, then $\mathbf{h}_{\rm M} \simeq \mathbf{h}_{\rm GN}$, which is a good step in the final stages of the iteration, when $\mathbf{x}$ is close to $\mathbf{x}^*$. If $F(\mathbf{x}^*) = 0$ (or very small), then we can get (almost) quadratic final convergence.

Thus, the damping parameter influences both the direction and the size of the step, and this leads us to make a method **without** a specific line search. The choice of initial μ-value should be related to the size of the elements in $\mathbf{A}_0 = \mathbf{J}_f(\mathbf{x}_0)^\top \mathbf{J}_f(\mathbf{x}_0)$, e.g. by letting

$$\mu_0 = \tau \cdot \max_i \ a_{ii}^{(0)} \ , \tag{3.14}$$

where τ is chosen by the user. During iteration the size of μ can be controlled by the *gain ratio*

$$\varrho = \frac{F(\mathbf{x}) - F(\mathbf{x}+\mathbf{h}_{\rm M})}{L(\mathbf{0}) - L(\mathbf{h}_{\rm M})} \ , \tag{3.15a}$$

where the denominator is the gain predicted by the linear model (3.7b),

$$\begin{aligned} L(\mathbf{0}) - L(\mathbf{h}_{\rm M}) &= -\mathbf{h}_{\rm M}^\top \mathbf{J}_f^\top \mathbf{f} - \tfrac{1}{2}\mathbf{h}_{\rm M}^\top \mathbf{J}_f^\top \mathbf{J}_f \mathbf{h}_{\rm M} \\ &= -\tfrac{1}{2}\mathbf{h}_{\rm M}^\top \Big(2\mathbf{g} + (\mathbf{J}_f^\top \mathbf{J}_f + \mu\mathbf{I} - \mu\mathbf{I})\mathbf{h}_{\rm M}\Big) \\ &= \tfrac{1}{2}\mathbf{h}_{\rm M}^\top(\mu \mathbf{h}_{\rm M} - \mathbf{g}) \ . \end{aligned} \tag{3.15b}$$

Note that both $\mathbf{h}_{\rm M}^\top \mathbf{h}_{\rm M}$ and $-\mathbf{h}_{\rm M}^\top \mathbf{g}$ are positive, so $L(\mathbf{0}) - L(\mathbf{h}_{\rm M})$ is guaranteed to be positive.

A large value of ϱ indicates that $L(\mathbf{h}_{\rm M})$ is a good approximation to $F(\mathbf{x}+\mathbf{h}_{\rm M})$, and we can decrease μ so that the next Marquardt step is closer to the Gauss-Newton step. If ϱ is small (maybe even negative), then $L(\mathbf{h}_{\rm M})$ is a poor approximation, and we should increase μ with the twofold aim of getting closer to the steepest descent direction **and** reducing the step length. These goals can be met in different ways, e.g. by using the following simple *updating strategy,*

$$\begin{array}{l} \texttt{if} \ \ \varrho > 0 \\ \quad \mathbf{x} := \mathbf{x} + \mathbf{h}_{\rm M} \\ \quad \mu := \mu * \max\{\tfrac{1}{3}, 1 - (2\varrho - 1)^3\}; \quad \nu := 2 \\ \texttt{else} \\ \quad \mu := \mu * \nu; \quad \nu := 2 * \nu \end{array} \tag{3.16}$$

The factor ν is initialized to $\nu = 2$. Note that $\mathbf{x}$ is updated only if $\varrho > 0 \;\Leftrightarrow\; F(\mathbf{x}+\mathbf{h}_{\mathrm{M}}) < F(\mathbf{x})$, i.e. if the descending condition (2.1) is satisfied, and that a series of consecutive failures results in rapidly increasing μ-values.

Example 3.6. The currently most widely used strategy has the form

$$\begin{array}{l} \texttt{if}\;\; \varrho > 0.75 \\ \quad \mu := \mu/3 \\ \texttt{if}\;\; \varrho < 0.25 \\ \quad \mu := \mu * 2 \\ \texttt{if}\;\; \varrho > 0 \\ \quad \mathbf{x} := \mathbf{x} + \mathbf{h}_{\mathrm{M}} \end{array} \tag{3.17}$$

This strategy was originally proposed by Marquardt (1963), and small changes in the thresholds 0.25 and 0.75 and in the factors 2 and $\frac{1}{3}$ can be seen. The two updating formulas are illustrated below.

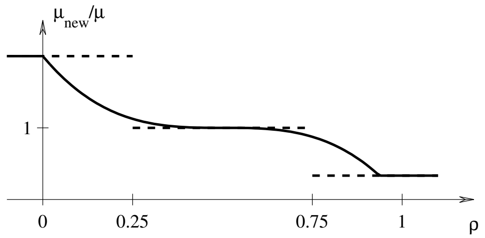


Figure 3.1. *Updating of μ by (3.16) with $\nu = 2$ (full line) Marquardt's strategy (dasheded line)*

The smoother change of μ for ϱ in the range $0 < \varrho < 1$ has a beneficial influence on the convergence, see Figure 3.2a-b below. Also, if $\varrho \le 0$ in consective steps, then (3.16) needs fewer steps to get μ sufficiently large. Extensive testing in Nielsen (1999) shows that generally (3.16) is significantly superior to (3.17).

The *stopping criteria* for the algorithm should reflect that at a global minimizer we have $\mathbf{F}'(\mathbf{x}^*) = \mathbf{g}(\mathbf{x}^*) = \mathbf{0}$, so we can use

$$\|\mathbf{g}\|_\infty \le \varepsilon_1 \;, \tag{3.18a}$$

where ε_1 is a small, positive number, chosen by the user. Another relevant criterion is to stop if the relative change in $\mathbf{x}$ is small,

$$\|\mathbf{x}_{\mathrm{new}} - \mathbf{x}\| \le \varepsilon_2 \|\mathbf{x}\| \;. \tag{3.18b}$$

Finally, to guard against an infinite loop, we need a "safety valve"

$$k \ge k_{\max} \;. \tag{3.18c}$$

Also ε_2 and $k_{\max}$ are chosen by the user.

The last two criteria come into effect e.g. if ε_1 is chosen so small that effects of rounding errors have large influence. This will typically reveal itself in a poor accordance between the actual gain in F and the gain predicted by the linear model (3.7b), and will result in μ being augmented in every step. Our strategy for augmenting μ implies that in this case μ grows fast, resulting in small $\|\mathbf{h}_{\mathrm{M}}\|$, and the process will be stopped by (3.18b).

The algorithm is summarized below.

As regards practical implementation, we do not need to store the complete $m \times n$ Jacobian matrix. Suppose e.g. that we compute it one row at a time, then we can build up the matrix $\mathbf{A} = \mathbf{J}_f(\mathbf{x})^\top \mathbf{J}_f(\mathbf{x})$ and vector $\mathbf{g} = \mathbf{J}_f(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$ by using the relations

$$\mathbf{A} = \sum_{i=1}^{m} \mathbf{J}_{i,:}^\top \mathbf{J}_{i,:} \;, \quad \mathbf{g} = \sum_{i=1}^{m} f_i(\mathbf{x}) \mathbf{J}_{i,:}^\top \;, \tag{3.19}$$

where $\mathbf{J}_{i,:}$ is the ith row in $\mathbf{J}_f(\mathbf{x})$, holding the derivatives of f_i.

Finally, remember that the Gauss-Newton step $\mathbf{h}_{\mathrm{GN}}$ minimizes the function $L(\mathbf{h})$,

$$\mathbf{h}_{\mathrm{GN}} = \mathrm{argmin}_{\mathbf{h}} \{L(\mathbf{h})\} \;.$$

Marquardt has shown that inside a ball of radius $\|\mathbf{h}_{\mathrm{M}}\|$ the Marquardt step $\mathbf{h}_{\mathrm{M}}$ minimizes L:

$$\mathbf{h}_{\mathrm{M}} = \mathrm{argmin}_{\|\mathbf{h}\| \le \|\mathbf{h}_{\mathrm{M}}\|} \{L(\mathbf{h})\} \;. \tag{3.20}$$

The proof is given in Appendix B.

