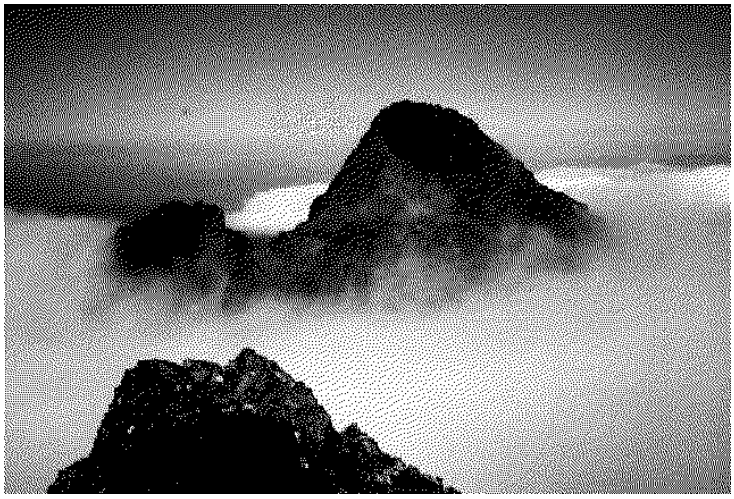
Kaj Madsen  Hans Bruun Nielsen

## Introduction to

# Optimization and Data Fitting



August 2010

DTU Informatics – IMM
Richard Petersens Plads
DTU – building 321
DK-2800 Kgs. Lyngby

Phone: +45 4525 3351
www.imm.dtu.dk

# Preface

The major part of this book is based on lecture notes for the DTU course *02611 Optimization and Data Fitting*. The notes were developed by Kaj Madsen and Hans Bruun Nielsen together with their former IMM colleague Ole Tingleff, Post Doc Kristian Jonasson and PhD student Poul Erik Frandsen.

We are grateful to the co-authors of the lecture notes and to the many students in Courses 02611 and 02610, who gave us constructive criticism that helped improve the presentation.

The reader is assumed to have mathematical skills corresponding to introductory university courses in calculus and linear algebra. Some of the nontrivial, but necessary concepts are reviewed in Appendix A.

The book is organized as follows: Chapter 1 introduces some basic concepts, that are used throughout the book. Chapters 2 – 4 deal with unconstrained optimization of general functions, with Chapter 3 concentrating on Newton–type methods, and Chapter 4 is a short introduction to direct search methods.

Chapters 5 – 7 discuss data fitting. In Chapters 5 – 6 we look at least squares methods for linear and nonlinear fitting models, including an introduction to statistical aspects, and special methods for fitting with polynomials and cubic splines. Chapter 7 deals with fitting in the $L_\infty$ norm and robust fitting by means of the $L_1$ norm and the Huber estimator.

Implementations of some of the algorithms discussed in the book are available as follows,
Fortran and C: http://www2.imm.dtu.dk/~km/F-pak.html
MATLAB: http://www2.imm.dtu.dk/~hbn/immoptibox

Kaj Madsen, Hans Bruun Nielsen

DTU Informatics – IMM
Technical University of Denmark

# Contents

# Chapter 1

# Introduction

Optimization plays an important role in many branches of science and applications: economics, operations research, network analysis, optimal design of mechanical or electrical systems, to mention a few. In this book we shall discuss numerical methods for the solution of so-called *continuous optimization*, where the problem is formulated in terms of a real function of several real variables, and we want to find a set of arguments that give a minimal function value. Such a problem may arise from parameter estimation, and we take data fitting as a special case of this.

**Definition 1.1. The optimization problem.** Let $f : \mathbb{R}^n \mapsto \mathbb{R}$,

$$\text{Find } \hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in \mathbb{R}^n}{\operatorname{argmin}} f(\boldsymbol{x}) \ .$$

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

The definition in terms of minimization is not en essential restriction. A *maximizer* of a function $f$ is clearly a minimizer for $-f$.

**Example 1.1.** In this example we consider functions of one variable. The function

$$f(x) = (x - \hat{x})^2$$

has a unique minimizer, $\hat{x}$, see Figure 1.1.

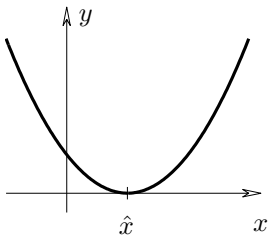



***Figure 1.1.*** $y = (x - \hat{x})^2$.
*One minimizer.*

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

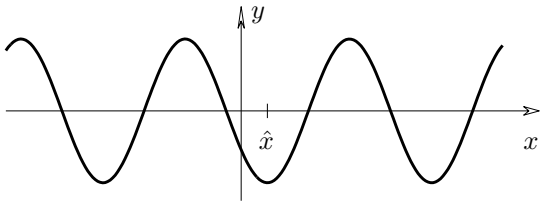



***Figure 1.2.*** $y = -2\cos(x - \hat{x})$. *Many minimizers.*

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

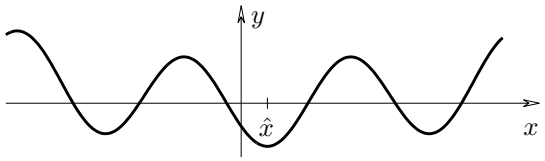



***Figure 1.3.*** $y = 0.015(x - \hat{x})^2 - 2\cos(x - \hat{x})$.
*One global minimizer and many local minimizers.*

■

The ideal situation for optimization computations is that the objective function has a unique minimizer, the *global minimizer*. In some cases $f$ has several (or even infinitely many) minimizers. In such problems it may be sufficient to find one of these minimizers. In many applications the objective function has a global minimizer and several local minimizers. It is difficult to develop methods which can find the global minimizer with certainty in this situation. See, for instance, [8] about a method for global optimization.

In the other parts of the book we concentrate on methods that can find a local minimizer for the objective function. When such a point has been found, we do not know whether it is a global minimizer or one

of the local minimizers; we cannot even be sure that the algorithm will find the local minimizer closest to the starting point. In order to explore several local minimizers we can try several runs with different starting points, or better still examine intermediate results produced by a global minimizer.

We end this section with an example which demonstrates that optimization methods based on too primitive ideas may be dangerous.

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

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

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

"Make a series of iterations from a starting point $\boldsymbol{x}_0$. In each iteration keep one of the variables fixed and seek a value of the other variable so as to minimize the $f$-value". Figure 1.4 shows the *level curves* or *contours* of $f$, ie curves along which $f$ is constant. We also show the first few iterations.

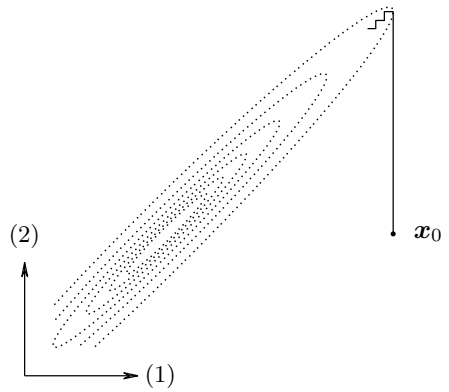



***Figure 1.4.*** *The method of Alternating Variables fails to determine the minimizer of a quadratic.*

After some iterations the steps begin to decrease rapidly in size. They can become so small that they do not change the $\boldsymbol{x}$-values, because they are represented with finite precision in the computer, and the progress stops completely, maybe far away from the solution. We say that the iterations are caught in *Stiefel's cage*.

The "method" is called the *method of alternating variables* or *coordinate search* and it is a classical example of a method that should be avoided.

■

## 1.1. Basic concepts

We need to be able to talk about the "length" of a vector $\boldsymbol{v} \in \mathbb{R}^n$, for instance the difference $\boldsymbol{x} - \hat{\boldsymbol{x}}$ between the solution $\hat{\boldsymbol{x}}$ and an approximation $\boldsymbol{x}$ to it. For that purpose we use a *norm* $\|\boldsymbol{v}\|$. We make use of the following three norms,

$$\begin{aligned} \|\boldsymbol{v}\|_1 &= |v_1| + \cdots |v_n| \quad , \\ \|\boldsymbol{v}\|_2 &= \left(|v_1|^2 + \cdots |v_n|^2\right)^{1/2} \quad , \\ \|\boldsymbol{v}\|_\infty &= \max\{|v_1|, \ldots |v_n|\} \quad . \end{aligned} \tag{1.2}$$

Unless otherwise specified $\|\boldsymbol{v}\|$ without index on the norm is used as shorthand for the 2-norm: $\|\boldsymbol{v}\| = \|\boldsymbol{v}\|_2$.

Next, we define the important concept of a *local minimizer* for the function $f$: This is an argument vector that gives the smallest function value inside a certain region, defined by $\varepsilon$:

**Definition 1.3. Local minimizer.**
$\hat{\boldsymbol{x}}$ is a *local minimizer* for $f : \mathbb{R}^n \mapsto \mathbb{R}$ if

$$f(\hat{\boldsymbol{x}}) \leq f(\boldsymbol{x}) \quad \text{for} \quad \|\hat{\boldsymbol{x}} - \boldsymbol{x}\| \leq \varepsilon \qquad (\varepsilon > 0).$$

Except for the problems discussed in Chapter 5 we deal with nonlinear objective functions $f : \mathbb{R}^n \mapsto \mathbb{R}$. In general we assume that $f$ has continuous partial derivatives of second order (although we do not always make use of them in the methods). Throughout the book we shall make frequent use of the vector and matrix defined in Definitions 1.4 and 1.5 below.

**Definition 1.4. Gradient.** Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be differentiable with respect to each component of its argument. The gradient of $f$ is the vector

$$\boldsymbol{\nabla} f(\boldsymbol{x}) \equiv \begin{pmatrix} \dfrac{\partial f}{\partial x_1}(\boldsymbol{x}) \\ \vdots \\ \dfrac{\partial f}{\partial x_n}(\boldsymbol{x}) \end{pmatrix} .$$

**Definition 1.5. Hessian.** Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be twice differentiable with respect to each component of its argument. The Hessian of $f$ is the matrix

$$\boldsymbol{\nabla}^2 f(\boldsymbol{x}) \equiv \begin{pmatrix} \dfrac{\partial^2 f}{\partial x_1 \partial x_1}(\boldsymbol{x}) & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n}(\boldsymbol{x}) \\ \vdots & & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1}(\boldsymbol{x}) & \cdots & \dfrac{\partial^2 f}{\partial x_n \partial x_n}(\boldsymbol{x}) \end{pmatrix} .$$

Note that the Hessian $\boldsymbol{\nabla}^2 f$ is a symmetric matrix.

The gradient and Hessian are used in Theorem 1.6, which is well known from calculus. It tells about approximations to the value of $f$ when you change the argument from $\boldsymbol{x}$ to a neighbouring point $\boldsymbol{x}+\boldsymbol{h}$.

**Theorem 1.6. 1st and 2nd order Taylor expansions.**

If $f : \mathbb{R}^n \mapsto \mathbb{R}$ has continuous partial derivatives of second order, then

$$f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) + O(\|\boldsymbol{h}\|^2) .$$

If $f$ has continuous partial derivatives of third order, then

$$\begin{aligned} f(\boldsymbol{x}+\boldsymbol{h}) &= f(\boldsymbol{x}) + \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) + \tfrac{1}{2}\, \boldsymbol{h}^T \boldsymbol{\nabla}^2 f(\boldsymbol{x})\, \boldsymbol{h} + O(\|\boldsymbol{h}\|^3) , \\ \boldsymbol{\nabla} f(\boldsymbol{x}+\boldsymbol{h}) &= \boldsymbol{\nabla} f(\boldsymbol{x}) + \boldsymbol{\nabla}^2 f(\boldsymbol{x})\, \boldsymbol{h} + \boldsymbol{\eta} , \quad \|\boldsymbol{\eta}\| = O(\|\boldsymbol{h}\|^2) . \end{aligned}$$

The notation $e(\boldsymbol{h}) = O(\|\boldsymbol{h}\|^p)$ means that there are positive numbers $K_1$ and $\delta$ such that $|e(\boldsymbol{h})| \leq K_1 \cdot \|\boldsymbol{h}\|^p$ for all $\|\boldsymbol{h}\| \leq \delta$. In practice it is often equivalent to $|e(\boldsymbol{h})| \simeq K_2 \cdot \|\boldsymbol{h}\|^p$ for $\boldsymbol{h}$ sufficiently small; here $K_2$ is yet another positive constant.

We also use the "Big–O" notation in connection with computational complexity. When we say that an algorithm has complexity $O(n^r)$ we mean that when it is applied to a problem with $n$ variables, it uses a number of floating point operations of the form

$$C_0 n^r + C_1 n^{r-1} + \cdots + C_r ,$$

where $r$ is a positive integer. When $n$ is large, the first term dominates, and – except if memory cache problems have dominating influence – you will normally see that if you double $n$, it will take roughly $2^r$ times longer to execute.

The following theorem deals with a special type of functions that play an important role in optimization algorithms for nonlinear functions.

**Theorem 1.7.** Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{b} \in \mathbb{R}^n$ and $c \in \mathbb{R}$ be given. The *quadratic*

$$f(\boldsymbol{x}) \;=\; \tfrac{1}{2}\,\boldsymbol{x}^T \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c$$

has the gradient and Hessian

$$\boldsymbol{\nabla} f(\boldsymbol{x}) \;=\; \tfrac{1}{2}(\boldsymbol{A}+\boldsymbol{A}^T)\boldsymbol{x} + \boldsymbol{b}\;, \quad \boldsymbol{\nabla}^2 f(\boldsymbol{x}) \;=\; \tfrac{1}{2}(\boldsymbol{A}+\boldsymbol{A}^T)\;.$$

If $\boldsymbol{A}$ is symmetric, then these expression simplify to

$$\boldsymbol{\nabla} f(\boldsymbol{x}) \;=\; \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{b}\;, \quad \boldsymbol{\nabla}^2 f(\boldsymbol{x}) \;=\; \boldsymbol{A}\;.$$

**Proof.** Obviously

$$f(\boldsymbol{x}) \;=\; \tfrac{1}{2}\sum_{r=1}^{n} x_r \left( \sum_{s=1}^{n} a_{rs}x_s \right) + \sum_{r=1}^{n} b_r x_r + c\;,$$

and well-known rules for differentiation and the definition of inner products between vectors in $\mathbb{R}^n$ give

$$\frac{\partial f}{\partial x_i} \;=\; \tfrac{1}{2}\left( \sum_{s=1}^{n} a_{is}x_s + \sum_{r=1}^{n} x_r a_{ri} \right) + b_i \;=\; \tfrac{1}{2}\left( \boldsymbol{A}_{i,:}\boldsymbol{x} + \boldsymbol{A}^T_{:,i}\boldsymbol{x} \right) + b_i\;.$$

This is the $i$th element in the vector $\boldsymbol{\nabla} f(\boldsymbol{x})$, and the expression for the whole vector follows.

Proceeding with the differentiation we see that the $(i,j)$th element in the Hessian is

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \;=\; \tfrac{1}{2}\left( a_{ij} + a_{ji} \right)\;.$$

This verifies the expression for $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$.