Algorithm 3.21. Marquardt's Method [3)]

begin
  $k := 0; \quad \nu := 2; \quad \mathbf{x} := \mathbf{x}_0$
  $\mathbf{A} := \mathbf{J}_f(\mathbf{x})^\top \mathbf{J}_f(\mathbf{x}); \quad \mathbf{g} := \mathbf{J}_f(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$
  $found := (\|\mathbf{g}\|_\infty \le \varepsilon_1); \quad \mu := \tau * \max\{a_{ii}\}$
  while (**not** $found$) **and** $(k < k_{\max})$
    $k := k+1;$ Solve $(\mathbf{A} + \mu\mathbf{I})\mathbf{h}_{\mathrm{M}} = -\mathbf{g}$
    if $\|\mathbf{h}_{\mathrm{M}}\| \le \varepsilon_2 \|\mathbf{x}\|$
      $found :=$ **true**
    else
      $\mathbf{x}_{\mathrm{new}} := \mathbf{x} + \mathbf{h}_{\mathrm{M}}$
      $\varrho := (F(\mathbf{x}) - F(\mathbf{x}_{\mathrm{new}}))/(L(\mathbf{0}) - L(\mathbf{h}_{\mathrm{M}}))$ {cf. (3.15)}
      if $\varrho > 0$ {step acceptable}
        $\mathbf{x} := \mathbf{x}_{\mathrm{new}}$
        $\mathbf{A} := \mathbf{J}_f(\mathbf{x})^\top \mathbf{J}_f(\mathbf{x}); \quad \mathbf{g} := \mathbf{J}_f(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$
        $found := (\|\mathbf{g}\|_\infty \le \varepsilon_1)$
        $\mu := \mu * \max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}; \quad \nu := 2$
      else
        $\mu := \mu * \nu; \quad \nu := 2 * \nu$
end

Example 3.7. Comparing (3.9) and the normal equations (3.5) we see that $\mathbf{h}_{\mathrm{GN}}$ is simply the least squares solution to the linear problem $\mathbf{f}(\mathbf{x}) + \mathbf{J}_f(\mathbf{x})\mathbf{h} \simeq \mathbf{0}$. Similarly, the Marquardt equations (3.13) are the normal equations for the linear problem

$$\begin{bmatrix} \mathbf{f}(\mathbf{x}) \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{J}_f(\mathbf{x}) \\ \sqrt{\mu}\mathbf{I} \end{bmatrix} \mathbf{h} \simeq \mathbf{0} \ .$$

As mentioned in Example 3.1, the most accurate solution is found via orthogonal transformation. However, the solution $\mathbf{h}_{\mathrm{M}}$ is just a step in an iterative process, and needs not be computed very accurately, and since the solution via the normal equations is "cheaper", this method is normally employed.

[3)] The algorithm is sometimes called the *Levenberg-Marquardt Method*.

Example 3.8. We have used Algorithm 3.21 on the data fitting problem from Examples 1.1 and 3.4. Figure 1.1 indicates that both x_1 and x_2 are negative and that $M(\mathbf{x}^*, 0) \simeq 0$. These conditions are satisfied by $\mathbf{x}_0 = [-1,\ -2,\ 1,\ -1]^\top$. Further, we used $\tau = 10^{-3}$ in the expression (3.14) for μ_0 and the stopping criteria given by (3.18) with $\varepsilon_1 = \varepsilon_2 = 10^{-8}$, $k_{\max} = 200$. The algorithm stopped after 62 iteration steps with $\mathbf{x} \simeq [-4,\ -5,\ 4,\ -4]^\top$. The performance is illustrated below; note the logarithmic ordinate axis.

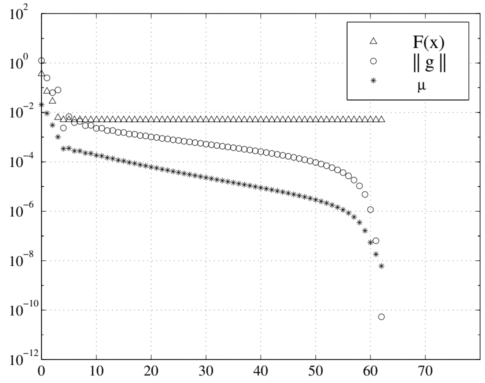


Figure 3.2a. *Marquardt's method applied to the fitting problem from Example 1.1*

This problem is not consistent, so we could expect linear final convergence. The last 7 iteration steps indicate a much better (superlinear) convergence. The explanation is, that the $\mathbf{f}_i''(\mathbf{x})$ are slowly varying functions of t_i, and the $f_i(\mathbf{x}^*)$ have "random" sign, so that the contributions to the "forgotten term" in (3.12) almost cancel out. Such a situation occurs in many data fitting applications.

For comparison, Figure 3.2b shows the performance with the updating strategy (3.17). From step 20 to step 68 we see that each decrease in μ is immediately followed by an increase, and the norm of the gradient has a rugged behaviour. This slows down the convergence, but the final stage is as in Figure 3.2a.

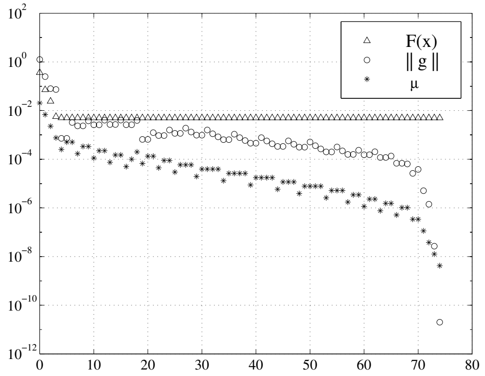


Figure 3.2b. *Performance with updating strategy (3.17)*

Example 3.9. Figure 3.3 illustrates the performance of Algorithm 3.21 applied to Powell's problem from Examples 3.2 and 3.5. The starting point is $\mathbf{x}_0 = [3,\ 1]^\top$, μ_0 given by $\tau = 1$ in (3.14), and we use $\varepsilon_1 = \varepsilon_2 = 10^{-15}$, $k_{\max} = 100$ in the stopping criteria (3.18).

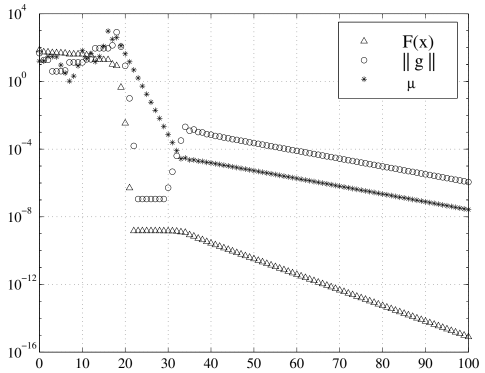


Figure 3.3. *Marquardt's method applied to Powell's problem*

The iteration seems to stall between steps 22 and 30. This as an effect of the (almost) singular Jacobian matrix. After that there seems to be linear convergence. The iteration is stopped by the "safety valve" at the point $\mathbf{x} = [$`-3.82e-08, -1.38e-03`$]^\top$. This is a better approximation

to $\mathbf{x}^* = \mathbf{0}$ than we found in Example 3.2, but still we want to be able to do better; see Examples 3.15 and 3.17.

3.3. A Hybrid Method: Marquardt and Quasi-Newton

In 1988 Madsen presented a hybrid method which combines Marquardt's method (quadratic convergence if $F(\mathbf{x}^*) = 0$, linear convergence otherwise) with a quasi-Newton method which gives superlinear convergence, even if $F(\mathbf{x}^*) \neq 0$. The iteration starts with a series of steps with the Marquardt method. If the performance indicates that $F(\mathbf{x}^*)$ is significantly nonzero, then we switch to the quasi-Newton method for better performance. It may happen that we get an indication that it is better to switch back to Marquardt's method, so there is also a mechanism for that.

The switch from Marquardt's method to the quasi-Newton method is made if

$$\|\mathbf{F}'(\mathbf{x})\|_\infty < 0.02 * F(\mathbf{x}) \tag{3.22}$$

in three consecutive, succesful iteration steps. This is interpreted as an indication that we are approaching an $\mathbf{x}^*$ with $\mathbf{F}'(\mathbf{x}^*) = \mathbf{0}$ and $F(\mathbf{x}^*)$ significantly nonzero. As discussed in connection with (3.12), this can lead to slow, linear convergence.

The quasi-Newton method is based on having an approximation $\mathbf{B}$ to the Hessian matrix $\mathbf{F}''(\mathbf{x})$ at the current iterate $\mathbf{x}$, and the step $\mathbf{h}_{\mathrm{qN}}$ is found by solving

$$\mathbf{B}\mathbf{h}_{\mathrm{qN}} = -\mathbf{F}'(\mathbf{x})\ , \tag{3.23}$$

which is an approximation to the Newton equation (2.10a).

The approximation $\mathbf{B}$ is updated by the BFGS strategy, cf. Section 5.10 in Frandsen et al. (1999): Every $\mathbf{B}$ in the series of approximation matrices is symmetric (as any $\mathbf{F}''(\mathbf{x})$) and positive definite. This ensures that $\mathbf{h}_{\mathrm{qN}}$ is "downhill", cf. (2.11). We start with the symmetric, positive definite matrix $\mathbf{B}_0 = \mathbf{I}$, and the BFGS update consists of a rank 2 matrix to be added to the current $\mathbf{B}$. Madsen (1988) uses the following version, advocated by Al-Baali and Fletcher (1985),

$$\begin{aligned} &\mathbf{h} := \mathbf{x}_{\text{new}} - \mathbf{x}; \quad \mathbf{y} := \mathbf{J}_{\text{new}}^{\top}\mathbf{J}_{\text{new}}\mathbf{h} + (\mathbf{J}_{\text{new}} - \mathbf{J})^{\top}\mathbf{f}(\mathbf{x}_{\text{new}}) \\ &\texttt{if} \;\; \mathbf{h}^{\top}\mathbf{y} > 0 \\ &\quad \mathbf{v} := \mathbf{B}\mathbf{h}; \quad \mathbf{B} := \mathbf{B} + \Big(\frac{\mathbf{1}}{\mathbf{h}^{\top}\mathbf{y}}\mathbf{y}\Big)\mathbf{y}^{\top} - \Big(\frac{\mathbf{1}}{\mathbf{h}^{\top}\mathbf{v}}\mathbf{v}\Big)\mathbf{v}^{\top} \end{aligned} \tag{3.24}$$

with $\mathbf{J} = \mathbf{J}_f(\mathbf{x})$, $\mathbf{J}_{\text{new}} = \mathbf{J}_f(\mathbf{x}_{\text{new}})$. As mentionened, the current $\mathbf{B}$ is positive definite, and it is changed only, if $\mathbf{h}^{\top}\mathbf{y} > 0$. In this case it can be shown that also the new $\mathbf{B}$ is positive definite.

The quasi-Newton method is not robust in the global stage of the iteration. Specifically it does not check a descending condition like (2.1). At the solution $\mathbf{x}^*$ we have $\mathbf{F}'(\mathbf{x}^*) = \mathbf{0}$, and good final convergence is indicated by rapidly decreasing values of $\|\mathbf{F}'(\mathbf{x})\|$. If these norm values do not decrease rapidly enough, then we switch back to the Marquardt method.

The algorithm is summarized below. It calls the auxiliary functions *M_Step* and *Q_Step*, implementing the two methods. We have the following remarks:

1° Initialization. μ_0 can be found by (3.14). The stopping criteria are given by (3.18).

2° The dots indicate that we also transfer current values of $\mathbf{f}$ and $\mathbf{J}_f$ etc. so that we do not have to recompute them for the same $\mathbf{x}$.

3° The currently best approximation found by Marquardt's method is saved. If the quasi-Newton method fails, we return to Marquardt's method and start it from $\mathbf{x}_{\text{best}}$.

4° Notice that both Marquardt and quasi-Newton steps contribute information for the approximation of the Hessian matrix.

Algorithm 3.25. A Hybrid Method

```
begin
  k := 0;   x := x0;   μ := μ0;   ν := 2;   B := I                          {1°}
  found := (‖F′(x)‖∞ ≤ ε1);   method := Marquardt
  while  (not found) and (k < kmax)
    k := k+1
    case method of
      Marquardt:
        [xnew, found, method, ...] := M_Step(x, ...)                       {2°}
        if  method = Quasi_Newton
          xbest := xnew                                                    {3°}
      Quasi_Newton:
        [xnew, found, method, ...] := Q_Step(x, B, xbest, ...)             {2°}
      Update B by (3.24);   x := xnew                                      {4°}
end
```

Function 3.25a. Marquardt Step

$[\mathbf{x}_{\text{new}}, \textit{found}, \textit{method}, \ldots] := \text{M\_Step}(\mathbf{x}, \ldots)$

```
begin
  xnew := x;   method := Marquardt
  Solve  (Jf(x)ᵀ Jf(x) + μI) hM = −F′(x)
  found := (‖hM‖ ≤ ε2‖x‖)
  if not found
    ϱ := (F(x) − F(x+hM))/(L(0) − L(hM))
    if  ϱ > 0
      xnew := x+hM;   found := (‖F′(xnew)‖∞ ≤ ε1)
      μ := μ * max{1/3, 1 − (2ϱ − 1)^3};   ν := 2;
      if  ‖F′(xnew)‖∞ < 0.02 * F(xnew)                                     {5°}
        count := count+1
        if  count = 3                                                      {6°}
          method := Quasi_Newton
      else
        count := 0
    else
      μ := μ * ν;   ν := 2 * ν;   count := 0
end
```

Function 3.25b Quasi-Newton Step

$[\mathbf{x}_{\text{new}}, \mathit{found}, \mathit{method}, \ldots] := \text{Q\_Step}(\mathbf{x}, \mathbf{B}, \mathbf{x}_{\text{best}}, \ldots)$

```
begin
  method := Quasi_Newton;   Solve  B h_qN = -F'(x_new)
  found := (||h_qN|| <= ε2||x||)
  if not found
    x_new := x + h_qN;      found := (||F'(x_new)||∞ <= ε1)
    if (not found) and (||F'(x_new)||∞ > 0.99 * ||F'(x)||∞)        {7°}
        method := Marquardt
        if F(x_new) > F(x_best)
          x_new := x_best;
end
```

We have the following remarks on the functions *M_step* and *Q_step*:

5° Indication that it might be time to switch method. The parameter *count* is initialized to zero at the start of Algorithm 3.25.

6° (3.22) was satisfied in three consecutive steps, all of which had $\varrho > 0$, i.e. $\mathbf{x}$ was changed.

7° The gradients do not decrease fast enough.

Example 3.10. Notice that in the updating formula (3.24) the computation of $\mathbf{y}$ involves the product $\mathbf{J}_f(\mathbf{x})^\top \mathbf{f}(\mathbf{x}_{\text{new}})$. This implies that we have to store the previous Jacobian matrix (or to recompute it if we want to exploit the possibilities discussed in connection with (3.19)). Instead, we could use

$$\mathbf{y} = \mathbf{F}'(\mathbf{x}_{\text{new}}) - \mathbf{F}'(\mathbf{x}) = \mathbf{g}_{\text{new}} - \mathbf{g}$$

in the updating formula, but Madsen (1988) found that (3.24) performs better.

Example 3.11. This hybrid method will not outperform Algorithm 3.21 on the problems discussed in Examples 3.8 and 3.9. In the latter case (see Figure 3.3) $F(\mathbf{x}) \to 0$, and the switching condition at remark 5° will

never be satisfied. In the former case, $F(\mathbf{x}^*)$ is significantly nonzero, but – as discussed in Example 3.8 – the simple Marquardt method has the desired superlinear final convergence.

To demonstrate the efficiency of Algorithm 3.25 we consider the modified *Rosenbrock problem*, cf. Example 5.5 in Frandsen et al. (1999), given by $\mathbf{f} : \mathbb{R}^2 \mapsto \mathbb{R}^3$,

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} 10(x_2 - x_1^2) \\ 1 - x_1 \\ \lambda \end{bmatrix},$$

where the parameter λ can be chosen. The minimizer of $F(\mathbf{x}) = \frac{1}{2}\mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$ is $\mathbf{x}^* = [1,\ 1]^\top$ with $F(\mathbf{x}^*) = \frac{1}{2}\lambda^2$.

Below we give results for Algorithms 3.21 and 3.25 for some values of λ. In all cases we use $\mathbf{x}_0 = [-1.2,\ 1]^\top$, the initial damping parameter μ_0 defined by $\tau = 10^{-3}$ in (3.14), and $(\varepsilon, k_{\max}) = (10^{-12}, 10^{-12}, 200)$ in the stopping criteria (3.18).

	Algorithm 3.21		Algorithm 3.25	
λ	its	$\|\mathbf{x}-\mathbf{x}^*\|$	its	$\|\mathbf{x}-\mathbf{x}^*\|$
0	18	8.84e-15	18	8.84e-15
10^{-5}	18	8.84e-15	18	8.84e-15
1	23	9.12e-09	19	3.61e-13
10^2	23	1.79e-06	22	1.20e-15
10^4	22	1.18e-04	22	1.20e-15

In the first two cases λ is too small to really influence the iterations, but for the larger λ-values we see that the hybrid method is much better than the simple Marquardt algorithm – especially with respect to the accuracy obtained. In Figure 3.4 we illustrate the performance of algorithms 3.21 and 3.25 in the case $\lambda = 10^4$.

With Marquardt's method all steps after no. 14 seem to fail to improve the objective function; μ increases rapidly, and the stopping criterion (3.18b) is satisfied at step no. 22.

With the hybrid method there are several attempts to use the quasi-Newton method, starting at step no. 5, 11 and 19. The last attempt ends with (3.18a) being satisfied.

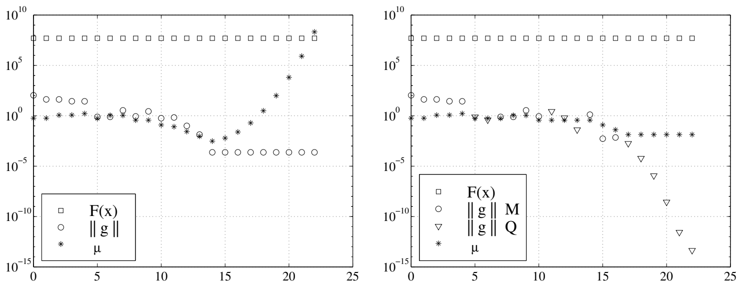


Figure 3.4. *Marquardt's method (left) and the hybrid method (right)*

3.4. A Secant Version of Marquardt's Method

The methods discussed in this booklet assume that the vector function $\mathbf{f}$ is differentiable, i.e. the Jacobian matrix

$$\mathbf{J}_f(\mathbf{x}) = \left[\frac{\partial f_i}{\partial x_j}\right]$$

exists. In many practical optimization problems it happens that we cannot give formulae for the elements in $\mathbf{J}_f$, e.g. $\mathbf{f}$ may be given by a "black box". The secant version of Marquardt's Method is intended for problems of this type.

The simplest remedy is to replace $\mathbf{J}_f(\mathbf{x})$ by a matrix $\mathbf{B}$ obtained by *numerical differentiation*: The $(i,j)^{th}$ element is approximated by the finite difference approximation

$$\frac{\partial f_i}{\partial x_j}(\mathbf{x}) \;\simeq\; \frac{f_i(\mathbf{x}+\delta\mathbf{e}_j) - f_i(\mathbf{x})}{\delta} \;\equiv\; b_{ij}\;, \qquad (3.26)$$

where $\mathbf{e}_j$ is the unit vector in the jth coordinate direction and δ is an appropriately small real number. With this strategy each iterate $\mathbf{x}$ needs $n+1$ evaluations of $\mathbf{f}$, and since δ is probably much smaller than the distance $\|\mathbf{x}-\mathbf{x}^*\|$, we do not get much more information on the **global** behavior of $\mathbf{f}$ than we would get from just evaluating $\mathbf{f}(\mathbf{x})$. We want better efficiency.