If $\boldsymbol{A}$ is symmetric, then $\frac{1}{2}(\boldsymbol{A}+\boldsymbol{A}^T) = \boldsymbol{A}$, and this finishes the proof. □

All optimization methods for general nonlinear functions are iterative: From a starting point $\boldsymbol{x}_0$ the method produces a series of vectors $\boldsymbol{x}_0,\ \boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \ldots$, which (hopefully) converge towards $\hat{\boldsymbol{x}}$, a local minimizer for the given objective function $f$, ie

$$\boldsymbol{x}_k \;\to\; \hat{\boldsymbol{x}} \quad \text{for} \quad k \to \infty\;.$$

This is equivalent to the condition

$$\boldsymbol{e}_k \;\to\; \boldsymbol{0} \quad \text{for} \quad k \to \infty\;,$$

where $\{\boldsymbol{e}_k\}$ is the error

$$\boldsymbol{e}_k \;\equiv\; \boldsymbol{x}_k - \hat{\boldsymbol{x}}\;.$$

We cannot be sure that the errors decrease from the start, but the condition for convergence is that after sufficiently many iterations, $K$, we have decrease,

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

For some of the methods we can give results for how fast the errors decrease when we are sufficiently close to $\hat{\boldsymbol{x}}$. We distinguish between

$$\begin{aligned} \textit{Linear convergence}: &\quad \|\boldsymbol{e}_{k+1}\| \leq c_1 \|\boldsymbol{e}_k\| \ , \quad 0 < c_1 < 1 \ , \\ \textit{Quadratic convergence}: &\quad \|\boldsymbol{e}_{k+1}\| \leq c_2 \|\boldsymbol{e}_k\|^2 \ , \quad c_2 > 0 \ , \\ \textit{Superlinear convergence}: &\quad \|\boldsymbol{e}_{k+1}\| / \|\boldsymbol{e}_k\| \to 0 \quad \text{for } k \to \infty \ . \end{aligned} \tag{1.8}$$

**Example 1.3.** Consider two iterative methods, one with linear and one with quadratic convergence. At a given step they have both achieved the result with an accuracy of 3 decimals: $\|\boldsymbol{e}_k\| \leq 0.0005$. Further, let the two methods have $c_1 = c_2 = 0.5$ in (1.8).

If we want an accuracy of 12 decimals, the iteration with quadratic convergence will only need 2 more steps, whereas the iteration with linear convergence will need about 30 more steps: $0.5^{30} \simeq 10^{-9}$. ∎

## 1.2. Convexity

Finally we introduce the concept of *convexity*. This is essential for a theorem on uniqueness of a global minimizer and also for some special methods.

A set $\mathcal{D}$ is convex if the line segment between two arbitrary points in the set is contained in the set:

**Definition 1.9. Convexity of a set.** The set $\mathcal{D} \subseteq \mathbb{R}^n$ is convex if the following holds for arbitrary $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$ ,

$$\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y} \in \mathcal{D} \quad \text{for all} \quad \theta \in [0,1] \ .$$

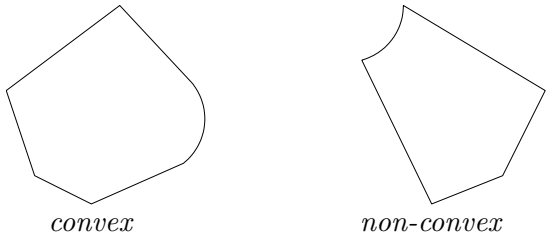



***Figure 1.5.*** *A convex and a non-convex set in* $\mathbb{R}^2$.

Convexity of a function is defined as follows,

**Definition 1.10. Convexity of a function.** Let $\mathcal{D} \subseteq \mathbb{R}^n$ be convex. The function $f$ is *convex* on $\mathcal{D}$ if the following holds for arbitrary $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$,

$$f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \;\leq\; \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y}) \quad \text{for all} \;\; \theta \in [0,1] \; .$$

$f$ is *strictly convex* on $\mathcal{D}$ if

$$f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \;<\; \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y}) \quad \text{for all} \;\; \theta \in ]0,1[ \; .$$

The figure shows a strictly convex function between two points $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$. The definition says that $f(\boldsymbol{x}_\theta)$, with

$$\boldsymbol{x}_\theta \;\equiv\; \theta\boldsymbol{x} + (1-\theta)\boldsymbol{y} \; ,$$

is below the secant between the points $\big(0, f(\boldsymbol{x})\big)$ and $\big(1, f(\boldsymbol{y})\big)$, and this holds for all choices of $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathcal{D}$.

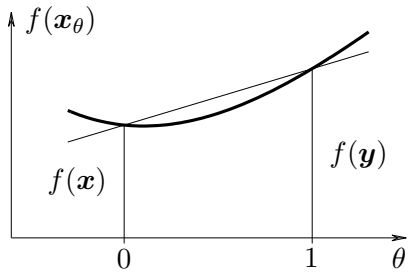



***Figure 1.6.*** *A strictly convex function.*

**Definition 1.11. Concavity of a function.** Assume that $\mathcal{D} \subseteq \mathbb{R}^n$ is convex. The function $f$ is *concave/strictly concave* on $\mathcal{D}$ if $-f$ is convex/strictly convex on $\mathcal{D}$.

**Definition 1.12. Convexity at a point.** The function $f$ is *convex* on $\mathcal{D}$ if there exists $\varepsilon > 0$ such that for arbitrary $\boldsymbol{y} \in \mathcal{D}$ with $\|\boldsymbol{x} - \boldsymbol{y}\| < \epsilon$,

$$f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \;\leq\; \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y}) \quad \text{for all} \quad \theta \in [0,1] \;.$$

$f$ is *strictly convex* at $\boldsymbol{x} \in \mathcal{D}$ if

$$f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \;<\; \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y}) \quad \text{for all} \quad \theta \in ]0,1[ \;.$$

It is easy to prove the following results:

**Theorem 1.13.** If $\mathcal{D} \subseteq \mathbb{R}^n$ is convex and $f$ is twice differentiable on $\mathcal{D}$, then

1° $f$ is convex on $\mathcal{D}$
$\quad \Leftrightarrow \quad \boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is positive semidefinite for all $\boldsymbol{x} \in \mathcal{D}$.

2° $f$ is strictly convex on $\mathcal{D}$ if $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is positive definite for all $\boldsymbol{x} \in \mathcal{D}$.

**Theorem 1.14. First sufficient condition.** If $\mathcal{D}$ is bounded and convex and if $f$ is convex on $\mathcal{D}$, then

$$f \text{ has a unique global minimizer in } \mathcal{D} \;.$$

**Theorem 1.15.** If $f$ is twice differentiable at $\boldsymbol{x} \in \mathcal{D}$, then

1° $f$ is convex at $\boldsymbol{x} \in \mathcal{D}$
$\quad \Leftrightarrow \quad \boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is positive semidefinite.

2° $f$ is strictly convex at $\boldsymbol{x} \in \mathcal{D}$ if $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is positive definite.

# Chapter 2

# Unconstrained Optimization

This chapter deals with methods for computing a local minimizer $\hat{\boldsymbol{x}}$ of a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ and starts by giving conditions that $\hat{\boldsymbol{x}}$ must satisfy. Next we discuss the general framework of a good optimization algorithm, including some basic tools such as line search and the trust region idea, which are also applicable in special cases of $f$, discussed in later chapters.

## 2.1. Conditions for a local minimizer

A local minimizer for $f$ is an argument vector $\hat{\boldsymbol{x}}$ such that $f(\hat{\boldsymbol{x}}) \leq f(\boldsymbol{x})$ for every $\boldsymbol{x}$ in some region around $\hat{\boldsymbol{x}}$, cf Definition 1.3.

We assume that $f$ has continuous partial derivatives of second order. The first order *Taylor expansion* for a function of several variables gives us an approximation to the function value at a point $\boldsymbol{x}+\boldsymbol{h}$ close to $\boldsymbol{x}$, cf Theorem 1.6,

$$f(\boldsymbol{x}+\boldsymbol{h}) \;=\; f(\boldsymbol{x}) + \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) + O\|\boldsymbol{h}\|^2 \;, \tag{2.1}$$

where $\boldsymbol{\nabla} f(\boldsymbol{x})$ is the *gradient* of $f$, cf Definition 1.4. If the point $\boldsymbol{x}$ is a local minimizer it is not possible to find an $\boldsymbol{h}$ so that $f(\boldsymbol{x}+\boldsymbol{h}) < f(\boldsymbol{x})$ when $\|\boldsymbol{h}\|$ is small. This together with (2.1) is the basis of the following theorem.

**Theorem 2.2. Necessary condition for a local minimum.**
If $\hat{\boldsymbol{x}}$ is a local minimizer for $f : \mathbb{R}^n \mapsto \mathbb{R}$, then

$$\boldsymbol{\nabla} f(\hat{\boldsymbol{x}}) \;=\; \mathbf{0} \;.$$

The local minimizers are not the only points with $\boldsymbol{\nabla} f(\boldsymbol{x}) = \mathbf{0}$. Such points have a special name:

**Definition 2.3. Stationary point.** If $\boldsymbol{\nabla} f(\boldsymbol{x}_{\mathrm{s}}) = \mathbf{0}$, then $\boldsymbol{x}_{\mathrm{s}}$ is said to be a *stationary point* for $f$.

The stationary points are the local maximizers, the local minimizers and "the rest". To distinguish between them, we need one extra term in the Taylor expansion. If $f$ has continuous third derivatives, then

$$f(\boldsymbol{x}+\boldsymbol{h}) \;=\; f(\boldsymbol{x}) + \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) + \tfrac{1}{2}\, \boldsymbol{h}^T \boldsymbol{\nabla}^2 f(\boldsymbol{x})\, \boldsymbol{h} + O\|\boldsymbol{h}\|^3 \;, \qquad (2.4)$$

where the *Hessian* $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ of the function $f$ is a matrix containing the second partial derivatives of $f$, cf Definition 1.5. For a stationary point (2.4) takes the form

$$f(\boldsymbol{x}_{\mathrm{s}}+\boldsymbol{h}) \;=\; f(\boldsymbol{x}_{\mathrm{s}}) + \tfrac{1}{2}\, \boldsymbol{h}^T \boldsymbol{\nabla}^2 f(\boldsymbol{x}_{\mathrm{s}})\, \boldsymbol{h} + O\|\boldsymbol{h}\|^3 \;.$$

If the second term is positive for all $\boldsymbol{h}$ we say that the matrix $\boldsymbol{\nabla}^2 f(\boldsymbol{x}_{\mathrm{s}})$ is *positive definite*[1)]. Further, we can take $\|\boldsymbol{h}\|$ so small that the remainder term is negligible, and it follows that $\boldsymbol{x}_{\mathrm{s}}$ is a local minimizer. Thus we have (almost) proved the following theorem.

**Theorem 2.5. Second order necessary condition.**
If $\hat{\boldsymbol{x}}$ is a local minimizer, then $\boldsymbol{\nabla}^2 f(\hat{\boldsymbol{x}})$ is positive semidefinite.

**Sufficient condition for a local minimum.**
Assume that $\boldsymbol{x}_{\mathrm{s}}$ is a stationary point and that $\boldsymbol{\nabla}^2 f(\boldsymbol{x}_{\mathrm{s}})$ is positive definite. Then $\boldsymbol{x}_{\mathrm{s}}$ is a local minimizer.

The Taylor expansion (2.4) is also used in the proof of the following corollary,

**Corollary 2.6.** Assume that $\boldsymbol{x}_{\mathrm{s}}$ is a stationary point and that $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is positive semidefinite for $\boldsymbol{x}$ in a neighbourhood of $\boldsymbol{x}_{\mathrm{s}}$. Then $\boldsymbol{x}_{\mathrm{s}}$ is a local minimizer.

[1)] Positive definite matrices are discussed in Appendix A.2, which also gives tools for checking definiteness.

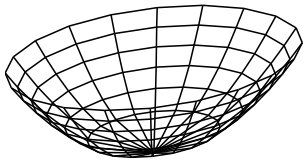
Now we are ready for the following corollary which can be used to characterize a stationary point. Again the Taylor expansion (2.4) is used in the proof.

**Corollary 2.7.** Assume that $\boldsymbol{x}_{\mathrm{s}}$ is a stationary point and that $\boldsymbol{\nabla}^2 f(\boldsymbol{x}_{\mathrm{s}}) \neq \boldsymbol{0}$. Then

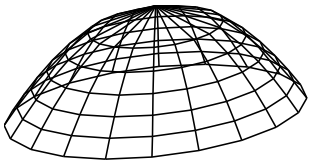
| $\boldsymbol{\nabla}^2 f(\boldsymbol{x}_{\mathrm{s}})$ | $\boldsymbol{x}_{\mathrm{s}}$ |
|---|---|
| positive definite | local minimizer |
| positive semidefinite | local minimizer or a saddle point |
| negative definite | local maximizer |
| negative semidefinite | local maximizer or a saddle point |
| indefinite | saddle point |

A matrix is indefinite if it is neither definite nor semidefinite. If the Hessian $\boldsymbol{\nabla}^2 f(\boldsymbol{x}_{\mathrm{s}}) = \boldsymbol{0}$, then we need higher order terms in the Taylor expansion in order to find the local minimizers among the stationary points.

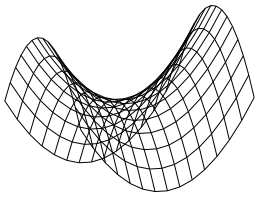
**Example 2.1.** We consider functions of two variables. Below we show the variation of the function value near a local minimizer (Figure 2.1a), a local maximizer (Figure 2.1b) and a saddle point (Figure 2.1c). It is a characteristic of a saddle point that there exist two lines through $\boldsymbol{x}_{\mathrm{s}}$ with the property that the variation of the $f$-value looks like a local minimum along one of the lines, and like a local maximum along the other line.



*a) minimum*



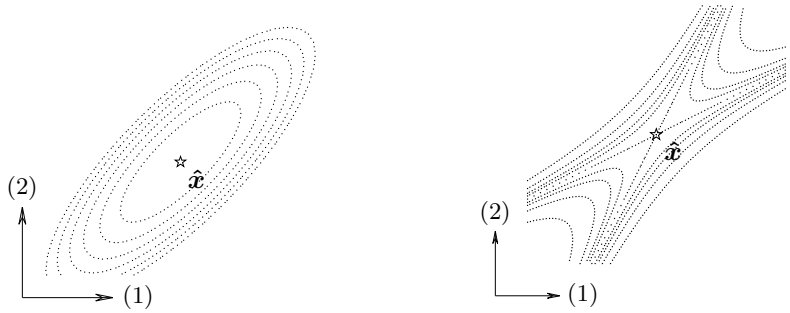
*b) maximum*



*c) saddle point*

***Figure 2.1.*** *$\boldsymbol{x} \in \mathbb{R}^2$. Surfaces $z = f(\boldsymbol{x})$ near a stationary point.*

The *contours* of our function look approximately like concentric ellipses near a local maximizer or a local minimizer (Figure 2.2a), whereas the hyperbolas shown in Figure 2.2b correspond to a saddle point.




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

***Figure 2.2.*** *Contours of a function near a stationary point.* ∎

## 2.2. Descent methods

As discussed in Section 1.1 we have to use an iterative method to solve a nonlinear optimization problem: From a starting point $\boldsymbol{x}_0$ the method produces a series of vectors $\boldsymbol{x}_0,\ \boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \ldots$, which in most cases converges under certain mild conditions. We want the series to converge towards $\hat{\boldsymbol{x}}$, a local minimizer for the given objective function $f : \mathbb{R}^n \mapsto \mathbb{R}$, ie

$$\boldsymbol{x}_k \ \to\ \hat{\boldsymbol{x}} \quad \text{for} \quad k \to \infty\ .$$

In (almost) all the methods there are measures which enforce the *descending property*

$$f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)\ . \tag{2.8}$$

This prevents convergence to a maximizer and also makes it less probable that we get convergence to a saddle point. We talk about the *global convergence* properties of a method, ie convergence when the iteration process starts at a point $\boldsymbol{x}_0$, which is not close to a local minimizer $\hat{\boldsymbol{x}}$. We want our method to produce iterates that move steadily towards a neighbourhood of $\hat{\boldsymbol{x}}$. For instance, there are methods for which it is possible to prove that any accumulation point (ie limit of a subseries) of $\{\boldsymbol{x}_k\}$ is a stationary point (Definition 2.3), ie the gradients tend to zero:

$$\boldsymbol{\nabla} f(\boldsymbol{x}_k) \to \boldsymbol{0} \quad \text{for} \quad k \to \infty\ .$$

This does not exclude convergence to a saddle point or even a maximizer, but the descending property (2.8) prevents this in practice. In this

"*global part*" of the iteration we are satisfied if the current errors do not increase except for the very first steps. The requirement is

$$\|\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}\| \ < \ \|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\| \quad \text{for} \quad k > K \ .$$

In the *final stages* of the iteration where the $\boldsymbol{x}_k$ are close to $\hat{\boldsymbol{x}}$ we expect faster convergence. The local convergence results tell us how quickly we can get a result which agrees with $\hat{\boldsymbol{x}}$ to a desired accuracy. See (1.8) about the three types of convergence: linear, quadratic and superlinear.

### 2.2.1. Fundamental structure of a descent method

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

In order not to walk in circles he decides to use straight strides, ie with constant compass bearing. From what his feet tell him about the local slope he chooses a direction and walks in that direction until his feet tell him that he is on an uphill slope.

Now he has to decide on a new direction and he starts his next stride. Let us hope that he reaches his goal in the end. ∎

The pattern of events in this example is the basis of the algorithms for descent methods, see Algorithm 2.9 below.

**Algorithm 2.9. Descent method**

Given starting point $\boldsymbol{x}_0$

**begin**
  $k := 0$; $\boldsymbol{x} := \boldsymbol{x}^{[0]}$; *found* := **false** {Initialise}
  **while** (**not** *found*) **and** ($k < k_{\max}$)
    $\boldsymbol{h}_{\mathrm{d}}$ := search_direction($\boldsymbol{x}$) {From $\boldsymbol{x}$ and downhill}
    **if** (no such $\boldsymbol{h}_{\mathrm{d}}$ exists)
      *found* := **true** {$\boldsymbol{x}$ is stationary}
    **else**
      Find "step length" $\alpha$ { see below}
      $\boldsymbol{x} := \boldsymbol{x} + \alpha\boldsymbol{h}_{\mathrm{d}}$; $k := k+1$ {next iterate}
      *found* := update(*found*) {Stopping criteria}
    **end**
  **end**
**end**

The search direction $\boldsymbol{h}_{\mathrm{d}}$ must be a *descent direction*, see Section 2.2.2. Then we are able to gain a smaller value of $f(\boldsymbol{x})$ by choosing an appropriate walking distance, and thus we can satisfy the descending condition (2.8). In Sections 2.3 – 2.4 we introduce different methods for choosing the appropriate step length, ie $\alpha$ in Algorithm 2.9.

Typically the *stopping criteria* include two or more from the following list, and the process stops if one of the chosen criteria is satisfied.

$$\begin{aligned} 1^\circ: &\quad \|\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\| \;\le\; \varepsilon_1 \;, \\ 2^\circ: &\quad f(\boldsymbol{x}_k)-f(\boldsymbol{x}_{k+1}) \;\le\; \varepsilon_2 \;, \\ 3^\circ: &\quad \|\boldsymbol{\nabla} f(\boldsymbol{x}_k)\|_\infty \;\le\; \varepsilon_3 \;, \\ 4^\circ: &\quad k \;> k_{\max} \;. \end{aligned} \tag{2.10}$$

The first two criteria are computable approximations to what we really want, namely that the current error is sufficiently small,

$$\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\| \;\le\; \delta_1 \;,$$

or that the current value of $f(\boldsymbol{x})$ is sufficiently close to the minimal value,

$$f(\boldsymbol{x}_k) - f(\hat{\boldsymbol{x}}) \;\le\; \delta_2 \;.$$

Both conditions reflect the convergence $\boldsymbol{x}_k \to \hat{\boldsymbol{x}}$.

Condition 3° reflects that $\boldsymbol{\nabla} f(\boldsymbol{x}_k) \to 0$ for $k \to \infty$. We must emphasize that even if one of the conditions 1° – 3° in (2.10) is satisfied with a small $\varepsilon$-value, we cannot be sure that $\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|$ or $f(\boldsymbol{x}_k)-f(\hat{\boldsymbol{x}})$ is small. Condition 4° is a must in any iterative algorithm. It acts as a "safe guard" against an infinite loop that might result if there were errors in the implementation of $f$ or $\boldsymbol{\nabla} f$, or if for instance $\varepsilon_1$ or $\varepsilon_3$ had been chosen so small that rounding errors prevent the other conditions from being satisfied.

The first criterion is often split into two

$$\begin{aligned} 1\mathrm{a}): &\quad \|\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\| \;\le\; \varepsilon_{1\mathrm{a}} \;, \\ 1\mathrm{b}): &\quad \|\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\| \;\le\; \varepsilon_{1\mathrm{r}} \|\boldsymbol{x}_{k+1}\| \;, \end{aligned}$$

with the purpose that if $\hat{\boldsymbol{x}}$ is small, then we want to find it with the *absolute accuracy* $\varepsilon_{1\mathrm{a}}$, while we are satisfied with *relative accuracy* $\varepsilon_{1\mathrm{r}}$ if $\hat{\boldsymbol{x}}$ is large. Sometimes these two criteria are combined into one,

$$1^\diamond: \quad \|\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\| \;\le\; \widetilde{\varepsilon}_1(\widetilde{\varepsilon}_1 + \|\boldsymbol{x}_{k+1}\|) \;. \tag{2.11}$$

This gives a gradual change from $\varepsilon_{1\mathrm{a}} = \widetilde{\varepsilon}_1^2$ when $\hat{\boldsymbol{x}}$ is close to $\mathbf{0}$, to $\varepsilon_{1\mathrm{r}} = \widetilde{\varepsilon}_1$ when $\hat{\boldsymbol{x}}$ is large.

### 2.2.2. Descent directions

From the current point $\boldsymbol{x}$ we wish to find a direction which brings us downhill, a *descent direction*. This means that if we take a small step in that direction we get to a point with a smaller function value.

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

The Taylor expansion (2.1) gives us a first order approximation to the function value in a neighbouring point to $\boldsymbol{x}$, in direction $\boldsymbol{h}$:

$$f(\boldsymbol{x}+\alpha\boldsymbol{h}) \;=\; f(\boldsymbol{x}) + \alpha\boldsymbol{h}^T\boldsymbol{\nabla} f(\boldsymbol{x}) + O(\alpha^2)\;, \quad \text{with} \;\; \alpha > 0\;.$$

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

$$f(\boldsymbol{x}+\alpha\boldsymbol{h}) \;\simeq\; f(\boldsymbol{x}) + \alpha\boldsymbol{h}^T\boldsymbol{\nabla} f(\boldsymbol{x})\;.$$

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

$$\mathcal{H} \;=\; \{\boldsymbol{x}+\boldsymbol{h} \mid \boldsymbol{h}^T\boldsymbol{\nabla} f(\boldsymbol{x}) = 0\}\;.$$

This hyperplane divides the space into an "uphill" half space and a "downhill" half space. The latter is the half space that we want; it has the vector $-\boldsymbol{\nabla} f(\boldsymbol{x})$ pointing into it. Figure 2.3 gives the situation in $\mathbb{R}^2$, where the hyperplane $\mathcal{H}$ is just a line.

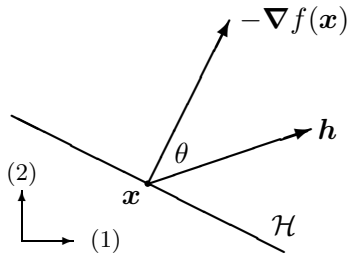



***Figure 2.3.*** *$\mathbb{R}^2$ divided into a "downhill" and an "uphill" half space.*

We now define a descent direction. This is a "downhill" direction, ie, it is inside the "good" half space:

**Definition 2.12. Descent direction.**

$\boldsymbol{h}$ is a descent direction from $\boldsymbol{x}$ if $\boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) < 0$ .

The vector $\boldsymbol{h}$ in Figure 2.3 is a descent direction. The angle $\theta$ between $\boldsymbol{h}$ and the negative gradient is

$$\theta \;=\; \angle(\boldsymbol{h}, -\boldsymbol{\nabla} f(\boldsymbol{x})) \quad \text{with} \quad \cos\theta \;=\; \frac{-\boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x})}{\|\boldsymbol{h}\| \cdot \|\boldsymbol{\nabla} f(\boldsymbol{x})\|} \;\; . \tag{2.13}$$

We state a condition on this angle,

**Definition 2.14. Absolute descent method.**

This is a method, where the search directions $\boldsymbol{h}_k$ satisfy

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

for all $k$, with $\mu > 0$ independent of $k$.

The discussion above is based on the equivalence between a vector in $\mathbb{R}^2$ and a geometric vector in the plane, and it is easily seen to be valid also in $\mathbb{R}^3$. If the dimension $n$ is larger than 3, we call $\theta$ the "*pseudo angle*" between $\boldsymbol{h}$ and $-\boldsymbol{\nabla} f(\boldsymbol{x})$. In this way we can use (2.13) and Definition 2.14 for all $n \geq 2$. The restriction that $\mu$ must be constant in all the steps is necessary for the global convergence result which we give in the next section.

The following theorem will be used several times in the remainder of the book.

**Theorem 2.15.** If $\boldsymbol{\nabla} f(\boldsymbol{x}) \neq \boldsymbol{0}$ and $\boldsymbol{B}$ is a symmetric, positive definite matrix, then

$$\acute{\boldsymbol{h}} \;=\; -\boldsymbol{B} \boldsymbol{\nabla} f(\boldsymbol{x}) \quad \text{and} \quad \grave{\boldsymbol{h}} = -\boldsymbol{B}^{-1} \boldsymbol{\nabla} f(\boldsymbol{x})$$

are descent directions.

**Proof.** A positive definite matrix $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ satisfies

$$\boldsymbol{u}^T \boldsymbol{B}\, \boldsymbol{u} > 0 \quad \text{for all } \boldsymbol{u} \in \mathbb{R}^n,\; \boldsymbol{u} \neq \boldsymbol{0} \;.$$

If we take $\boldsymbol{u} = \widetilde{\boldsymbol{h}}$ and exploit the symmetry of $\boldsymbol{B}$, we get

$$\acute{\boldsymbol{h}}^T \boldsymbol{\nabla} f(\boldsymbol{x}) \;=\; -\boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{B}^T \boldsymbol{\nabla} f(\boldsymbol{x}) \;=\; -\boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{B}\, \boldsymbol{\nabla} f(\boldsymbol{x}) < 0 \;.$$

With $\boldsymbol{u} = \grave{\boldsymbol{h}}$ we get

$$\grave{\boldsymbol{h}}^T \boldsymbol{\nabla} f(\boldsymbol{x}) \;=\; \grave{\boldsymbol{h}}^T (-\boldsymbol{B}\, \grave{\boldsymbol{h}} \;=\; -\grave{\boldsymbol{h}}^T \boldsymbol{B}\, \grave{\boldsymbol{h}} < 0 \;.$$

Thus, the condition in Definition 2.12 is satisfied in both cases. □

## 2.3. Line search

Once a descent direction has been determined, it must be decided how long the step in this direction should be. In this section we shall introduce the idea of *line search*. We study the variation of the objective function $f$ along the direction $\boldsymbol{h}$ from the current $\boldsymbol{x}$,

$$\varphi(\alpha) \;=\; f(\boldsymbol{x}+\alpha\boldsymbol{h}), \quad \text{with fixed } \boldsymbol{x} \text{ and } \boldsymbol{h} \;. \tag{2.16}$$

Figure 2.4 shows an example of the variation of $\varphi(\alpha)$ when $\boldsymbol{h}$ is a descent direction. The obvious idea is to use what is known as *exact line search*: Determine the step as $\alpha = \alpha_{\mathrm{e}}$, the local minimizer of $\varphi$ shown in the figure.

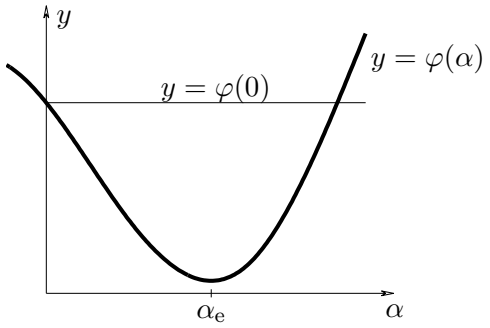



***Figure 2.4.*** *Variation of the cost function along the search line.*

Often, however, we are not only given $\boldsymbol{h}$, but also a guess on $\alpha$, for instance $\alpha = 1$, and it might be wasteful to spend a lot of function evaluations in order to determine the local minimum in the direction given by the fixed $\boldsymbol{h}$. Experience shows that in general it is better to use *soft line search*, which we shall describe now; we return to exact line search in Section 2.3.2.

First we look at the slope of $\varphi$. Applying well-known rules of differentiations to the function $\varphi$ defined by (2.16) we get

$$\frac{d\,\varphi}{d\,\alpha} \;=\; \frac{\partial f}{\partial x_1}\,\frac{d\,x_1}{d\,\alpha} + \cdots + \frac{\partial f}{\partial x_n}\,\frac{d\,x_n}{d\,\alpha} \;,$$

which is equivalent to

$$\varphi'(\alpha) \;=\; \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}+\alpha\boldsymbol{h}) \;. \tag{2.17}$$

This means that

$$\varphi'(0) \;=\; \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) \;<\; 0 \;,$$

since $\boldsymbol{h}$ is a descent direction, cf Definition 2.12. Therefore, if $\alpha > 0$ is very small, we are guaranteed to get to a point that satisfies the descending condition (2.8), but we should guard against the step being so short that the decrease in function value is unnecessarily small. On the other hand, the figure shows that it may happen that $\varphi(\alpha) > \varphi(0)$ if we take $\alpha$ too large.

The aim of soft line search is to find a "step length" $\alpha_s$ such that we get a useful decrease in the value of the cost function. Figure 2.5 shows that in general there will be an interval of acceptable $\alpha_s$-values. This interval is defined by the conditions

$$\begin{aligned} 1^\circ: &\quad \varphi(\alpha_s) \le \lambda(\alpha_s) = \varphi(0) + \beta_1 \varphi'(0) \cdot \alpha_s \,, \\ 2^\circ: &\quad \varphi'(\alpha_s) \ge \beta_2 \varphi'(0) \,, \end{aligned} \tag{2.18}$$

where $0 < \beta_1 < 0.5$ and $\beta_1 < \beta_2 < 1$. Often $\beta_1$ is chosen very small, for instance $\beta_1 = 10^{-3}$ and $\beta_2$ is close to 1, for instance $\beta_2 = 0.99$.