Example 3.12. Let $m=n=1$ and consider one nonlinear equation

$$f : \mathbb{R} \mapsto \mathbb{R}. \qquad \text{Find } \widehat{x} \text{ so that } f(\widehat{x}) = 0\;.$$

For this problem we can write the Newton-Raphson algorithm (3.6) in the form

$$\begin{array}{l} f(x+h) \;\simeq\; \ell(h) \;\equiv\; f(x) + f'(x)h \\ \text{solve the linear problem } \;\ell(h) = 0 \\ x_{\text{new}} := x + h \end{array} \qquad (3.27)$$

If we cannot implement $f'(x)$, then we can approximate it by $(f(x+\delta) - f(x))/\delta$ with δ chosen appropriately small. More generally, we can replace (3.27) by

$$\begin{array}{l} f(x+h) \;\simeq\; \lambda(h) \;\equiv\; f(x) + bh \quad \text{with } \; b \simeq f'(x) \\ \text{solve the linear problem } \;\lambda(h) = 0 \\ x_{\text{new}} := x + h \end{array} \qquad (3.28a)$$

Suppose that we already know x_{prev} and $f(x_{\text{prev}})$. Then we can fix the factor b (the approximation to $f'(x)$) by requiring that

$$f(x_{\text{prev}}) = \lambda(x_{\text{prev}} - x)\;. \qquad (3.28b)$$

This gives us $b = \big(f(x) - f(x_{\text{prev}})\big)/(x - x_{\text{prev}})$, and with this choice of b we recognize (3.28) as the *secant method*; see e.g. p. 29ff in Barker & Tingleff (1991). The main advantage of the secant method over an alternative finite difference approximation to Newton-Raphson's method is that we only need one function evaluation per iteration step instead of two. For a more thorough discussion of computational efficiency see p. 30f in Barker and Tingleff (1991).

Now, consider the linear model (3.7a) for $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$,

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) \;\simeq\; \boldsymbol{\ell}(\mathbf{h}) \;\equiv\; \mathbf{f}(\mathbf{x}) + \mathbf{J}_f(\mathbf{x})\mathbf{h}\;.$$

We will replace it by

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) \;\simeq\; \boldsymbol{\lambda}(\mathbf{h}) \;\equiv\; \mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{h}\;,$$

where $\mathbf{B}$ is the current approximation to $\mathbf{J}_f(\mathbf{x})$. In the next iteration step we need $\mathbf{B}_{\text{new}}$ so that

$$\mathbf{f}(\mathbf{x}_{\text{new}}+\mathbf{h}) \simeq \mathbf{f}(\mathbf{x}_{\text{new}}) + \mathbf{B}_{\text{new}}\mathbf{h} \ .$$

Especially, we want this model to hold with equality for $\mathbf{h} = \mathbf{x}-\mathbf{x}_{\text{new}}$, i.e.

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}_{\text{new}}) + \mathbf{B}_{\text{new}}(\mathbf{x}-\mathbf{x}_{\text{new}}) \ . \tag{3.29a}$$

This gives us m equations in the $m{\cdot}n$ unknown elements of $\mathbf{B}_{\text{new}}$, so we need more conditions. Broyden (1965) suggested to supplement (3.30a) with

$$\mathbf{B}_{\text{new}}\mathbf{v} = \mathbf{B}\mathbf{v} \quad \text{for all} \ \ \mathbf{v} \perp (\mathbf{x}-\mathbf{x}_{\text{new}}) \ . \tag{3.29b}$$

It is easy to verify that the conditions (3.29a–b) are satisfied by

Broyden's Rank One Update

$$\mathbf{B}_{\text{new}} = \mathbf{B} + \mathbf{u}\mathbf{h}^\top$$

where

$$\mathbf{h} = \mathbf{x}_{\text{new}} - \mathbf{x} \ , \quad \mathbf{u} = \frac{1}{\mathbf{h}^\top\mathbf{h}}(\mathbf{f}(\mathbf{x}_{\text{new}}) - \mathbf{f}(\mathbf{x}) - \mathbf{B}\mathbf{h}) \ . \tag{3.30}$$

Note that condition (3.29a) corresponds to the secant condition (3.28b) in the case $n=1$. We say that this approach is a *generalized secant method.*

A brief sketch of the central part of Algorithm 3.21 with this modification has the form

solve $(\mathbf{B}^\top\mathbf{B} + \mu\mathbf{I})\mathbf{h}_{\text{sM}} = -\mathbf{B}^\top\mathbf{f}(\mathbf{x})$
$\mathbf{x}_{\text{new}} := \mathbf{x} + \mathbf{h}_{\text{sM}}$
Update $\mathbf{B}$ by (3.30)
Update μ and $\mathbf{x}$ as in (3.16)

Powell has shown that if the set of vectors $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ converges to $\mathbf{x}^*$ and if the set of steps $\{\mathbf{h}_k \equiv \mathbf{x}_k-\mathbf{x}_{k-1}\}$ satisfy the condition that $\{\mathbf{h}_{k-n+1}, \ldots, \mathbf{h}_k\}$ are linearly independent (they span the whole of $\mathbb{R}^n$) for each $k \geq n$, then the set of approximations $\{\mathbf{B}_k\}$ converges to $\mathbf{J}_f(\mathbf{x}^*)$, irrespective of the choice of $\mathbf{B}_0$.

In practice, however, it often happens that the previous n steps do **not** span the whole of $\mathbb{R}^n$, and there is a risk that after some iteration steps the current $\mathbf{B}$ is such a poor approximation to the true Jacobian matrix, that $-\mathbf{B}^\top\mathbf{f}(\mathbf{x})$ is not even a downhill direction. In that case $\mathbf{x}$ will stay unchanged and μ is increased. The approximation $\mathbf{B}$ is changed, but may still be a poor approximation, leading to a further increase in μ, etc. Eventually the process is stopped by $\mathbf{h}_{\text{sM}}$ being so small that (3.18b) is satisfied, although $\mathbf{x}$ may be far from $\mathbf{x}^*$.

A number of strategies have been proposed to overcome this problem. A simple idea is occasionally to recompute $\mathbf{B}$ by finite differences. This idea is used in Algorithm 3.31 below. We have the following remarks:

1° Again the stopping criteria are given by (3.18), and the input is the starting vector $\mathbf{x}_0$ and the scalars τ, ε_1, ε_2, $k_{\max}$ and δ (for use in (3.30)). We also need K and ν_{th}, see 3°.

2° The expression for μ_0 is identical with (3.14) with $\mathbf{J}_f(\mathbf{x}_0)$ replaced by $\mathbf{B}$.

3° *updx*=`true` shows that $\mathbf{x}$ has moved since the previous difference approximation was computed. $\nu > \nu_{\text{th}}$ might be caused by $-\mathbf{B}^\top\mathbf{f}(\mathbf{x})$ not being downhill, and if K updates have been performed, it is time for a fresh approximation. The results given in the following example were computed with $\nu_{\text{th}} = 16$, $K = \max\{n, 10\}$.

4° Whereas the iterate $\mathbf{x}$ is updated only if the descending condition (2.1) is satisfied, the approximation $\mathbf{B}$ is normally updated in every step, see 5°. Therefore the approximate gradient $\mathbf{g}$ may change also when $\mathbf{f}(\mathbf{x})$ is unchanged.

Algorithm 3.31
Secant Version of Marquardt's Method

```
begin
  k := 0;   x := x_0;                                              {1°}
  Compute B by (3.26);   μ := τ * max{‖B_{:,j}‖^2};                 {2°}
  ν := 2;   updB := 0;   updx := false;   found := false
  while  (not found) and (k < k_max)
    if  (updx and ν > ν_th) or updB = K                             {3°}
      Compute B by (3.26);
      ν := 2;   updB = 0;   updx := false
    g := B^T f(x);   found := (‖g‖_∞ ≤ ε_1)                         {4°}
    if  not found
      k := k+1;   Solve  (B^T B + μI) h_sM = −g
      if  ‖h_sM‖ ≤ ε_2‖x‖
        found := true
      else
        x_new := x + h_sM;   dF = F(x) − F(x_new)
        if  updx or (dF > 0)                                        {5°}
          Update B by (3.30);   updB := updB+1
        ϱ := dF/(L(0) − L(h_M))                                     {6°}
        if  ϱ > 0
          x := x_new;   updx := true
          μ := μ * max{1/3, 1 − (2ϱ−1)^7};   ν := 2                 {7°}
        else
          μ := μ * ν;   ν := 2 * ν
end
```

5° If a new difference approximation does not immediately lead to a lower value of F (because μ is too small), then keep the current $\mathbf{B}$. Otherwise, update the approximation: also a "failing" step contributes information about the local behaviour of $\mathbf{f}$.

6° As in (3.15b) we can show that $L(\mathbf{0})-L(\mathbf{h}_{\mathrm{sM}}) = \frac{1}{2}\mathbf{h}_{\mathrm{sM}}^{\top}(\mu\mathbf{h}_{\mathrm{sM}}-\mathbf{g})$.