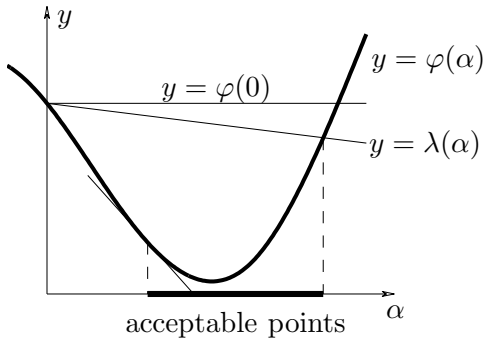



***Figure 2.5.*** *Acceptable points according to conditions* (2.18).

Descent methods with line search governed by (2.18) are normally convergent; see Theorem 2.19 below.

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

**Theorem 2.19.** Consider an absolute descent method following Algorithm 2.9 with search directions that satisfy Definition 2.14 and with line search controlled by (2.18).

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

$$\begin{aligned} &\text{either} \quad \boldsymbol{\nabla} f(\boldsymbol{x}_k) = \boldsymbol{0} \text{ for some } k \ , \\ &\text{or} \quad f(\boldsymbol{x}_k) \ \to \ -\infty \ , \\ &\text{or} \quad \boldsymbol{\nabla} f(\boldsymbol{x}_k) \ \to \ \boldsymbol{0} \ . \end{aligned}$$

**Proof.** See [19, pp 30–33]. □

### 2.3.1. An algorithm for soft line search

Many researchers in optimization have proved their inventiveness by producing new line search methods or modifications to known methods. We shall present a useful combination of ideas with different origin.

The purpose of the algorithm is to find $\alpha_{\mathrm{s}}$, an acceptable argument for the function

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

The acceptability is decided by the criteria (2.18), which express the demands that $\alpha_{\mathrm{s}}$ must be sufficiently small to give a useful decrease in the objective function, and sufficiently large to ensure that we have left the starting tangent of the curve $y = \varphi(\alpha)$ for $\alpha \geq 0$, cf Figure 2.3.

The algorithm has two parts. First we find an interval $[a, b]$ that contains acceptable points, see Figure 2.6.

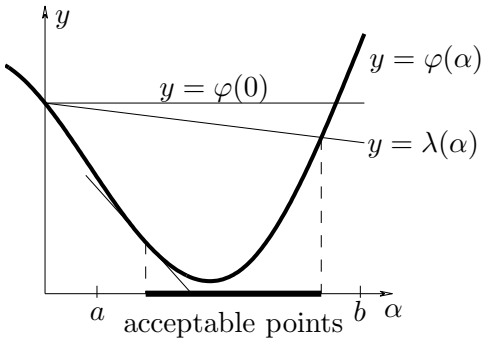



***Figure 2.6.*** *Interval $[a, b]$ containing acceptable points.*

In the second part of the algorithm we successively reduce the interval: We find a point $\alpha$ in the strict interior of $[a, b]$. If both conditions in (2.18) are satisfied by this $\alpha$-value, then we are finished ($\alpha_s = \alpha$). Otherwise, the reduced interval is either $[a, b] := [a, \alpha]$ or $[a, b] := [\alpha, b]$, where the choice is made so that the reduced $[a, b]$ contains acceptable points.

**Algorithm 2.20. Soft line search**

**begin**
  $\alpha := 0$;  $k := 0$;
  **if** $\varphi'(0) < 0$ {1°}
    $a := 0$;  $b := \min\{1, \alpha_{max}\}$;  *stop* := **false** {2°}
    **while not** *stop* **and** $k < k_{max}$
      $k := k{+}1$
      **if** $\varphi(b) < \lambda(b)$ {3°}
        $a := b$
        **if** $\varphi'(b) < \beta_2\varphi'(0)$ **and** $b < \alpha_{\max}$ {4°}
          $b := \min\{2b, \alpha_{\max}\}$
        **else**
          *stop* := **true**
      **elseif** $a = 0$ **and** $\varphi'(b) < 0$ {5°}
        $b := b/10$
      **else**
        *stop* := **true**
    **end**
    $\alpha := b$;  *stop* := $\big(a > 0$ **and** $\varphi'(b) \geq \beta_2\varphi'(0)\big)$ {6°}
      **or** $\big(b \geq \alpha_{\max}$ **and** $\varphi'(b) < \beta_2\varphi'(0)\big)$
    **while not** *stop* **and** $k < k_{max}$
      $k := k{+}1$;  Refine $\alpha$ and $[a, b]$ {7°}
      *stop* := $\varphi(\alpha) \leq \lambda(\alpha)$ **and** $\varphi'(\alpha) \geq \beta_2\varphi'(0)$
    **end**
    **if** $\varphi(\alpha) \geq \varphi(0)$ {8°}
      $\alpha := 0$
**end**

We have the following remarks to Algorithm 2.20.

1° If $\boldsymbol{x}$ is a stationary point ($\boldsymbol{\nabla} f(\boldsymbol{x}) = \boldsymbol{0} \Rightarrow \varphi'(0) = 0$) or if $\boldsymbol{h}$ is not downhill, then we do not wish to move, and return $\alpha = 0$.

2° The initial choice $b = 1$ is used because in many optimization methods (for instance the Newton-type methods in Chapter 3) $\alpha = 1$ is a very good guess in the final iterations. The upper bound $\alpha_{\max}$ must be supplied by the user. It acts as a guard against overflow if $f$ is unbounded.

3° The value $\alpha = b$ satisfies condition 1° in (2.18), and we shall use it as lower bound for the further search.

4° Condition 2° in (2.18) is not satisfied, but we can increase the right hand end of the interval $[a, b]$. If $\alpha_{\max}$ is sufficiently large, then the series of $b$-values is $1, 2, 4, \ldots$, corresponding to an "expansion factor" of 2. Other factors could be used.

5° This situation occurs if $b$ is to the right of a maximum of $\varphi(\alpha)$. We know that $\varphi'(0) < 0$, and by sufficient reduction of $b$ we can get to the left of the maximum. It should be noted that we cannot guarantee that the algorithm finds the smallest minimizer of $\varphi$ in case there are several minimizers in $[0, \alpha_{\max}]$.

6° Initialization for second part of the algorithm, if necessary. If $\alpha = b$ satisfies both conditions in (2.18), or if $\alpha_{\max}$ is so small that $b$ is to the left of the set of acceptable points in Figure 2.6, then we are finished.

7° See Algorithm 2.21 below.

8° The algorithm may have stopped abnormally, for instance because we have used all the allowed $k_{\max}$ function evaluations. If the current value of $\alpha$ does not decrease the objective function, then we return $\alpha = 0$, cf 1°.

The refinement can be made by the following Algorithm 2.21. The input is an interval $[a, b]$ which we know contains acceptable points, and the output is an $\alpha$ found by interpolation. We want to be sure that the intervals have strictly decreasing widths, so we only accept the new $\alpha$ if it is inside $[a+d, b-d]$, where $d = \frac{1}{10}(b - a)$. The $\alpha$ splits $[a, b]$ into two subintervals, and we also return the subinterval which must contain acceptable points.

**Algorithm 2.21. Refine**

**begin**
  $D := b - a; \quad c := \big(\varphi(b) - \varphi(a) - D * \varphi'(a)\big)/D^2$ {9°}
  **if** $c > 0$
    $\alpha := a - \varphi'(a)/(2c)$
    $\alpha := \min\big\{\max\{\alpha,\ a{+}0.1D\},\ b{-}0.1D\}\big\}$ {10°}
  **else**
    $\alpha := (a + b)/2$
  **if** $\varphi(\alpha) < \lambda(\alpha)$ {11°}
    $a := \alpha$
  **else**
    $b := \alpha$
**end**

We have the following remarks to this algorithm.

9° The second degree polynomial

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

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

10° Ensure that $\alpha$ is in the middle 80% of the interval

11° If $\varphi(\alpha)$ is sufficiently small, then the right hand part of $[a, b]$ contain points that satisfy both of the constraints (2.18). Otherwise, $[\alpha, b]$ is sure to contain acceptable points.

Finally, we give some remarks about the *implementation* of the algorithm: The function and slope values are computed as

$$\varphi(\alpha) = f(\boldsymbol{x}{+}\alpha\boldsymbol{h}), \qquad \varphi'(\alpha) = \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}{+}\alpha\boldsymbol{h}) \ .$$

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

$$\boldsymbol{x}_{\text{new}} \ = \ \text{line\_search}(\boldsymbol{x}, \boldsymbol{h}) \tag{2.22}$$

to denote the result $\boldsymbol{x}{+}\alpha\boldsymbol{h}$ of a line search from $\boldsymbol{x}$ in the direction $\boldsymbol{h}$.

### 2.3.2. Exact line search

An algorithm for exact line search produces a "step length" which is sufficiently close to the true result, $\alpha_s \simeq \alpha_e$ with

$$\alpha_e \;\equiv\; \text{argmin}_{\alpha \geq 0}\; \varphi(\alpha)\;. \tag{2.23}$$

The algorithm is similar to the soft line search in Algorithm 2.20. The main differences are that the updating of $a$ is moved from the line before remark 4° to the line after this condition, and in the refinement loop the expression for *stop* is changed to

$$\textit{stop} \;:=\; \big(\,|\varphi'(\alpha)| \;\leq\; \tau * |\varphi'(0)|\,\big) \;\;\textbf{or}\;\; \big(\,b{-}a \;\leq\; \varepsilon\,\big)\;. \tag{2.24}$$

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

An advantage of an exact line search is that (in theory at least) it can produce its results exactly, and this is needed in some theoretical convergence results concerning conjugate gradient methods, see Section 2.7.

The exact minimizer, as defined by (2.23), has $\varphi'(\alpha_e) = 0$. From (2.17), $\varphi'(\alpha) = \boldsymbol{h}^T \boldsymbol{\nabla} f((\boldsymbol{x}{+}\alpha\boldsymbol{h})$, we see that either $\boldsymbol{\nabla} f(\boldsymbol{x}{+}\alpha_e\boldsymbol{h}) = \boldsymbol{0}$, which is a perfect result (we have found a stationary point for $f$), or

$$\boldsymbol{\nabla} f(\boldsymbol{x}{+}\alpha_e\boldsymbol{h}) \perp \boldsymbol{h}\;. \tag{2.25}$$

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

**Example 2.4.** A "divine power" with a radar set follows the movements of our wayward tourist. He has decided to continue in a given direction, until his feet tell him that he starts to go uphill. The "divine power" can see that he stops where the given direction is tangent to a local contour. This is equivalent to the orthogonality formulated in (2.25).

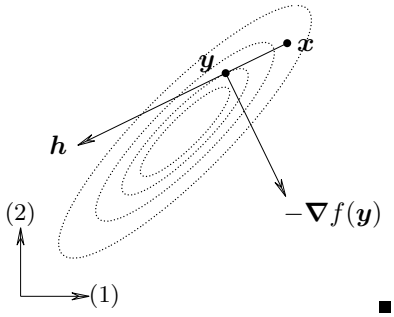



***Figure 2.7.*** *An exact line search stops at $\boldsymbol{y} = \boldsymbol{x}{+}\alpha_e\boldsymbol{h}$, where the local gradient is orthogonal to the search direction.*

■

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

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

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

## 2.4. Descent methods with trust region

The methods in this chapter produce series of steps leading from the starting point to the final result, we hope. Generally the directions of the steps are determined by the properties of $f(\boldsymbol{x})$ at the current iterate. Similar considerations lead us to the trust region methods, where the iteration steps are determined from the properties of a *model* of the objective function inside a given region. The size of the region is modified during the iteration.

In the methods that we discuss, the model is either the affine approximation to $f$ given by the 1st order Taylor expansion, Theorem 1.6:

$$f(\boldsymbol{x}+\boldsymbol{h}) \simeq q(\boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) , \tag{2.26}$$

or it is a quadratic approximation to $f$:

$$f(\boldsymbol{x}+\boldsymbol{h}) \simeq q(\boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) + \tfrac{1}{2}\, \boldsymbol{h}^T \boldsymbol{H}\, \boldsymbol{h} , \tag{2.27}$$

where $\boldsymbol{H}$ is a symmetric, positive definite matrix. In both cases $q(\boldsymbol{h})$ is a good approximation to $f(\boldsymbol{x}+\boldsymbol{h})$ only if $\|\boldsymbol{h}\|$ is sufficiently small. These considerations lead us to determine the new iteration step as the solution

to the following *model problem*,

$$\boldsymbol{h}_{\rm tr} \;=\; \underset{\boldsymbol{h}\,\in\,\mathcal{T}}{\rm argmin}\{q(\boldsymbol{h})\}\;; \quad \mathcal{T} \;=\; \{\boldsymbol{h} \mid \|\boldsymbol{h}\| \le \Delta\}, \quad \Delta > 0\;, \tag{2.28}$$

where $q(\boldsymbol{h})$ is given by (2.26) or (2.27). The region $\mathcal{T}$ is called the *trust region* and $\Delta$ is the trust region radius.

We use $\boldsymbol{h} = \boldsymbol{h}_{\rm tr}$ as a candidate for our next step, and reject the step if it proves not to be downhill, ie if $f(\boldsymbol{x}+\boldsymbol{h}) \ge f(\boldsymbol{x})$. The gain in cost function value controls the size of the trust region for the next step: The gain is compared to the gain predicted by the approximation function, and we introduce the *gain ratio*

$$\varrho \;=\; \frac{f(\boldsymbol{x}) - f(\boldsymbol{x}+\boldsymbol{h})}{q(\boldsymbol{0}) - q(\boldsymbol{h})}\;. \tag{2.29}$$

By construction the denominator is positive, so a negative $\varrho$ indicates an uphill step. When $\varrho$ is small (maybe even negative) our approximation agrees poorly with $f$, and we should reduce $\Delta$ in order to get shorter steps, and thereby a better agreement between $f(\boldsymbol{x}+\boldsymbol{h})$ and the approximation $q(\boldsymbol{h})$. A large value for $\varrho$ indicates a satisfactory decrease in the cost function value and we can increase $\Delta$ in the hope that larger steps will bring us to the target $\hat{\boldsymbol{x}}$ in fewer iterations. These ideas are summarized in the following algorithm.

**Algorithm 2.30. Descent method with trust region**
Given $\boldsymbol{x}_0$ and $\Delta_0$
**begin**
  $k := 0$; $\boldsymbol{x} := \boldsymbol{x}_0$; $\Delta := \Delta_0$; *found* := **false** {starting point}
  **repeat**
    $k := k+1$; $\boldsymbol{h}_{\rm tr}$ := Solution of model problem (2.28)
    $\varrho$ := gain factor (2.29)
    **if** $\varrho > 0.75$ {very good step}
      $\Delta := 2 * \Delta$ {larger trust region}
    **if** $\varrho < 0.25$ {poor step}
      $\Delta := \Delta/3$ {smaller trust region}
    **if** $\varrho > 0$ {reject step if $\varrho \le 0$}
      $\boldsymbol{x} := \boldsymbol{x} + \boldsymbol{h}_{\rm tr}$
    Update *found* {stopping criteria, for instance from (2.10)}
  **until** *found* **or** $k > k_{\max}$
**end**

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

There are versions of the trust region method where "$\varrho < 0.25$" initiates an interpolation between $\boldsymbol{x}$ and $\boldsymbol{x}+\boldsymbol{h}$ based on known values of $f$ and $\boldsymbol{\nabla} f$, and/or "$\varrho > 0.75$" leads to an extrapolation along the direction $\boldsymbol{h}$, a line search actually. Actions like this can be rather costly, and Fletcher [19, Chapter 5] claims that the improvements in performance may be marginal. In the same reference you can find theorems about the global performance of methods like Algorithm 2.30.