7° Compared with (3.16) we have changed the updating formula for μ. As in the Marquardt algorithm, ϱ is a measure of how well the linear model for $\mathbf{f}$ approximates the true behaviour. Now, however, it is influenced also by the diffence between the current $\mathbf{B}$ and the current $\mathbf{J}_f(\mathbf{x})$. It is not possible to distinguish between these two error contributions, so we use a more conservative updating of μ.

Example 3.13. We have used Algorithm 3.31 on the modified Rosenbrock problem from Example 3.11 with $\lambda=0$. If we use the same starting point and stopping criteria as in that example, and take $\delta=10^{-6}$ in the difference approximation (3.26), we find the solution after 26 iteration steps, involving a total of 33 evaluations of $\mathbf{f}(\mathbf{x})$. For comparison, the "true" Marquardt algorithm needs only 18 steps, implying a total of 19 evaluations of $\mathbf{f}(\mathbf{x})$ **and** $\mathbf{J}_f(\mathbf{x})$.

We have also used the secant algorithm on the data fitting problem from Examples 1.1, 3.4 and 3.8. With $\delta=10^{-6}$ and the same starting point and stopping criteria as in Example 3.8 the iteration was stopped by (3.18a) after 104 steps, involving a total of 149 evaluations of $\mathbf{f}(\mathbf{x})$. For comparison, Algorithm 3.21 needs 62 iteration steps.

These two problems indicate that Algorithm 3.31 is robust, but they also illustrate a general rule of thumb: If gradient information is available, it normally pays to use it.

In many practical applications the numbers m and n are large, but each of the functions $f_i(\mathbf{x})$ depends only on a few of the elements in $\mathbf{x}$. In that case most of the $\frac{\partial f_i}{\partial x_j}(\mathbf{x})$ are zero, and we say that $\mathbf{J}_f(\mathbf{x})$ is a *sparse matrix*. There are efficient methods exploiting sparsity in the solution of the Marquardt equation (3.13), see e.g. Nielsen (1997). In the updating formula (3.30), however, normally all elements in the vectors $\mathbf{h}$ and $\mathbf{u}$ are nonzero, so that $\mathbf{B}_{\mathrm{new}}$ will be a *dense matrix*. It is outside the scope of this booklet to discuss how to cope with this; we refer to Gill et al. (1984) and Toint (1987).

3.5. Powell's Dog Leg Method

As the Marquardt method, this method works with combinations of the Gauss-Newton and the steepest descent directions. Now, however controlled explicitly via the radius of a *trust region*, cf. Section 2.4 in Frandsen et al. (1999).

Given $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$. At the current iterate $\mathbf{x}$ we can compute the Gauss-Newton step $\mathbf{h}_{\mathrm{GN}}$ by solving

$$\left(\mathbf{J}_f(\mathbf{x})^\top \mathbf{J}_f(\mathbf{x})\right) \mathbf{h}_{\mathrm{GN}} = -\mathbf{J}_f(\mathbf{x})^\top \mathbf{f}(\mathbf{x}) \tag{3.32}$$

and the steepest descent direction

$$\mathbf{h}_{\mathrm{sd}} = -\mathbf{g} = -\mathbf{J}_f(\mathbf{x})^\top \mathbf{f}(\mathbf{x}) \; .$$

The latter is a direction, **not** a step, and to see how far we should go, we look at the linear model

$$\begin{aligned} &\mathbf{f}(\mathbf{x}+\alpha\mathbf{h}_{\mathrm{sd}}) \simeq \mathbf{f}(\mathbf{x}) + \alpha\mathbf{J}_f(\mathbf{x})\mathbf{h}_{\mathrm{sd}} \\ \Downarrow \\ &F(\mathbf{x}+\alpha\mathbf{h}_{\mathrm{sd}}) \simeq \tfrac{1}{2}\|\mathbf{f}(\mathbf{x}) + \alpha\mathbf{J}_f(\mathbf{x})\mathbf{h}_{\mathrm{sd}}\|^2 \\ &\qquad = F(\mathbf{x}) + \alpha\mathbf{h}_{\mathrm{sd}}^\top\mathbf{J}_f(\mathbf{x})^\top\mathbf{f}(\mathbf{x}) + \tfrac{1}{2}\alpha^2\|\mathbf{J}_f(\mathbf{x})\mathbf{h}_{\mathrm{sd}}\|^2 \; . \end{aligned}$$

This function of α is minimal for

$$\alpha = -\frac{\mathbf{h}_{\mathrm{sd}}^\top\mathbf{J}_f(\mathbf{x})^\top\mathbf{f}(\mathbf{x})}{\|\mathbf{J}_f(\mathbf{x})\mathbf{h}_{\mathrm{sd}}\|^2} = \frac{\|\mathbf{g}\|^2}{\|\mathbf{J}_f(\mathbf{x})\mathbf{g}\|^2} \; . \tag{3.33}$$

Now we have two candidates for the step to take from the current point $\mathbf{x}$: $\mathbf{a} = \alpha\mathbf{h}_{\mathrm{sd}}$ and $\mathbf{b} = \mathbf{h}_{\mathrm{GN}}$. Powell suggested to use the following strategy for choosing the step, when the trust region has radius Δ. The last case in the strategy is illustrated in Figure 3.5.

$$\begin{aligned} &\texttt{if}\ \ \|\mathbf{h}_{\mathrm{GN}}\| \le \Delta \\ &\quad \mathbf{h}_{\mathrm{dl}} := \mathbf{h}_{\mathrm{GN}} \\ &\texttt{elseif}\ \ \|\alpha\mathbf{h}_{\mathrm{sd}}\| \ge \Delta \\ &\quad \mathbf{h}_{\mathrm{dl}} := (\Delta/\|\mathbf{h}_{\mathrm{sd}}\|)\mathbf{h}_{\mathrm{sd}} \\ &\texttt{else} \\ &\quad \mathbf{h}_{\mathrm{dl}} := \alpha\mathbf{h}_{\mathrm{sd}} + \beta(\mathbf{h}_{\mathrm{GN}} - \alpha\mathbf{h}_{\mathrm{sd}}) \\ &\quad \text{with } \beta \text{ chosen so that } \|\mathbf{h}_{\mathrm{dl}}\| = \Delta \; . \end{aligned} \tag{3.34a}$$

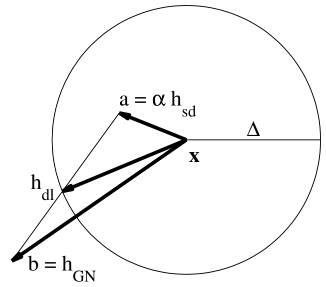


Figure 3.5. *Trust region and Dog Leg step*[4)]

With $\mathbf{a}$ and $\mathbf{b}$ as defined above, and $c = \mathbf{a}^\top(\mathbf{b}-\mathbf{a})$ we can write

$$\psi(\beta) \equiv \|\mathbf{a} + \beta(\mathbf{b}-\mathbf{a})\|^2 - \Delta^2 = \|\mathbf{b}-\mathbf{a}\|^2\beta^2 + 2c\beta + \|\mathbf{a}\|^2 - \Delta^2 \; .$$

We seek a root for this second order polynomial, and note that $\psi \to +\infty$ for $\beta \to -\infty$; $\psi(0) = \|\mathbf{a}\|^2 - \Delta^2 < 0$; $\psi(1) = \|\mathbf{h}_{\mathrm{GN}}\|^2 - \Delta^2 > 0$. Thus, ψ has one negative root and one root in $]0,1[$. We seek the latter, and the most accurate computation of it is given by

$$\begin{aligned} &\texttt{if}\ \ c \le 0 \\ &\quad \beta = \left(-c + \sqrt{c^2 + \|\mathbf{b}-\mathbf{a}\|^2(\Delta^2 - \|\mathbf{a}\|^2)}\right) / \|\mathbf{b}-\mathbf{a}\|^2 \\ &\texttt{else} \\ &\quad \beta = \left(\Delta^2 - \|\mathbf{a}\|^2\right) / \left(c + \sqrt{c^2 + \|\mathbf{b}-\mathbf{a}\|^2(\Delta^2 - \|\mathbf{a}\|^2)}\right) \end{aligned} \tag{3.34b}$$

---

4) The name *Dog Leg* is taken from golf: The fairway at a "dog leg hole" has a shape as the line from $\mathbf{x}$ (the tee point) via the end point of $\mathbf{a}$ to the end point of $\mathbf{h}_{\mathrm{dl}}$ (the hole). Powell is a keen golfer!

As in Marquardt's method we introduce the gain ratio

$$\varrho = (F(\mathbf{x}) - F(\mathbf{x}+\mathbf{h}_{\text{dl}})) / (L(\mathbf{0}) - L(\mathbf{h}_{\text{dl}})) \ ,$$

where L is the linear model

$$L(\mathbf{h}) = \tfrac{1}{2}\|\mathbf{f}(\mathbf{x}) + \mathbf{J}_f(\mathbf{x})\mathbf{h}\|^2 \ .$$

In Marquardt's method we used ϱ to control the size of the damping parameter. Here, we use it to control the radius Δ of the trust region. A large value of ϱ indicates that the linear model is good: We can increase Δ and thereby take longer steps, and they will be closer to the Gauss-Newton direction. If ϱ is small (maybe even negative) then we reduce Δ, implying smaller steps, closer to the steepest descent direction. Similar to (3.16) we can use

```
if  ϱ > 0
  x := x + h_dl
  Δ := Δ/ max{1/3, 1 − (2ϱ − 1)^3};    ν := 2                    (3.35)
else
  Δ := Δ/ν;    ν := 2 ∗ ν
```

For general least squares problems the Dog Leg method has the same disadvantages as Marquardt's method: the final convergence can be expected to be linear (and slow) if $F(\mathbf{x}^*) \neq 0$. This problem does not arise, however, in the case where we seek a root of $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$. Then $\mathbf{J}_f(\mathbf{x})$ is a square matrix, which we shall assume to be nonsingular, and (3.32) is equivalent with

$$\mathbf{J}_f(\mathbf{x})\mathbf{h}_{\text{GN}} = -\mathbf{f}(\mathbf{x}) \ , \tag{3.36}$$

which we recognize as the system of equations defining the Newton step. In the final stage of the Dog Leg algorithm we can expect $\|\mathbf{h}_{\text{GN}}\| \leq \Delta$, which means that we shall use Newton-Raphson's method, known to have quadratic final convergence – still provided that $\mathbf{J}_f(\mathbf{x}^*)$ is nonsingular.

Now we can formulate

Algorithm 3.37. Dog Leg Method
for systems of nonlinear equations

```
begin
  k := 0;   ν := 2;   x := x_0;   Δ := Δ_0                                    {1°}
  g := J_f(x)^T f(x);   found := (‖f(x)‖_∞ ≤ ε_3) or (‖g‖_∞ ≤ ε_1)          {2°}
  while  (not found) and (k < k_max)
    k := k+1;   Compute α by (3.33)
    h_sd := −αg;   Solve  J_f(x)h_GN = −f(x)
    Compute h_dl by (3.34)
    if  ‖h_dl‖ ≤ ε_2‖x‖
      found := true
    else
      x_new := x + h_dl
      ϱ := (F(x) − F(x_new))/(L(0) − L(h_dl))                                 {3°}
      if  ϱ > 0
        x := x_new;   g := J_f(x)^T f(x)
        found := (‖f(x)‖_∞ ≤ ε_3) or (‖g‖_∞ ≤ ε_1)
        Δ := Δ/ max{1/3, 1 − (2ϱ−1)^3};   ν := 2
      else
        Δ := Δ/ν;   ν := 2 ∗ ν;   found := (Δ ≤ ε_2‖x‖)                     {4°}
end
```

We have the following remarks.

1° Initialization. Δ_0 should be supplied by the user.

2° We use the stopping criteria (3.18) supplemented with $\|\mathbf{f}(\mathbf{x})\|_\infty \leq \varepsilon_3$, reflecting that $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$.

3° Corresponding to the three cases in (3.34a) we can show that

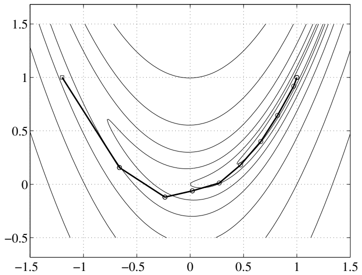
$$L(\mathbf{0}) - L(\mathbf{h}_{\text{dl}}) = \begin{cases} F(\mathbf{x}) & \text{if } \mathbf{h}_{\text{dl}} = \mathbf{h}_{\text{GN}} \\ \dfrac{\Delta(2\|\alpha\mathbf{g}\| - \Delta)}{2\alpha} & \text{if } \mathbf{h}_{\text{dl}} = \dfrac{-\Delta}{\|\mathbf{g}\|}\mathbf{g} \\ \tfrac{1}{2}\alpha(1-\beta)^2\|\mathbf{g}\|^2 + \beta(2-\beta)F(\mathbf{x}) & \text{otherwise} \end{cases}$$

4° Extra stopping criterion. If $\Delta \le \varepsilon_2 \|\mathbf{x}\|$, then (3.18b) will surely be satisfied in the next step.

Example 3.14. We have used Algorithm 3.37 on the Rosenbrock function $\mathbf{f} : \mathbb{R}^2 \mapsto \mathbb{R}^2$, given by

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} 10(x_2 - x_1^2) \\ 1 - x_1 \end{bmatrix} ,$$

cf. Example 3.11. With the starting point $\mathbf{x}_0 = [-1.2,\ 1]^\top$, $\Delta_0 = 1$ and the stopping criteria given by $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 10^{-12}$ we found the solution after 13 iteration steps. In Figure 3.6 we illustrate the convergence. In the left hand part we give the iterates and the level curves of F, and in the right hand part we show the trust region radius Δ and the parameter β in (3.34) as functions of iteration step number. $\beta = 1$ in the last three steps indicates that here the algorithm uses the "pure" Newton step $\mathbf{h}_{\mathrm{GN}}$.



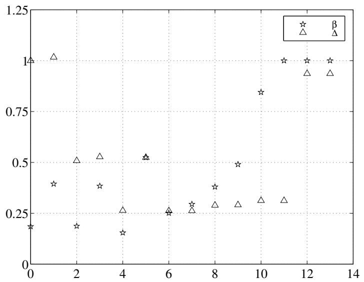


Figure 3.6. *Dog Leg method applied to Rosenbrock's function*

Example 3.15. In Figure 3.7 we illustrate the performance of the Dog Leg method applied to Powell's problem from Examples 3.2 and 3.9 with starting point $\mathbf{x}_0 = [3,\ 1]^\top$, $\Delta_0 = 1$ and the stopping criteria given by $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 10^{-20}$, $k_{\max} = 100$.

The criterion from Remark 2° on Algorithm 3.37 was satisfied after 42 iteration steps, returning $\mathbf{x} = [3.68{\cdot}10^{-38},\ -4.72{\cdot}10^{-11}]^\top$, which is quite a good approximation to $\mathbf{x}^* = \mathbf{0}$. After the 6th iteration step the algorithm uses "pure" Newton iteration, and Δ is increased in each step. As in Figure 3.3 we see that the convergence is linear (caused by

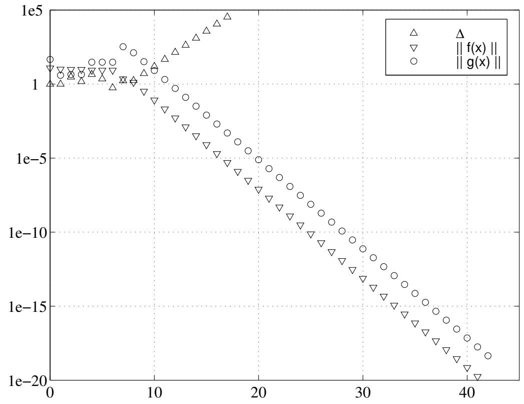


Figure 3.7. *Dog Leg method applied to Powell's problem*

the singular $\mathbf{J}_f(\mathbf{x}^*)$), but considerably faster than with the Marquardt method.

The above examples indicate that the Dog Leg method is good. It is presently considered as the best method for solving systems of nonlinear equations, but its real success is in the solution of such problems, when the Jacobian matrix is not available. As discussed in Section 3.4 we can compute $\mathbf{B} \simeq \mathbf{J}_f(\mathbf{x})$ via Broyden's updating formula (3.30)

$$\begin{aligned} &\mathbf{B}_{\text{new}} = \mathbf{B} + \Big(\frac{1}{\mathbf{h}^\top\mathbf{h}}(\mathbf{y} - \mathbf{B}\mathbf{h})\Big)\mathbf{h}^\top \\ &\text{where} \quad \mathbf{h} = \mathbf{x}_{\text{new}} - \mathbf{x} , \quad \mathbf{y} = \mathbf{f}(\mathbf{x}_{\text{new}}) - \mathbf{f}(\mathbf{x}) . \end{aligned} \tag{3.38a}$$

Broyden (1965) has also given a formula for updating an approximate inverse of the Jacobian matrix, $\mathbf{D} \simeq \mathbf{J}_f(\mathbf{x})^{-1}$. The formula is

$$\mathbf{D}_{\text{new}} = \mathbf{D} + \Big(\frac{1}{\mathbf{h}^\top\mathbf{D}\mathbf{y}}(\mathbf{h} - \mathbf{D}\mathbf{y})\Big)(\mathbf{h}^\top\mathbf{D}) , \tag{3.38b}$$

where $\mathbf{h}$ and $\mathbf{y}$ are defined in (3.38a).

With these matrices the steepest descent direction $\mathbf{h}_{\mathrm{sd}}$ and the Gauss-Newton step $\mathbf{h}_{\mathrm{GN}}$ (3.36) are approximated by

$$\mathbf{h}_{\mathrm{ssd}} = -\mathbf{B}^\top \mathbf{f}(\mathbf{x}) \quad \text{and} \quad \mathbf{h}_{\mathrm{sGN}} = -\mathbf{D}\mathbf{f}(\mathbf{x}) \; . \tag{3.39}$$

Algorithm 3.37 is easily modified to use these approximations. The initial $\mathbf{B} = \mathbf{B}_0$ can be found by the difference approximation (3.26), and $\mathbf{D}_0$ computed as $\mathbf{B}_0^{-1}$. It is easy to show that then the current $\mathbf{B}$ and $\mathbf{D}$ satisfy $\mathbf{BD} = \mathbf{I}$. The step parameter α is found by (3.33) with $\mathbf{J}_f(\mathbf{x})$ replaced by $\mathbf{B}$, and the predicted gain is still given by the expressions in remark 3° on Algorithm 3.37. As in remark 6° on Algorithm 3.31 we recommend the updating formula for the trust region radius to be changed to the more conservative $\Delta := \Delta / \max\{\frac{1}{3}, 1-(2\varrho-1)^7\}$. Also, we recommend occasional recomputation of $\mathbf{B}$ by finite differences, followed by $\mathbf{D} := \mathbf{B}^{-1}$.