## 2.5. The Steepest Descent method

Until now we have not answered an important question connected with Algorithm 2.9: Which of the possible descent directions (see Definition 2.12) do we choose as search direction?

Our first considerations will be based purely on local first order information. Which descent direction gives us the greatest gain in function value relative to the step length? Using the first order Taylor expansion from Theorem 1.6 we get the following approximation

$$\frac{f(\boldsymbol{x}) - f(\boldsymbol{x}+\alpha\boldsymbol{h})}{\alpha\|\boldsymbol{h}\|} \;\simeq\; -\frac{\boldsymbol{h}^T\boldsymbol{\nabla} f(\boldsymbol{x})}{\|\boldsymbol{h}\|} \;=\; \|\boldsymbol{\nabla} f(\boldsymbol{x})\| \cos\theta \;,$$

where $\theta$ is the pseudo angle between $\boldsymbol{h}$ and $-\boldsymbol{\nabla} f(\boldsymbol{x})$, cf (2.13). We see that the relative gain is greatest when the angle $\theta = 0$, ie when $\boldsymbol{h} = \boldsymbol{h}_{\mathrm{sd}}$, given by

$$\boldsymbol{h}_{\mathrm{sd}} \;=\; -\boldsymbol{\nabla} f(\boldsymbol{x}) \;. \tag{2.31}$$

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

According to Theorem 2.19 a descent method based on steepest descent and soft or exact line search is convergent. If we make a method

using $\boldsymbol{h}_{\rm sd}$ and a version of line search that ensures sufficiently short steps, then the global convergence will manifest itself as a very robust global performance. The disadvantage is that the method will have linear final convergence and this will often be exceedingly slow. If we use exact line search together with steepest descent, we invite trouble.

**Example 2.5.** We test a steepest descent method with exact line search on the function from Example 1.2,

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

The gradient is

$$\boldsymbol{\nabla} f(\boldsymbol{x}) \;=\; \begin{pmatrix} 2(x_1 + x_2 - 2) + 200(x_1 - x_2) \\ 2(x_1 + x_2 - 2) - 200(x_1 - x_2) \end{pmatrix} \;.$$

If the starting point is taken as $\boldsymbol{x}_0 = \begin{pmatrix} 3 & 598/202 \end{pmatrix}^T$, then the first search direction is

$$\boldsymbol{h}_{\rm sd} \;=\; \begin{pmatrix} 3200/202 \\ 0 \end{pmatrix} \;.$$

This is parallel to the $x_1$-axis. The exact line search will stop at a point where the gradient is orthogonal to this. Thus the next search direction will be parallel to the $x_2$-axis, etc. This is illustrated in Figure 2.8, where we represent the successive steepest descent directions by unit vectors, $\boldsymbol{h}_{k+1} = \boldsymbol{h}_{\rm sd}/\|\boldsymbol{h}_{\rm sd}\|$ where $\boldsymbol{h}_{\rm sd} = -\boldsymbol{\nabla} f(\boldsymbol{x}_k)$.

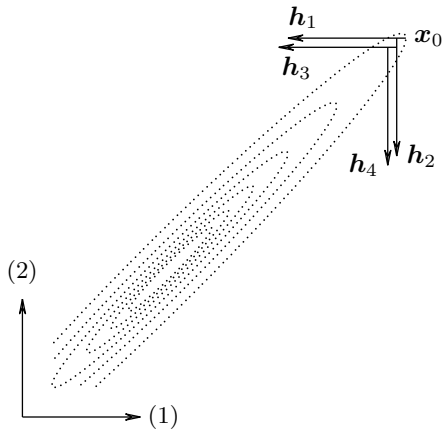



***Figure 2.8.*** *Slow progress of the Steepest Descent method applied a quadratic*

The iteration steps will be exactly as in Example 1.2. If we allow 100 iterations we are still a considerable distance from the minimizer $\hat{\boldsymbol{x}} = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$: $\|\boldsymbol{x}_{100} - \hat{\boldsymbol{x}}\| \simeq 0.379$. It takes 317 iterations to get the error smaller than $5{\cdot}10^{-3}$ and 777 iterations if we want 6 correct decimals. On

a computer with unit roundoff $\varepsilon_{\mathsf{M}} = 2^{-53} \simeq 1.11 \cdot 10^{-16}$ all the $\boldsymbol{x}_k$ are identical for $k \geq 1608$ and the error is approximately $1.26 \cdot 10^{-14}$. ∎

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

The method is useful, however, when we are far from the solution. It performs a little better if we ensure that the steps taken are small enough. In such a version it is included in several modern hybrid methods, where there is a switch between two methods, one with robust global performance and one with superlinear (or even quadratic) final convergence. In this context the Steepest Descent method does a very good job as the "global part" of the hybrid, see Sections 3.2 and 6.2.

## 2.6. Quadratic models

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

$$q(\boldsymbol{x}) \;=\; a + \boldsymbol{b}^T\boldsymbol{x} + \tfrac{1}{2}\,\boldsymbol{x}^T\boldsymbol{H}\,\boldsymbol{x}\;, \tag{2.32}$$

where $\boldsymbol{H}$ is a symmetric matrix. Such a model could for instance be the 2nd order Taylor expansion from the current approximation $\boldsymbol{x}$, but we shall also see other origins of the model. Often the matrix $\boldsymbol{H}$ is required to be positive definite.

Such a model can be used directly or indirectly. In the first case we simply use the minimizer of $q$ to approximate $\hat{\boldsymbol{x}}$ and then repeat the process with a new approximation. This is the basis of the Newton–type methods described in Chapter 3. For the conjugate gradient methods in the next section the model function (2.32) will be employed indirectly.

A related concept is that of *quadratic termination*, which is said to hold for methods that find the exact minimum of the quadratic (2.32) in a finite number of steps. The steepest descent method does not have quadratic termination, but all the methods discussed from the next section do. Quadratic termination has proved to be an important idea and worth striving for in the design of optimization methods.

Because of the importance of quadratic models we now take a closer

look at the quadratic function (2.32). In Theorem 1.7 we showed that its gradient and Hessian at $\boldsymbol{x}$ are

$$\boldsymbol{\nabla} q(\boldsymbol{x}) \;=\; \boldsymbol{H}\,\boldsymbol{x} + \boldsymbol{b}\;, \qquad \boldsymbol{\nabla}^2 q(\boldsymbol{x}) \;=\; \boldsymbol{H}\;. \tag{2.33}$$

Thus, the Hessian is independent of $\boldsymbol{x}$.

If $\boldsymbol{H}$ is positive definite, then $q$ has a unique minimizer,

$$\hat{\boldsymbol{x}} \;=\; -\boldsymbol{H}^{-1}\boldsymbol{b}\;. \tag{2.34}$$

In the case $n=2$ the contours of $q$ are ellipses centered at $\hat{\boldsymbol{x}}$. The shape and orientation of the ellipses are determined by the eigenvalues and eigenvectors of $\boldsymbol{H}$. For $n=3$ this generalizes to ellipsoids, and in higher dimensions we get $(n-1)$-dimensional hyper-ellipsoids. It is of course possible to define quadratic functions with a non-positive definite Hessian, but then there is no longer a unique minimizer.

Finally, a useful fact is derived in a simple way from (2.33): Multiplication by $\boldsymbol{H}$ maps differences in $\boldsymbol{x}$-values to differences in the corresponding gradients:

$$\boldsymbol{H}(\boldsymbol{x}-\boldsymbol{z}) \;=\; \boldsymbol{\nabla} q(\boldsymbol{x}) - \boldsymbol{\nabla} q(\boldsymbol{z})\;. \tag{2.35}$$

## 2.7. Conjugate Gradient methods

Starting with this section we describe methods of practical importance. The conjugate gradient methods are simple and easy to implement, and generally they are superior to the steepest descent method, but Newton's method and its relatives (see the next chapter) are usually even better. If, however, the number $n$ of variables is large, then the conjugate gradient methods may be more efficient than Newton–type methods. The reason is that the latter rely on matrix operations, whereas conjugate gradient methods only use vectors. Ignoring sparsity, Newton's method needs $O(n^3)$ operations per iteration step, Quasi-Newton methods need $O(n^2)$, but the conjugate gradient methods only use $O(n)$ operations per iteration step. Similarly for storage: Newton-type methods require an $n\times n$ matrix to be stored, while conjugate gradient methods only need a few vectors.

The basis for the methods presented in this section is the following definition, and the relevance for our problems is indicated in the next example.

**Definition 2.36. Conjugate directions.** A set of directions corresponding to vectors $\{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots\}$ is said to be *conjugate* with respect to a symmetric positive definite matrix $\boldsymbol{A}$, if

$$\boldsymbol{h}_i^T \boldsymbol{A}\, \boldsymbol{h}_j \;=\; 0 \quad \text{for all} \;\; i \neq j \;.$$

**Example 2.6.** In $\mathbb{R}^2$ we want to find the minimizer of a quadratic

$$q(\boldsymbol{x}) = a + \boldsymbol{b}^T \boldsymbol{x} + \tfrac{1}{2} \boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} \;,$$

where the matrix $\boldsymbol{H}$ is assumed to be positive definite. Figure 2.9 shows contours of such a polynomial.

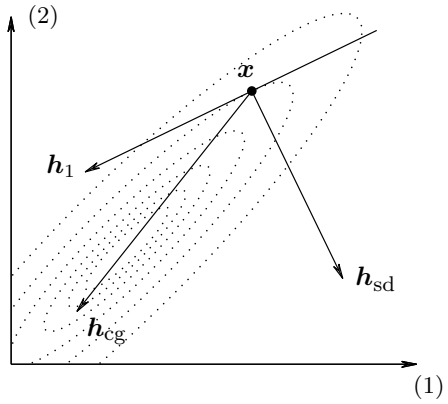



***Figure 2.9.*** *In the 2-dimensional case, the second conjugate gradient step determines the minimizer of a quadratic.*

Remember how Examples 1.2 and 2.5 showed that the methods of Alternating Directions and of Steepest Descent could be caught in Stiefel's cage and fail to find the solution $\hat{\boldsymbol{x}}$. The conjugate gradient method performs better:

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

$$0 \;=\; \boldsymbol{h}_1^T \left(-\boldsymbol{\nabla} q(\boldsymbol{x})\right) \;=\; \boldsymbol{h}_1^T \left(-\boldsymbol{b} - \boldsymbol{H}\, \boldsymbol{x}\right) \;=\; \boldsymbol{h}_1^T\, \boldsymbol{H} \left(\hat{\boldsymbol{x}} - \boldsymbol{x}\right) .$$

In the last reformulation we used the fact that the minimizer satisfies $\boldsymbol{H}\hat{\boldsymbol{x}} = -\boldsymbol{b}$. This shows that if we are at $\boldsymbol{x}$ after an exact line search along a descent direction, $\boldsymbol{h}_1$, then the direction $\hat{\boldsymbol{x}} - \boldsymbol{x}$ to the minimizer is conjugate to $\boldsymbol{h}_1$ with respect to $\boldsymbol{H}$. We can further prove that the conjugate direction is a linear combination of the search direction $\boldsymbol{h}_1$ and the steepest descent direction, $\boldsymbol{h}_{\text{sd}}$, with positive coefficients, ie, it is in the angle between $\boldsymbol{h}_1$ and $\boldsymbol{h}_{\text{sd}}$. ∎

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

### 2.7.1. Structure of a Conjugate Gradient method

Let us have another look at Figure 2.8 where the slow convergence of the steepest descent method is demonstrated. An idea for a possible cure is to generalise the observation from Example 2.6: take a linear combination of the previous search direction and the current steepest descent direction to get a direction toward the solution. This gives a method of the following type.

**Algorithm 2.37. Conjugate gradient method**

**begin**
  $\boldsymbol{x} := \boldsymbol{x}_0$; $k := 0$; $\gamma := 0$; $\boldsymbol{h}_{\mathrm{cg}} := \boldsymbol{0}$; *found* $:= \cdots$ {1°}
  **while** (**not** *found*) **and** $(k < k_{\max})$
    $\boldsymbol{h}_{\mathrm{prev}} := \boldsymbol{h}_{\mathrm{cg}}$; $\boldsymbol{h}_{\mathrm{cg}} := -\boldsymbol{\nabla} f(\boldsymbol{x}) + \gamma * \boldsymbol{h}_{\mathrm{prev}}$
    **if** $\boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{h}_{\mathrm{cg}} \geq 0$ {2°}
      $\boldsymbol{h}_{\mathrm{cg}} := -\boldsymbol{\nabla} f(\boldsymbol{x})$
    $\boldsymbol{x}_{\mathrm{prev}} := \boldsymbol{x}$; $\boldsymbol{x} := \mathrm{line\_search}(\boldsymbol{x}, \boldsymbol{h}_{\mathrm{cg}})$; {3°}
    $\gamma := \cdots$ {4°}
    $k := k+1$; *found* $:= \cdots$ {5°}
**end**

We have the following remarks:

1° Initialization. We recommend to stop if one of the criteria

$$\|\boldsymbol{\nabla} f(\boldsymbol{x})\|_\infty \leq \varepsilon_3 \quad \text{or} \quad k > k_{\max} \tag{2.38}$$

is satisfied, cf (2.10). We may have started at a point $\boldsymbol{x}_0$ where the first of these criteria is satisfied with $\boldsymbol{x}_0$ as a sufficiently good approximation to $\hat{\boldsymbol{x}}$.

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

3° New iterate, cf (2.22).

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

5° Check stopping criteria, cf 1°.

The next theorem shows that a method employing conjugate search directions and exact line search is very good for minimizing quadratics. Theorem 2.41 shows that, if $f$ is a quadratic and we use exact line search, then a proper choice of $\gamma$ gives conjugate search directions.

**Theorem 2.39.** If Algorithm 2.37 with exact line search is applied to a quadratic like (2.32) with $\boldsymbol{x} \in \mathbb{R}^n$, and with the iteration steps $\boldsymbol{h}_i = \boldsymbol{x}_i - \boldsymbol{x}_{i-1}$ corresponding to conjugate directions, then

1° The search directions $\boldsymbol{h}_{\mathrm{cg}}$ are downhill.

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

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

**Proof.** According to Definition 2.12 the inner product between the gradient $\boldsymbol{\nabla} f(\boldsymbol{x})$ and $\boldsymbol{h}_{\mathrm{cg}}$ should be negative:

$$\begin{aligned} \boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{h}_{\mathrm{cg}} &= -\boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{\nabla} f(\boldsymbol{x}) + \gamma \boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{h}_{\mathrm{prev}} \\ &= -\|\boldsymbol{\nabla} f(\boldsymbol{x})\|_2^2 \;\leq\; 0 \;. \end{aligned}$$

In the second reformulation we exploited that the use of exact line search implies that $\boldsymbol{\nabla} f(\boldsymbol{x})$ and $\boldsymbol{h}_{\mathrm{prev}}$ are orthogonal, cf (2.25). Thus, $\boldsymbol{h}_{\mathrm{cg}}$ is downhill (unless $\boldsymbol{x}$ is a stationary point, where $\boldsymbol{\nabla} f(\boldsymbol{x}) = \boldsymbol{0}$), and we have proven 1°.

This result can be expressed as

$$\boldsymbol{h}_i^T \boldsymbol{\nabla} f(\boldsymbol{x}_i) \;=\; 0 \;, \quad i = 1, \ldots, k \;,$$

and by means of (2.35) we see that for $j < k$,

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

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

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

We remark that if $\boldsymbol{\nabla} f(\boldsymbol{x}_k) = \boldsymbol{0}$ for some $k \leq n$, then the solution has been found and Algorithm 2.37 stops.

What remains is to find a clever way to determine $\gamma$. The approach used is to determine $\gamma$ in such a way that the resulting method will work well for minimizing quadratic functions. Taylor's formula shows that smooth functions are locally well approximated by quadratics, and therefore the method can be expected also to work well on more general functions.

### 2.7.2. The Fletcher–Reeves method

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

$$\gamma \;=\; \frac{\boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{\nabla} f(\boldsymbol{x})}{\boldsymbol{\nabla} f(\boldsymbol{x}_{\text{prev}})^T \boldsymbol{\nabla} f(\boldsymbol{x}_{\text{prev}})} \quad , \tag{2.40}$$

where $\boldsymbol{x}_{\text{prev}}$ is the previous iterate. Algorithm 2.37 with this choice for $\gamma$ is called the *Fletcher–Reeves method* after the people who invented it in 1964.

**Theorem 2.41.** If the Fletcher–Reeves method with exact line search is applied to the quadratic function (2.32), and if $\boldsymbol{\nabla} f(\boldsymbol{x}_k) \neq \boldsymbol{0}$ for $k$=1, ..., $n$, then the search directions $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$ are conjugate with respect to $\boldsymbol{H}$.

**Proof.** See Appendix B.1. □

According to Theorem 2.39 this implies that the Fletcher–Reeves method with exact line search used on a quadratic will terminate in at most $n$ steps.

Point 1° in Theorem 2.39 shows that a conjugate gradient method with exact line search produces descent directions. Al-Baali [1] proved that this is also the case for the Fletcher–Reeves method with soft line search satisfying certain mild conditions. We return to this result in Theorem 2.43 below.

### 2.7.3. The Polak–Ribière method

An alternative formula for $\gamma$ is

$$\gamma \;=\; \frac{\left(\boldsymbol{\nabla} f(\boldsymbol{x}) - \boldsymbol{\nabla} f(\boldsymbol{x}_{\text{prev}})\right)^T \boldsymbol{\nabla} f(\boldsymbol{x})}{\boldsymbol{\nabla} f(\boldsymbol{x}_{\text{prev}})^T \boldsymbol{\nabla} f(\boldsymbol{x}_{\text{prev}})} \quad . \tag{2.42}$$

Algorithm 2.37 with this choice of $\gamma$ is called the *Polak–Ribière method*. It dates from 1971 (and again it is named after the inventors).

For a quadratic (2.42) is equivalent to (2.40) (because then $\nabla f(\boldsymbol{x}_{\text{prev}})^T \nabla f(\boldsymbol{x}) = 0$, see (B.6) in Appendix B.1). For general functions, however, the two methods differ, and experience has shown that (2.42) is superior to (2.40). Of course the search directions are still downhill if exact line search is used in the Polak–Ribière Method. For soft line search there is however no result parallel to that of Al-Baali for the Fleetcher–Reeves Method. In fact M.J.D. Powell has constructed an example where the method fails to converge even with exact line search (see [43, page 213]). The success of the Polak–Ribière formula is therefore not so easily explained by theory.

**Example 2.7. (Resetting).** A possibility that has been proposed, is to reset the search direction $\boldsymbol{h}$ to the steepest descent direction $\boldsymbol{h}_{\text{sd}}$ in every $n$th iteration. The rationale behind this is the $n$-step quadratic termination property. If we enter a neighbourhood of the solution where $f$ behaves like a quadratic, resetting will ensure fast convergence. Another apparent advantage of resetting is that it will guarantee global convergence (by Theorem 2.19). However, practical experience has shown that the profit of resetting is doubtful.

In connection with this we remark that the Polak–Ribière method has a kind of resetting built in. Should we encounter a step with very little progress, so that $\|\boldsymbol{x}-\boldsymbol{x}_{\text{prev}}\|$ is small compared with $\|\nabla f(\boldsymbol{x}_{\text{prev}})\|$, then $\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{x}_{\text{prev}})\|$ will also be small and therefore $\gamma$ is small, and $\boldsymbol{h}_{\text{cg}} \simeq \boldsymbol{h}_{\text{sd}}$ in this situation. Also, the modification before the line search in Algorithm 2.37 may result in an occasional resetting. ∎

### 2.7.4. Convergence properties

In Theorem 2.39 we saw that the search directions $\boldsymbol{h}_{\text{cg}}$ of a conjugate gradient method are descent directions and thus the $\theta$ of (2.13) satisfies $\theta<\pi/2$. There is no guarantee, however, that the $\mu$ of Definition 2.14 will stay constant, and Theorem 2.19 is therefore not directly applicable.

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

**Theorem 2.43.** Let the line search used in Algorithm 2.37 satisfy (2.18) with parameter values $0 < \beta_1 < \beta_2 < 0.5$. Then there is a $c > 0$ such that for all $k$

$$\nabla f(\boldsymbol{x})^T \boldsymbol{h}_{\text{cg}} \;\leq\; -c\,\|\nabla f(\boldsymbol{x})\|_2^2$$

and

$$\lim_{k\to\infty} \|\nabla f(\boldsymbol{x})\|_2 \;=\; 0\ .$$

**Proof.** See [1]. □

Finally, it has been shown, [12], that conjugate gradient methods with exact line search have a linear convergence rate, as defined in (1.8). This should be contrasted with the superlinear convergence rate that holds for Quasi–Newton methods and the quadratic convergence rate that Newton's method possesses.

**Example 2.8.** *Rosenbrock's function*,

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

is widely used for testing optimization algorithms. Figure 2.10 shows level curves for this function (and illustrates, why it is sometimes called the *"banana function"*).

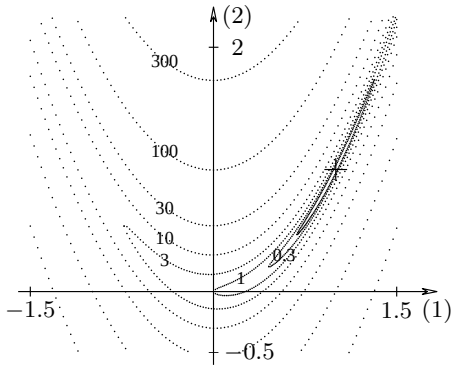



***Figure 2.10.*** *Contours of Rosenbrock's function.*

The function has one minimizer $\hat{\boldsymbol{x}} = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$ (marked by a + in the figure) with $f(\hat{\boldsymbol{x}}) = 0$, and there is a "valley" with sloping bottom following the parabola $x_2 = x_1^2$. Most optimization algorithms will try to follow this valley. Thus, a considerable amount of iterations is needed, if we take the starting point $\boldsymbol{x}_0$ in the 2nd quadrant.

Below we give the number of iteration steps and evaluations of $f(\boldsymbol{x})$ and $\nabla f(\boldsymbol{x})$ when applying Algorithm 2.37 on this function. In all cases we

use the starting point $\boldsymbol{x}_0 = \begin{pmatrix} -1.2 & 1 \end{pmatrix}^T$, and stopping criteria given by $\varepsilon_1 = 10^{-8}$, $\varepsilon_2 = 10^{-12}$ in (2.38). In case of exact line search we use $\tau = \varepsilon = 10^{-6}$ in (2.24), while we take $\beta_1 = 0.01$, $\beta_2 = 0.1$ in Algorithm 2.20 for soft line search.

| Method | Line search | # iterations | # fct. evals |
|---|---|---|---|
| Fletcher–Reeves | exact | 343 | 2746 |
| Fletcher–Reeves | soft | 81 | 276 |
| Polak–Ribière | exact | 18 | 175 |
| Polak–Ribière | soft | 41 | 127 |

Thus, in this case the Polak–Ribière method with soft line search performs best. The performance is illustrated in Figure 2.11.

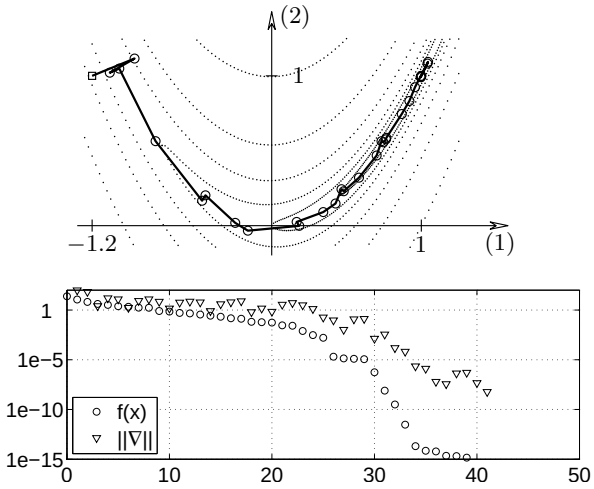



***Figure 2.11.*** *Polak–Ribière method with soft line search applied to Rosenbrock's function.*
*Top: iteration path.*
*Bottom: $f(\boldsymbol{x}_k)$ and $\|\boldsymbol{\nabla} f(\boldsymbol{x}_k)\|_\infty$.*
*Note the logarithmic ordinate axis.* ∎

### 2.7.5. Implementation aspects

To implement a conjugate gradient algorithm in a computer program, some decisions must be made.

First, we must choose a formula for $\gamma$. We recommend the Polak–Ribière formula.

Next, we need to specify the exactness of the line search. For Newton-type methods it is usually recommended that the line search be quite liberal, so for the line search in Algorithm 2.20 it is common to choose the parameter values $\beta_1 = 0.001$ and $\beta_2 = 0.99$. For conjugate gradient

methods experience indicates that a line search with stricter tolerances should be used, say $\beta_1 = 0.01$ and $\beta_2 = 0.1$.

We also have to specify the stopping criteria, and recommend to use 3° and 4° in (2.10). For methods with a fast convergence rate criterion 1° may be quite satisfactory, but its use for conjugate gradient methods must be discouraged because their final convergence rate is only linear.

Finally some remarks on the storage of vectors. The Fletcher–Reeves method may be implemented using three $n$-vectors of storage, `x`, `g` and `h`. If these contain $\boldsymbol{x}$, $\boldsymbol{\nabla} f(\boldsymbol{x})$ and $\boldsymbol{h}_{\text{prev}}$ at the beginning of the current iteration step, we may overwrite `h` with $\boldsymbol{h}_{\text{cg}}$ and during the line search we overwrite `x` with $\boldsymbol{x}+\alpha\boldsymbol{h}_{\text{cg}}$ and `g` with $\boldsymbol{\nabla} f(\boldsymbol{x}+\alpha\boldsymbol{h}_{\text{cg}})$. Before overwriting the gradient, we find $\boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{\nabla} f(\boldsymbol{x})$ for use in the denominator in (2.40) for the next iteration. For the Polak–Ribière method we need access to $\boldsymbol{\nabla} f(\boldsymbol{x})$ and $\boldsymbol{\nabla} f(\boldsymbol{x}_{\text{prev}})$ simultaneously, and thus four vectors are required, say `x`, `g`, `gnew` and `h`.

### 2.7.6. Other methods and further reading

Over the years numerous other conjugate gradient formulae and amendments to the Fletcher–Reeves and Polak–Ribière method have been proposed. We only give a short summary here, and refer the interested reader to [19] and [43] for details and further information.

A possible amendment to the Polak–Ribière method is to choose $\gamma = \max(\gamma_{\text{PR}}, 0)$, where $\gamma_{\text{PR}}$ is the $\gamma$ of (2.42). With this choice of $\gamma$ it is possible to guarantee global convergence with inexact line search. See [43, page 213] for further discussion and references.

The conjugate gradient methods belong to a class of methods sometimes referred to as conjugate direction methods. Other examples of these may be found in [19].

Finally we want to mention two classes of methods that have received much attention in recent years. The first class is called limited memory Quasi-Newton methods, and the second class is truncated Newton methods or inexact Newton methods. These are not conjugate direction methods, but they are also aimed at solving large problems. See [43, pp 233–234] for some discussion and further references.

### 2.7.7. The CG method for linear systems

Conjugate gradient methods are widely used to solve linear systems of equations,

$$\boldsymbol{A}\,\boldsymbol{x} \;=\; \boldsymbol{b}\,,$$

where $\boldsymbol{A}$ is a large, sparse, symmetric and positive matrix. In the present context, cf (2.34), the solution $\hat{\boldsymbol{x}}$ to this system is the minimizer of the quadratic

$$q(\boldsymbol{x}) \;=\; a - \boldsymbol{b}^T\boldsymbol{x} + \tfrac{1}{2}\,\boldsymbol{x}^T\boldsymbol{A}\,\boldsymbol{x}\ .$$

Let $\boldsymbol{x}$ denote the current approximation to $\hat{\boldsymbol{x}}$, and let $\boldsymbol{r}$ denote the corresponding negative gradient

$$\boldsymbol{r} \;\equiv\; -\boldsymbol{\nabla} q(\boldsymbol{x}) \;=\; \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}\ .$$

We recognize this as the *residual vector* corresponding to the approximation $\boldsymbol{x}$, and the search direction can be expressed as

$$\boldsymbol{h}_{\mathrm{cg}} \;=\; \boldsymbol{r} + \gamma\,\boldsymbol{h}_{\mathrm{prev}}\ .$$

The residual is an affine function of $\boldsymbol{x}$, so that

$$\boldsymbol{x} \;=\; \boldsymbol{x}_{\mathrm{prev}} + \alpha\boldsymbol{h}_{\mathrm{cg}} \quad\Rightarrow\quad \boldsymbol{r} \;=\; \boldsymbol{r}_{\mathrm{prev}} - \alpha\boldsymbol{A}\,\boldsymbol{h}_{\mathrm{cg}}\ .$$

The special form of the problem also implies that instead of having to make a proper exact line search we have a closed form expression for $\alpha = \alpha_{\mathrm{e}}$: The search direction and the gradient are orthogonal at the minimizer of the line search function, $\boldsymbol{h}_{\mathrm{cg}}^T\boldsymbol{r} = 0$, so that

$$\alpha \;=\; \big(\boldsymbol{h}_{\mathrm{cg}}^T\boldsymbol{r}\big)\big/\,\big(\boldsymbol{h}_{\mathrm{cg}}^T\boldsymbol{A}\,\boldsymbol{h}_{\mathrm{cg}}\big)\ . \tag{2.44}$$

Actually, the present problem was the subject of Theorem 2.41, and in the proof we show that not only are the search directions conjugate with respect to $\boldsymbol{A}$, but according to (B.5) – (B.6) we have the following orthogonalities

$$\boldsymbol{r}_i^T\boldsymbol{r}_j \;=\; 0 \quad\text{and}\quad \boldsymbol{h}_i^T\boldsymbol{r}_j \;=\; 0\text{for}\ \ i \neq j\ .$$

This implies that the Fletcher–Reeves and the Polak–Ribière formulas give identical values $\gamma = \boldsymbol{r}^T\boldsymbol{r}/{\boldsymbol{r}_{\mathrm{prev}}}^T\boldsymbol{r}_{\mathrm{prev}}$. Further, since $\boldsymbol{h}_{\mathrm{cg}} = \boldsymbol{r} - \gamma\boldsymbol{h}_{\mathrm{prev}}$, we see that (2.44) can be replaced by $\alpha = \boldsymbol{r}^T\boldsymbol{r}/(\boldsymbol{h}_{\mathrm{cg}}^T\boldsymbol{A}\,\boldsymbol{h}_{\mathrm{cg}})$.