Each update with (3.38) "costs" $10n^2$ flops[5] and the computation of the two step vectors by (3.39) plus the computation of α by (3.33) costs $6n^2$ flops. Thus, easch iteration step with the gradient-free version of the Dog Leg method costs about $16n^2$ flops plus evaluation of $\mathbf{f}(\mathbf{x}_{\mathrm{new}})$. For comparison, each step with Algorithm 3.37 costs about $\frac{2}{3}n^3 + 6n^2$ flops plus evaluation of $\mathbf{f}(\mathbf{x}_{\mathrm{new}})$ and $\mathbf{J}_f(\mathbf{x}_{\mathrm{new}})$. Thus, for large values of n the gradient-free version is cheaper per step. It should be mentioned, however, that the number of iteration steps often is considerably larger, and if the Jacobian matrix is available, then the gradient version is normally faster.

Example 3.16. We have used the gradient-free Dog Leg method on Rosenbrock's function from Example 3.14 with the same starting values and stopping criteria as given there and with $\delta = 10^{-6}$ in the difference approximation (3.26) used to find $\mathbf{B}_0$. The solution was found after 24 iteration steps, i.e. about twice as many as needed with the gradient version.

[5] One "flop" is a simple arithmetic operation between two floating point numbers.

3.6. Final Remarks

We have discussed a number of algorithms for solving nonlinear least squares problems. All of them 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 this booklet were computed in Matlab. The programs are available via

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

Finally, it should be mentioned that sometimes a reformulation of the problem can make it easier to solve. We shall illustrate this claim by examples, involving ideas that may be applicable also to **your** problem.

Example 3.17. In Powell's problem from Examples 3.2, 3.9 and 3.15 the variable x_2 occurs only as x_2^2. We can introduce new variables $\mathbf{z} = [x_1, \; x_2^2]^\top$, and the problem takes the form: Find $\mathbf{z}^* \in \mathbb{R}^2$ such that $\mathbf{f}(\mathbf{z}^*) = \mathbf{0}$, where

$$\mathbf{f}(\mathbf{z}) = \begin{bmatrix} z_1 \\ \frac{10z_1}{z_1+0.1} + 2z_2 \end{bmatrix} \qquad \text{with} \qquad \mathbf{J}_f(\mathbf{z}) = \begin{bmatrix} 1 & 0 \\ (z_1+0.1)^{-2} & 2 \end{bmatrix} .$$

This Jacobian matrix is nonsingular for all $\mathbf{z}$. Marquardt's algorithm 3.21 with starting point $\mathbf{z}_0 = [3, \; 1]^\top$, $\tau = 10^{-16}$ and $\varepsilon_1 = \varepsilon_2 = 10^{-15}$ in the stopping criteria (3.18) stops after 3 steps with $\mathbf{z} \simeq [$`-3.77e-26`, `-2.83e-24`$]^\top$. This is a good approximation to $\mathbf{z}^* = \mathbf{0}$.

Example 3.18. The data fitting problem from Examples 1.1, 3.4 and 3.8 can be reformulated to have only two parameters, x_1 and x_2: We can write the model in the form

$$M(\mathbf{x}, t) = c_1 e^{x_1 t} + c_2 e^{x_2 t} \; ,$$

where, for given $\mathbf{x}$, the vector $\mathbf{c} = \mathbf{c}(\mathbf{x}) \in \mathbb{R}^2$ is found as the least squares solution to the linear problem

$$\mathbf{Ec} \simeq \mathbf{y}$$

with $\mathbf{E} = \mathbf{E}(\mathbf{x}) \in \mathbb{R}^{m\times 2}$ given by the rows $(\mathbf{E})_{i,:} = [e^{x_1 t_i} \quad e^{x_2 t_i}]$. As in Example 1.1 the function $\mathbf{f}$ is defined by $f_i(\mathbf{x}) = y_i - M(\mathbf{x}, t_i)$, leading to

$$\mathbf{f}(\mathbf{x}) = \mathbf{y} - \mathbf{E}(\mathbf{x})\mathbf{c}(\mathbf{x}) \ .$$

It can be shown that the Jacobian matrix is

$$\mathbf{J}_f = -\mathbf{E}\mathbf{G} - \mathbf{H}[\mathbf{c}] \ ,$$

where, for any vector $\mathbf{u}$ we define the diagonal matrix $[\mathbf{u}] = \mathrm{diag}(\mathbf{u})$, and

$$\mathbf{H} = [\mathbf{t}]\mathbf{E}, \quad \mathbf{G} = (\mathbf{E}^\top\mathbf{E})^{-1}\left([\mathbf{H}^\top\mathbf{f}] - \mathbf{H}^\top\mathbf{E}[\mathbf{c}]\right) \ .$$

Marquardt's algorithm with the same poor starting guess as in Example 3.8, $\mathbf{x}_0 = [-1, \ -2]^\top$, $\tau = 1$ and $\varepsilon_1 = \varepsilon_2 = 10^{-8}$ finds the solution $\mathbf{x} \simeq [-4, \ -5]^\top$ after 9 iteration steps; about $\frac{1}{7}$ of the number of steps needed with the 4-parameter model.

Example 3.19. The final example illustrates a frequent difficulty with least squares problems: Normally the algorithms work best when the problem is scaled so that all the (nonzero) $|x_j^*|$ are of the same order of magnitude.

Consider the socalled *Meyer's problem*

$$f_i(\mathbf{x}) = y_i - x_1 \exp\left(\frac{x_2}{t_i + x_3}\right) \ , \quad i = 1, \ldots, 16 \ ,$$

with $t_i = 45 + 5i$ and

i	y_i	i	y_i	i	y_i
1	34780	7	11540	12	5147
2	28610	8	9744	13	4427
3	23650	9	8261	14	3820
4	19630	10	7030	15	3307
5	16370	11	6005	16	2872
6	13720				

The minimizer is $\mathbf{x}^* \simeq \left[\, 5.61\cdot 10^{-3} \quad 6.18\cdot 10^{3} \quad 3.45\cdot 10^{2} \,\right]^\top$ with $F(\mathbf{x}^*) \simeq 43.97$.

An alternative formulation is

$$\phi_i(\mathbf{x}) = 10^{-3} y_i - z_1 \exp\left(\frac{10 z_2}{u_i + z_3} - 13\right) \ , \quad i = 1, \ldots, 16 \ ,$$

with $u_i = 0.45 + 0.05i$. The reformulation corresponds to $\mathbf{z} = \left[\, 10^{-3} e^{13} x_1 \quad 10^{-3} x_2 \quad 10^{-2} x_3 \,\right]^\top$, and the minimizer is $\mathbf{z}^* \simeq \left[\, 2.48 \quad 6.18 \quad 3.45 \,\right]^\top$ with $\Phi(\mathbf{x}^*) \simeq 4.397\cdot 10^{-5}$.

If we use Algorithm 3.21 with $\tau = 1$, $\varepsilon_1 = \varepsilon_2 = 10^{-12}$ and the equivalent starting vectors

$$\mathbf{x}_0 = \left[\, 2\cdot 10^{-2} \quad 4\cdot 10^{3} \quad 2.5\cdot 10^{2} \,\right]^\top, \qquad \mathbf{z}_0 = \left[\, 8.85 \quad 4 \quad 2.5 \,\right]^\top \ ,$$

then the iteration is stopped by (3.18b) after 182 iteration steps with the first formulation, 97 steps with the well-scaled reformulation.

Appendix

A. Symmetric, Positive Definite Matrices

The 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

$$\begin{aligned} &\textit{positive definite} \quad \Leftrightarrow \quad \mathbf{x}^\top \mathbf{A}\, \mathbf{x} > 0 \ \text{ for all } \mathbf{x} \in \mathbb{R}^n,\ \mathbf{x} \neq \mathbf{0} \\ &\textit{positive semidefinite} \quad \Leftrightarrow \quad \mathbf{x}^\top \mathbf{A}\, \mathbf{x} \geq 0 \ \text{ for all } \mathbf{x} \in \mathbb{R}^n,\ \mathbf{x} \neq \mathbf{0} \end{aligned} \tag{A.1}$$

Some useful properties of such matrices are listed in Theorem A below. The proof can be found by combining theorems in almost any textbooks on linear algebra and on numerical linear algebra. At the end of this appendix we give some practical implications of the theorem.

Now, let $\mathbf{J} \in \mathbb{R}^{m\times n}$ be given, and let

$$\mathbf{A} = \mathbf{J}^\top \mathbf{J} .$$

Then $\mathbf{A}^\top = \mathbf{J}^\top (\mathbf{J}^\top)^\top = \mathbf{A}$, i.e. $\mathbf{A}$ is symmetric. Further, for any nonzero $\mathbf{x} \in \mathbb{R}^n$ let $\mathbf{y} = \mathbf{J}\mathbf{x}$. Then

$$\mathbf{x}^\top \mathbf{A}\mathbf{x} = \mathbf{x}^\top \mathbf{J}^\top \mathbf{J}\mathbf{x} = \mathbf{y}^\top \mathbf{y} \geq 0 ,$$

showing that $\mathbf{A}$ is positive semidefinite. If $m \geq n$ and the columns in $\mathbf{J}$ are linearly independent, then $\mathbf{x} \neq \mathbf{0} \Rightarrow \mathbf{y} \neq \mathbf{0}$ and $\mathbf{y}^\top \mathbf{y} > 0$. Thus, in this case $\mathbf{A}$ is positive definite.