The theorem further tells us that we need at most $n$ iterations before $\boldsymbol{r} = \boldsymbol{0}$, ie $\boldsymbol{x} = \hat{\boldsymbol{x}}$. However, in applications it is often sufficient to find an approximation $\boldsymbol{x}$ such that

$$\|\boldsymbol{r}(\boldsymbol{x})\| \;\le\; \varepsilon\|\boldsymbol{b}\|\ ,$$

where $\varepsilon$ is a small, positive number specified by the user. Note that this stopping criterion is of the type 3° in the list of stopping criteria given in (2.10).

Summarizing this discussion we can present the following specialized version of the general CG algorithm in 2.37.

**Algorithm 2.45. CG method for linear systems**

**begin**
  $\boldsymbol{x} := \boldsymbol{x}_0$;  $k := 0$;  $\boldsymbol{r} := \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$
  $u = \boldsymbol{r}^T\boldsymbol{r}$;  *found* $:= \left(u \leq \varepsilon^2\|\boldsymbol{b}\|^2\right)$
  **while** (**not** *found*) **and** $(k < k_{\max})$
    **if** $k = 0$
      $\boldsymbol{h}_{\rm cg} := \boldsymbol{r}$
    **else**
      $\boldsymbol{h}_{\rm cg} := \boldsymbol{r} + (u/u_{\rm prev})\boldsymbol{h}_{\rm cg}$
    **end**
    $\boldsymbol{v} := \boldsymbol{A}\,\boldsymbol{h}_{\rm cg}$;  $\alpha := u/(\boldsymbol{h}_{\rm cg}^T\boldsymbol{v})$
    $\boldsymbol{x} := \boldsymbol{x} + \alpha\,\boldsymbol{h}_{\rm cg}$;  $\boldsymbol{r} := \boldsymbol{r} - \alpha\,\boldsymbol{v}$
    $u_{\rm prev} := u$;  $u := \boldsymbol{r}^T\boldsymbol{r}$
    $k := k{+}1$;  *found* $:= \left(u \leq \varepsilon^2\|\boldsymbol{b}\|^2\right)$
**end**

In an implementation of the method we need four $n$-vectors, `x`, `r`, `h` and `v`. Each iteration involves one matrix-vector multiplication, to get $\boldsymbol{v}$, and 5 vector operations.

The vectors $\boldsymbol{r}_0, \boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ belong to the so-called *Krylov subspace*

$$\mathbb{K} \;=\; \text{span}\left(\boldsymbol{r}_0, \boldsymbol{A}\,\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \ldots, \boldsymbol{A}^k\boldsymbol{r}_0\right) \;,$$

and the CG method is said to be an iterative Krylov method. It is not the only one, however, and in recent years there has been much interest in the further improvement of this class of very efficient methods for approximate solution of large, sparse systems of equations. For a more thorough discussion of CG methods for linear systems we refer to [23, Chapter 10] and the monographs [24], [54].

# Chapter 3

# Newton–Type Methods

The main goal of this chapter is to present a class of methods for unconstrained optimization which are based on Newton's method. This class is called Quasi-Newton[1)] methods. This class includes some of the best methods on the market for solving the unconstrained optimization problem.

In order to explain these methods we first describe Newton's method for unconstrained optimization in detail. Newton's method leads to another type of methods known as Damped Newton methods, which will also be presented.

## 3.1. Newton's method

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

**Example 3.1.** In Examples 1.2 and 2.5 we saw the methods of alternating directions and steepest descent fail to find the minimizer of a simple quadratic in two dimensions. In Section 2.7 we saw that the conjugate gradient method finds the minimizer of a quadratic in $n$ steps ($n$ being the dimension of the space), in two steps in Example 2.6.

Newton's method can find the minimizer of a quadratic in $n$-dimensional space in one step. This follows from equation (3.2) below. Figure 3.1 gives the contours of our 2-dimensional quadratic together with (an arbitrary) $\boldsymbol{x}_0$, the corresponding $\boldsymbol{x}_1$ and the minimizer $\hat{\boldsymbol{x}}$, marked by $+$.

---

[1)] From Latin, "quasi" = "nearly" (or "wanna be")

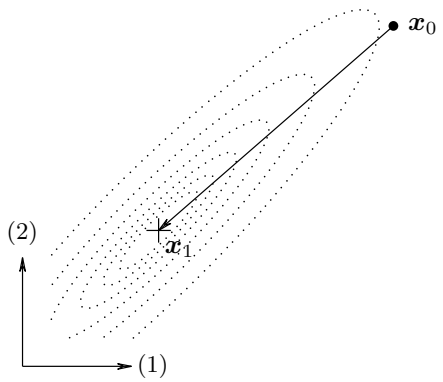



***Figure 3.1.*** *Newton's method finds the minimizer of a quadratic in one step.* ∎

The version of *Newton's method* used in optimization is derived from the 2nd order Taylor approximation from Theorem 1.6. This means that in the neighbourhood of the current iterate $\boldsymbol{x}$ we use the quadratic model

$$\begin{aligned} f(\boldsymbol{x}+\boldsymbol{h}) &\simeq q(\boldsymbol{h}) \\ &\equiv f(\boldsymbol{x}) + \boldsymbol{h}^T\nabla f(\boldsymbol{x}) + \tfrac{1}{2}\boldsymbol{h}^T\nabla^2 f(\boldsymbol{x})\,\boldsymbol{h}\ . \end{aligned} \tag{3.1}$$

The idea now is to find the next iterate as a minimizer for the model $q$. If $\nabla^2 f(\boldsymbol{x})$ is positive definite, then $q$ has a unique minimizer at a point where its gradient equals zero, ie where

$$\nabla q(\boldsymbol{h}) \;=\; \nabla f(\boldsymbol{x}) + \nabla^2 f(\boldsymbol{x})\,\boldsymbol{h} \;=\; \boldsymbol{0}\ . \tag{3.2}$$

Hence, in Newton's method the step to the next iterate is obtained as the solution to this system, as shown in the following algorithm.

**Algorithm 3.3. Newton's method**

**begin**
  $\boldsymbol{x} := \boldsymbol{x}_0$; {initialize}
  **repeat**
    Solve $\nabla^2 f(\boldsymbol{x})\,\boldsymbol{h}_{\rm n} \;=\; -\nabla f(\boldsymbol{x})$ {find step}
    $\boldsymbol{x} := \boldsymbol{x} + \boldsymbol{h}_{\rm n}$ {next iterate}
  **until** stopping criteria satisfied
**end**

Newton's method is well defined as long as $\nabla^2 f(\boldsymbol{x})$ remains non-singular. Also, if the Hessian is positive definite, then it follows from

Theorem 2.15 that $\boldsymbol{h}_{\mathrm{n}}$ is a descent direction. Further, if $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ stays positive definite in all the steps and if the starting point is sufficiently close to a minimizer, then the method usually converges rapidly towards such a solution. More precisely the following theorem holds.

**Theorem 3.4.** Let $f$ be three times continuously differentiable and let $\boldsymbol{x}$ be sufficiently close to a local minimizer $\hat{\boldsymbol{x}}$ with positive definite $\boldsymbol{\nabla}^2 f(\hat{\boldsymbol{x}})$. Then Newton's method is well defined in all the following steps, and it converges quadratically towards $\hat{\boldsymbol{x}}$.

**Proof.** See Appendix B.2. □

**Example 3.2.** We shall use Newton's method to find the minimizer of the function[2)]

$$\begin{aligned} f(\boldsymbol{x}) \;&=\; 0.5 * x_1^2 * (x_1^2/6 + 1) \\ &\quad + x_2 * \mathrm{Arctan}(x_2) - 0.5 * \log\,(x_2^2 + 1)\ . \end{aligned} \tag{3.5}$$

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

$$\boldsymbol{\nabla} f(\boldsymbol{x}) \;=\; \begin{pmatrix} x_1^3/3 + x_1 \\ \mathrm{Arctan}(x_2) \end{pmatrix}\ , \qquad \boldsymbol{\nabla}^2 f(\boldsymbol{x}) \;=\; \begin{pmatrix} x_1^2 + 1 & 0 \\ 0 & 1/(1 + x_2^2) \end{pmatrix}\ .$$

We can see in Figure 3.2 that in a region around the minimizer $\hat{\boldsymbol{x}} = \boldsymbol{0}$ the function looks very well-behaved and extremely simple to minimize.

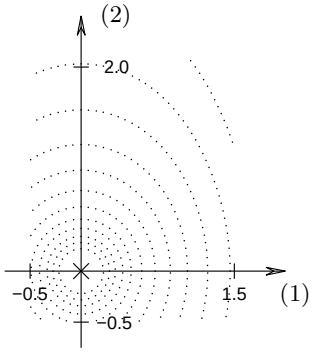



***Figure 3.2.*** *Contours of the function (3.5). The level curves are symmetric across both axes.*

Table 3.2 gives results of the iterations with the starting point $\boldsymbol{x}_0 = \begin{pmatrix} 1 & 0.7 \end{pmatrix}^T$. According to Theorem 3.4 we expect quadratic convergence. If the factor $c_2$ in (1.8) is of the order of magnitude 1, then the column of $\boldsymbol{x}_k^T$ would show the number of correct digits doubled in each iteration step, and the $f$-values and step lengths would be squared in each iteration step. The convergence is faster than this; actually for any starting point $\boldsymbol{x}_0 = \begin{pmatrix} u & v \end{pmatrix}^T$ with $|v| < 1$ we will get *cubic convergence*; see the end of the next example.

[2)] "log" is the natural (or Naperian) logarithm.

| $k$ | $\boldsymbol{x}_k^T$ | $f$ | $\|\boldsymbol{\nabla} f\|$ | $\|\boldsymbol{h}_k\|$ |
|---|---|---|---|---|
| 0 | [1.0000000000, 0.7000000000] | 8.11e-01 | 1.47e+00 | 1.13e+00 |
| 1 | [0.3333333333, -0.2099816869] | 7.85e-02 | 4.03e-01 | 3.79e-01 |
| 2 | [0.0222222222, 0.0061189580] | 2.66e-04 | 2.31e-02 | 2.30e-02 |
| 3 | [0.0000073123, -0.0000001527] | 2.67e-11 | 7.31e-06 | 7.31e-06 |
| 4 | [0.0000000000, 0.0000000000] | 3.40e-32 | 2.61e-16 | 2.61e-16 |

***Table 3.2.*** *Newton's method applied to (3.5).* $\boldsymbol{x}_0 = \begin{pmatrix} 1 & 0.7 \end{pmatrix}^T$. ∎

Until now, everything said about Newton's method is very promising: It is simple and if the conditions of Theorem 3.4 are satisfied, then the rate of convergence is excellent. Nevertheless, due to a series of drawbacks the basic version of the method is not suitable for a general purpose optimization algorithm.

The first and by far the most severe drawback is the method's lack of global convergence.

**Example 3.3.** With the starting point $\boldsymbol{x}_0 = \begin{pmatrix} 1 & 2 \end{pmatrix}^T$ the Newton method behaves very badly:

| $k$ | $\boldsymbol{x}_k^T$ | $f$ | $\|\boldsymbol{\nabla} f\|$ | $\|\boldsymbol{h}_k\|$ |
|---|---|---|---|---|
| 0 | [1.0000000000, 2.0000000000] | 1.99e+00 | 1.73e+00 | 5.58e+00 |
| 1 | [0.3333333333, -3.5357435890] | 3.33e+00 | 1.34e+00 | 1.75e+01 |
| 2 | [0.0222222222, 13.9509590869] | 1.83e+01 | 1.50e+00 | 2.93e+02 |
| 3 | [0.0000073123, -2.793441e+02] | 4.32e+02 | 1.57e+00 | 1.22e+05 |
| 4 | [0.0000000000, 1.220170e+05] | 1.92e+05 | 1.57e+00 | 2.34e+10 |

***Table 3.3.*** *Newton's method applied to (3.5).* $\boldsymbol{x}_0 = \begin{pmatrix} 1 & 2 \end{pmatrix}^T$.

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

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

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

∎

The next point to discuss is that $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ may not be positive definite unless $\boldsymbol{x}$ is close to the solution. This means that the Newton sequence may head towards a saddle point or a maximizer, because the iteration process is identical to the one used for solving the nonlinear system of equations $\boldsymbol{\nabla} f(\boldsymbol{x}) = \boldsymbol{0}$. Any stationary point of $f$ is a solution to this system. Also, $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ may be ill-conditioned or singular so that the linear system (3.2) cannot be solved without considerable errors in $\boldsymbol{h}_{\text{n}}$. Such

ill-conditioning may be detected by a well designed matrix factorization (for instance a Cholesky factorization as described in Appendix A.2), but it still leaves the question of what to do in case ill-conditioning occurs.

The need for second order derivatives may also be a major drawback for Newton's method. They must either be provided by the user or they may be obtained by *automatic differentiation*. In the first case there is a serious risk of errors in the derivation or implementation, leading to malfunctioning of the optimization algorithm. We shall not discuss the other choice, but refer to for instance [9]. With both choices the evaluation of the $\frac{1}{2}n(n+1)$ different elements in $\boldsymbol{\nabla}^2 f$ may be a costly affair if $n$ is large.

This discussion of advantages and disadvantages of Newton's method is summarized in Table 3.3, with the purpose of pointing out properties to be retained and areas where improvements and modifications are required.

**Summary 3.6. Advantages and disadvantages of Newton's method for unconstrained optimization problems**

**Advantages**

1° Quadratically convergent from a good starting point if $\boldsymbol{\nabla}^2 f(\hat{\boldsymbol{x}})$ is positive definite.

2° Simple and easy to implement.

**Disadvantages**

1° Not globally convergent for many problems.

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

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

4° Requires second order derivatives of $f$.

## 3.2. Damped Newton methods

A simple idea to overcome the lack of robustness, disadvantage 1° in Summary 3.6, is to combine Newton's method with line search in which the Newton step $\boldsymbol{h}_{\rm n} = -\big[\boldsymbol{\nabla}^2 f(\boldsymbol{x})\big]^{-1}\boldsymbol{\nabla} f(\boldsymbol{x})$ is used as search direction. In other words, the step $\boldsymbol{x} := \boldsymbol{x}+\boldsymbol{h}_{\rm n}$ in Algorithm 3.3 is replaced by

$$\boldsymbol{x} := \text{line\_search}(\boldsymbol{x}, \boldsymbol{h}_{\rm n}) \ .$$

Provided that $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is positive definite, this method will work fine, since in this case $\boldsymbol{h}_{\rm n}$ is a descent direction, cf Theorem 2.15. The main difficulty arises when $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is not positive definite. The Newton step can still be computed if $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is non-singular, and one may search along $\pm\boldsymbol{h}_{\rm n}$ with the sign chosen in each iteration to ensure a descent direction. However, this rather primitive approach is questionable since the quadratic model $q(\boldsymbol{h})$, (3.1), will not even possess a unique minimum.