From (A.2) follows immediately that

$$(\mathbf{A} + \mu\mathbf{I})\mathbf{v}_j = (\lambda_j + \mu)\mathbf{v}_j , \quad j = 1, \ldots, n$$

Theorem A

Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be symmetric and let $\mathbf{A} = \mathbf{L}\mathbf{U}$, where $\mathbf{L}$ is a unit lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix. Further, let $\{(\lambda_j, \mathbf{v}_j)\}_{j=1}^n$ denote the eigensolutions of $\mathbf{A}$, i.e.

$$\mathbf{A}\mathbf{v}_j = \lambda_j \mathbf{v}_j , \quad j = 1, \ldots, n . \tag{A.2}$$

Then

1° The eigenvalues are real, $\lambda_j \in \mathbb{R}$, and the eigenvectors $\{\mathbf{v}_j\}$ form an orthonormal basis of $\mathbb{R}^n$.

2° The following statements are equivalent

a) $\mathbf{A}$ is positive definite (positive semidefinite)

b) All $\lambda_j > 0$ ($\lambda_j \geq 0$)

c) All $u_{ii} > 0$ ($u_{ii} \geq 0$) .

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

3° The LU-factorization is numerically stable.

4° $\mathbf{U} = \mathbf{D}\mathbf{L}^\top$ with $\mathbf{D} = \mathrm{diag}(u_{ii})$.

5° $\mathbf{A} = \mathbf{C}\mathbf{C}^\top$, the *Cholesky factorization.* $\mathbf{C} \in \mathbb{R}^{n\times n}$ is lower triangular.

for any $\mu \in \mathbb{R}$. Combining this with 2° in Theorem A we see that if $\mathbf{A}$ is symmetric and positive semidefinite and $\mu > 0$, then the matrix $\mathbf{A}+\mu\mathbf{I}$ is also symmetric and it is guaranteed to be positive definite.

The *condition number* of a symmetric matrix $\mathbf{A}$ is

$$\kappa_2(\mathbf{A}) = \max\{|\lambda_j|\} / \min\{|\lambda_j|\} .$$

If $\mathbf{A}$ is positive (semi)definite and $\mu > 0$, then

$$\kappa_2(\mathbf{A}+\mu\mathbf{I}) = \frac{\max\{\lambda_j\} + \mu}{\min\{\lambda_j\} + \mu} \leq \frac{\max\{\lambda_j\} + \mu}{\mu} ,$$

and this is a decreasing function of μ.

Finally, some remarks on Theorem A and practical details: 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. Also note that points 4°–5° give the following relation between the LU- and the Cholesky-factorization

$$\mathbf{A} = \mathbf{L}\,\mathbf{U} = \mathbf{L}\mathbf{D}\mathbf{L}^\top = \mathbf{C}\mathbf{C}^\top \ ,$$

showing that

$$\mathbf{C} = \mathbf{L}\mathbf{D}^{1/2} \ , \quad \text{with} \quad \mathbf{D}^{1/2} = \mathrm{diag}(\sqrt{u_{ii}}) \ .$$

The Cholesky factorization can be computed directly (i.e. without the intermediate results $\mathbf{L}$ and $\mathbf{U}$) by the following algorithm, that includes a test for positive definitenes.

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
        c_ik := (a_ik − Σ_{j=1}^{k−1} c_ij c_kj) / c_kk    {subdiagonal elements}
    else
      posdef := false
end
```

The "cost" of this algorithm is about $\frac{1}{3}n^3$ flops. Once $\mathbf{C}$ is computed, the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ can be solved by forward and back substitution in

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

respectively. Each of these steps costs about n^2 flops.

B. Proof of (3.20)

We introduce the function

$$G(\mathbf{h}) = L(\mathbf{h}) + \tfrac{1}{2}\mu\mathbf{h}^\top\mathbf{h} \quad \text{with} \quad \mu > 0 \ .$$

The gradient of this is the linear function

$$\begin{aligned} \mathbf{G}'(\mathbf{h}) &= \mathbf{L}'(\mathbf{h}) + \mu\mathbf{h} \\ &= \mathbf{J}_f^\top\mathbf{f} + (\mathbf{J}_f^\top\mathbf{J}_f + \mu\mathbf{I})\mathbf{h} \ , \end{aligned}$$

where $\mathbf{J}_f = \mathbf{J}_f(\mathbf{x})$, $\mathbf{f} = \mathbf{f}(\mathbf{x})$, and we have used (3.8) in the reformulation. According to Appendix A the matrix $\mathbf{J}_f^\top\mathbf{J}_f + \mu\mathbf{I}$ is positive definite, so the linear system of equations $\mathbf{G}'(\mathbf{h}) = \mathbf{0}$ has a unique solution, and this is the minimizer for G. By comparison with (3.13) we see that this minimizer is $\mathbf{h}_{\mathrm{M}}$.

Now, let

$$\mathbf{h}_{\mathrm{m}} = \mathrm{argmin}_{\|\mathbf{h}\| \le \|\mathbf{h}_{\mathrm{M}}\|}\{L(\mathbf{h})\} \ .$$

Then $L(\mathbf{h}_{\mathrm{m}}) \le L(\mathbf{h}_{\mathrm{M}})$ and $\mathbf{h}_{\mathrm{m}}^\top\mathbf{h}_{\mathrm{m}} \le \mathbf{h}_{\mathrm{M}}^\top\mathbf{h}_{\mathrm{M}}$, and

$$\begin{aligned} G(\mathbf{h}_{\mathrm{m}}) &= L(\mathbf{h}_{\mathrm{m}}) + \mu\mathbf{h}_{\mathrm{m}}^\top\mathbf{h}_{\mathrm{m}} \\ &\le L(\mathbf{h}_{\mathrm{M}}) + \mu\mathbf{h}_{\mathrm{M}}^\top\mathbf{h}_{\mathrm{M}} = G(\mathbf{h}_{\mathrm{M}}) \ . \end{aligned}$$

However, $\mathbf{h}_{\mathrm{M}}$ is the unique minimizer of G, so $\mathbf{h}_{\mathrm{m}} = \mathbf{h}_{\mathrm{M}}$. □

References

1. M. Al-Baali & R. Fletcher (1985): "Variational Methods for Non-Linear Least-Squares". J. Opl. Res. Soc. **36**, No. 5, pp 405–421.
2. V.A. Barker & O. Tingleff (1991): "Numerisk Løsning af Ikke-Lineære Ligninger" (in Danish). Hæfte 57, IMM, DTU.
3. C.G. Broyden (1965): "A Class of Methods for Solving Nonlinear Simultaneous Equations". Maths. Comp. **19**, pp 577–593.
4. J.E. Dennis, Jr. & R.B. Schnabel (1983): "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice Hall.
5. P.E. Frandsen, K. Jonasson, H.B. Nielsen & O. Tingleff (1999): "Unconstrained Optimization", IMM, DTU. Available at http://www.imm.dtu.dk/~hbn/publ/H69.ps
6. P.E. Gill, W. Murray, M.A. Saunders & M.H. Wright (1984): "Sparse Matrix Methods in Optimization", SIAM J.S.S.C. **5**, pp 562–589.
7. G. Golub & C.F. van Loan (1989): "Matrix Computations", John Hopkins Univ. Press.
8. P. Hegelund, K. Madsen & P.C. Hansen (1991): "Robust C Functions for Non-Linear Optimization". Institute for Numerical Analysis (now part of IMM), DTU. Report NI-91-03.
9. K. Levenberg (1944): "A Method for the Solution of Certain Problems in Least Squares. Quart. Appl. Math. **2**, pp 164–168.
10. K. Madsen (1988): "A Combined Gauss-Newton and Quasi-Newton Method for Non-Linear Least Squares". Institute for Numerical Analysis (now part of IMM), DTU. Report NI-88-10.
11. K. Madsen & O. Tingleff (1990): "Robust Subroutines for Non-Linear Optimization". Institute for Numerical Analysis (now part of IMM), DTU. Report NI-90-06.
12. K. Madsen, H.B. Nielsen & O. Tingleff (1999): "Optimization with Constraints". IMM, DTU. Available at http://www.imm.dtu.dk/~hbn/publ/H40.ps.Z
13. D. Marquardt (1963): "An Algorithm for Least Squares Estimation on Nonlinear Parameters". SIAM J. Appl. Math. **11**, pp 431–441.
14. H.B. Nielsen (1996): "Numerisk Lineær Algebra" (in Danish), IMM, DTU
15. H.B. Nielsen (1997): "Direct Methods for Sparse Matrices". IMM, DTU. Available at http://www.imm.dtu.dk/~hbn/publ/Nsparse.ps.Z
16. 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
17. M.J.D. Powell (1970): "A Hybrid Method for Non-Linear Equations". In P. Rabinowitz (ed): *Numerical Methods for Non-Linear Algebraic Equations*, Gordon & Breach. pp 87ff.
18. P.L. Toint (1987): "On Large Scale Nonlinear Least Squares Calculations". SIAM J.S.S.C. **8**, pp 416–435.

INDEX