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

In this section we shall present such a method. In the typical iteration we find the step $\boldsymbol{h}_{\rm dn}$ as the minimizer of the quadratic

$$\begin{aligned} q_\mu(\boldsymbol{h}) &= q(\boldsymbol{h}) + \tfrac{1}{2}\mu\, \boldsymbol{h}^T\boldsymbol{h} \\ &= f(\boldsymbol{x}) + \boldsymbol{h}^T\boldsymbol{\nabla} f(\boldsymbol{x}) + \tfrac{1}{2}\boldsymbol{h}^T(\boldsymbol{\nabla}^2 f(\boldsymbol{x}) + \mu\boldsymbol{I})\boldsymbol{h} \ , \end{aligned} \tag{3.7}$$

where $\mu \geq 0$, so that large steps are penalized. The gradient of this quadratic is

$$\boldsymbol{\nabla} q_\mu(\boldsymbol{h}) \ = \ \boldsymbol{\nabla} f(\boldsymbol{x}) + (\boldsymbol{\nabla}^2 f(\boldsymbol{x}) + \mu\, \boldsymbol{I})\boldsymbol{h} \ ,$$

and by setting this gradient equal to zero we get the vector $\boldsymbol{h}_{\rm dn}$ as the solution to the linear system

$$(\boldsymbol{\nabla}^2 f(\boldsymbol{x}) + \mu\, \boldsymbol{I})\boldsymbol{h}_{\rm dn} \ = \ -\boldsymbol{\nabla} f(\boldsymbol{x}) \ .$$

By comparison with (3.2) we see that if $\mu = 0$, then $\boldsymbol{h}_{\rm dn} = \boldsymbol{h}_{\rm n}$, and if $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is positive definite, then this is a downhill step for $f$, and it is the minimizer for $q$.

In Appendix A.2 we show that if we choose $\mu$ sufficiently large, then the matrix $\boldsymbol{\nabla}^2 f(\boldsymbol{x}) + \mu \boldsymbol{I}$ will be positive definite, and then the step $\boldsymbol{h}_{\rm dn}$ is downhill according to Theorem 2.15. If $\mu$ is very big, then we get

$$\boldsymbol{h}_{\rm dn} \;\simeq\; -\frac{1}{\mu}\,\boldsymbol{\nabla} f(\boldsymbol{x}) \;,$$

ie a short step in the steepest descent direction. As discussed earlier, this is useful in the early stages of the iteration process, if the current $\boldsymbol{x}$ is far from the minimizer $\hat{\boldsymbol{x}}$. On the other hand, if $\mu$ is small, then $\boldsymbol{h}_{\rm dn} \simeq \boldsymbol{h}_{\rm n}$, the Newton step, which is good when we are close to $\hat{\boldsymbol{x}}$ (where $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is positive definite). Thus, by proper adjustment of the damping parameter $\mu$ we have a method that combines the good qualities of the steepest descent method in the global part of the iteration process with the fast ultimate convergence of Newton's method.

This discussion is summarized in the following framework for a damped Newton method.

**Algorithm 3.8. Damped Newton method**

**begin**
  $\boldsymbol{x} := \boldsymbol{x}_0$; $\quad \mu := \mu_0$ {initialize}
  **repeat**
    Solve $\quad (\boldsymbol{\nabla}^2 f(\boldsymbol{x}) + \mu \boldsymbol{I})\, \boldsymbol{h}_{\rm dn} \;=\; -\boldsymbol{\nabla} f(\boldsymbol{x})$ {find step}
    $\boldsymbol{x} := \boldsymbol{x} + \alpha \boldsymbol{h}_{\rm dn}$ {next iterate}
    Adjust $\mu$
  **until** stopping criteria satisfied
**end**

There are several variants of this algorithms, differing in the way that $\alpha$ is chosen and $\mu$ is updated during the process. In some variants the parameter $\alpha$ is found by line search, and information gathered during this may be used to update $\mu$. Other variants exploit that $\mu$ has two functions: it influences both the direction and the length of the step $\boldsymbol{h}_{\rm dn}$, and if $f(\boldsymbol{x}+\boldsymbol{h}_{\rm dn}) < f(\boldsymbol{x})$, then $\boldsymbol{x}+\boldsymbol{h}_{\rm dn}$ is our new iterate, ie $\alpha = 1$. Otherwise we let $\alpha = 0$ but increase $\mu$. These so-called Levenberg–Marquardt type variants seem to be the most successful, and we shall give some details later.

It is also possible to use a trust region approach, cf Section 2.4. See for instance [19] or [44]. An interesting relation between a trust region approach and Algorithm 3.8 is given in the following theorem, which was first given by Marquardt (1963), [36].

**Theorem 3.9.** If the matrix $\boldsymbol{\nabla}^2 f(\boldsymbol{x})+\mu\boldsymbol{I}$ is positive definite, then

$$\boldsymbol{h}_{\rm dn} \;=\; \underset{\|\boldsymbol{h}\| \le \|\boldsymbol{h}_{\rm dn}\|}{\rm argmin} \;\{q(\boldsymbol{h})\}\;,$$

where $q$ is given by (3.1)) and $\boldsymbol{h}_{\rm dn}$ is defined in Algorithm 3.8.

**Proof.** The vector $\boldsymbol{h}_{\rm dn}$ is the minimizer of $q_\mu(\boldsymbol{h})$, (3.5). Now let

$$\boldsymbol{h}_{\rm tr} \;=\; {\rm argmin}_{\|\boldsymbol{h}\|\le\|\boldsymbol{h}_{\rm dn}\|}\{q(\boldsymbol{h})\}\;.$$

Then $q(\boldsymbol{h}_{\rm tr}) \le q(\boldsymbol{h}_{\rm dn})$ and $\boldsymbol{h}_{\rm tr}^T\boldsymbol{h}_{\rm tr} \le \boldsymbol{h}_{\rm dn}^T\boldsymbol{h}_{\rm dn}$, so that

$$\begin{aligned} q_\mu(\boldsymbol{h}_{\rm tr}) \;&=\; q(\boldsymbol{h}_{\rm tr}) + \tfrac{1}{2}\,\mu\;\boldsymbol{h}_{\rm tr}^T\boldsymbol{h}_{\rm tr} \\ &\le\; q(\boldsymbol{h}_{\rm dn}) + \tfrac{1}{2}\,\mu\;\boldsymbol{h}_{\rm dn}^T\boldsymbol{h}_{\rm dn} \;=\; q_\mu(\boldsymbol{h}_{\rm dn})\;. \end{aligned}$$

However, the matrix $\boldsymbol{\nabla}^2 f(\boldsymbol{x})+\mu\boldsymbol{I}$ is assumed to be positive definite. This implies that the minimizer of $q_\mu$ is unique, and therefore $\boldsymbol{h}_{\rm tr} = \boldsymbol{h}_{\rm dn}$. □

In a proper trust region method we monitor the trust region radius $\Delta$. The theorem shows that if we monitor the damping parameter instead, we can think of it as a trust region method with the trust region radius given implicitly as $\Delta = \|\boldsymbol{h}_{\rm dn}\|$.

In *Levenberg–Marquardt* type methods $\mu$ is updated in each iteration step. Given the present value of the parameter, the Cholesky factorization of $\boldsymbol{\nabla}^2 f(\boldsymbol{x})+\mu\boldsymbol{I}$ is employed to check for positive definiteness, and $\mu$ is increased if the matrix is not significantly positive definite. Otherwise, the solution $\boldsymbol{h}_{\rm dn}$ is easily obtained via the factorization.

The direction given by $\boldsymbol{h}_{\rm dn}$ is sure to be downhill, and we get the "trial point" $\boldsymbol{x}+\boldsymbol{h}_{\rm dn}$ (corresponding to $\alpha = 1$ in Algorithm 3.8). As in a trust region method (see Section 2.4) we can compare the changes in the objective function $f$ and the current approximation to it, $q$. More specifically, we compute the *gain ratio*

$$\varrho \;=\; \frac{f(\boldsymbol{x}) - f(\boldsymbol{x}+\boldsymbol{h}_{\rm dn})}{q(\boldsymbol{0}) - q(\boldsymbol{h}_{\rm dn})}\;. \tag{3.10}$$

A $\varrho$-value close to one indicates that $q$ was a good model and we may reduce $\mu$, ie the penalty on large steps. This may also be a good idea whenever $\varrho > 1$, because the gain in $f$ was at least as large as the gain predicted by $q$. Note that a decrease in $\mu$ not only allows larger steps, but also turns the direction of $\boldsymbol{h}_{\rm dn}$ towards the direction given by $\boldsymbol{h}_{\rm n}$, the

Newton step, which gives quadratic convergence when $\boldsymbol{x}$ is sufficiently close to $\hat{\boldsymbol{x}}$.

If, on the other hand, $\varrho$ is small, we should increase $\mu$ with the twofold objective of reducing the step length and turning it towards the steepest descent direction, which is good when $\boldsymbol{x}$ is far from $\hat{\boldsymbol{x}}$.

We could use an updating strategy similar to the one employed in Algorithm 2.30,

$$\begin{aligned} &\textbf{if} \;\; \varrho > 0.75 \;\; \textbf{then} \;\; \mu := \mu/3 \\ &\textbf{if} \;\; \varrho < 0.25 \;\; \textbf{then} \;\; \mu := \mu * 2 \end{aligned} \tag{3.11}$$

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

$$\begin{aligned} \textbf{if} \;\; \varrho > 0 \;\; &\textbf{then} \;\; \mu := \mu * \max\{\tfrac{1}{3}, 1 - (2\varrho - 1)^3\} \\ &\textbf{else} \quad \mu := \mu * 2 \end{aligned} \tag{3.12}$$

The two strategies are illustrated in Figure 3.3 and are further discussed in [40] and in Section 6.2.

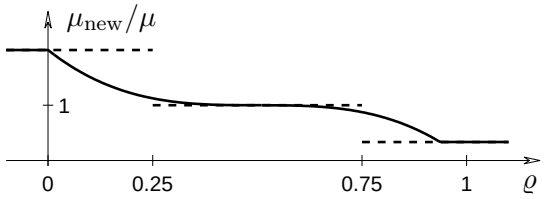



***Figure 3.3.*** *Updating of $\mu$ by (3.11) (dasheded line) and by (3.12) (full line).*

The initial value for $\mu$ should be chosen with respect to the size of the elements in the initial Hessian. We recommend to let

$$\mu_0 \;=\; \tau \, \|\boldsymbol{\nabla}^2 f(\boldsymbol{x}_0)\|_\infty \;, \tag{3.13}$$

where $\tau > 0$ is chosen by the user. If $\tau \geq 1$, then the initial matrix $\boldsymbol{\nabla}^2 f(\boldsymbol{x}_0) + \mu_0 \boldsymbol{I}$ is guaranteed to be positive definite.

In order to prove convergence for the whole procedure one should use an $\boldsymbol{x}$-updating of the form

$$\textbf{if} \;\; \varrho > \delta \;\; \textbf{then} \;\; \boldsymbol{x} := \boldsymbol{x} + \boldsymbol{h}_{\text{dn}} \;,$$

where $\delta$ is a small positive number, $\delta = 10^{-3}$, say. This implies that if $\varrho \le \delta$, then $\boldsymbol{x}$ is not changed ($\alpha = 0$ in Algorithm 3.8), but $\mu$ will be increased. In practice $\delta = 0$ is often used.

The method is summarized in Algorithm 3.15 below. We recommend to use the stopping criteria

$$\|\boldsymbol{\nabla} f(\boldsymbol{x})\|_\infty \le \varepsilon_1 \quad \text{or} \quad \|\boldsymbol{h}_{\rm dn}\|_2 \le \varepsilon_2(\varepsilon_2 + \|\boldsymbol{x}\|_2) \; . \tag{3.14}$$

**Algorithm 3.15. Levenberg–Marquardt type damped Newton method**

**begin**
  $\boldsymbol{x} := \boldsymbol{x}_0$; $\mu := \mu_0$; *found* := **false**; $k := 0$; {initialize}
  **repeat**
    **while** $\boldsymbol{\nabla}^2 f(\boldsymbol{x}) + \mu \boldsymbol{I}$ not pos. def. {using ...}
      $\mu := 2\mu$
    Solve $\left(\boldsymbol{\nabla}^2 f(\boldsymbol{x}) + \mu \boldsymbol{I}\right) \boldsymbol{h}_{\rm dn} = -\boldsymbol{\nabla} f(\boldsymbol{x})$ {... Cholesky}
    Compute gain factor $\varrho$ by (3.10)
    **if** $\varrho > \delta$ {$f$ decreases}
      $\boldsymbol{x} := \boldsymbol{x} + \boldsymbol{h}_{\rm dn}$ {new iterate}
      $\mu := \mu * \max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}$ {new $\mu$}
    **else**
      $\mu := \mu * 2$ {increase $\mu$ but keep $\boldsymbol{x}$}
    $k := k+1$; update *found* {see (3.14)}
  **until** *found* **or** $k > k_{\max}$
**end**

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

**Example 3.4.** Table 3.4 illustrates the performance of Algorithm 3.15 when applied to the tricky function (3.5) with the poor starting point. We use $\eta = 0.5$ in (3.13) and $\varepsilon_1 = 10^{-8}$, $\varepsilon_2 = 10^{-12}$ in (3.14).

| $k$ | $\boldsymbol{x}_k^T$ | $f$ | $\|\boldsymbol{\nabla} f\|_\infty$ | $\mu$ |
|---|---|---|---|---|
| 0 | [ 1.00000000, 2.00000000] | 1.99e+00 | 1.33e+00 | 1.00e+00 |
| 1 | [ 0.55555556, 1.07737607] | 6.63e-01 | 8.23e-01 | 3.33e-01 |
| 2 | [ 0.18240045, 0.04410287] | 1.77e-02 | 1.84e-01 | 1.96e-01 |
| 3 | [ 0.03239405, 0.00719666] | 5.51e-04 | 3.24e-02 | 6.54e-02 |
| 4 | [ 0.00200749, 0.00044149] | 2.11e-06 | 2.01e-03 | 2.18e-02 |
| 5 | [ 0.00004283, 0.00000942] | 9.61e-10 | 4.28e-05 | 7.27e-03 |
| 6 | [ 0.00000031, 0.00000007] | 5.00e-14 | 3.09e-07 | 2.42e-03 |
| 7 | [ 0.00000000, 0.00000000] | 3.05e-19 | 7.46e-10 | |

***Table 3.4.*** *Algorithm 3.15 applied to (3.5).* $\boldsymbol{x}_0^T = [1,\ 2]$, $\mu_0 = 1$.

The solution is found without problems, and the columns with $f$ and $\|\nabla f\|$ show *superlinear convergence*, as defined in (1.8). ∎

**Example 3.5.** We have used Algorithm 3.15 on Rosenbrock's function from Example 2.8. We use the same starting point, $\boldsymbol{x}_0 = \begin{pmatrix} -1.2 & 1 \end{pmatrix}^T$, and with $\tau = 10^{-2}$, $\varepsilon_1 = 10^{-10}$, $\varepsilon_2 = 10^{-12}$ we found the solution after 29 iteration steps. The performance is illustrated below

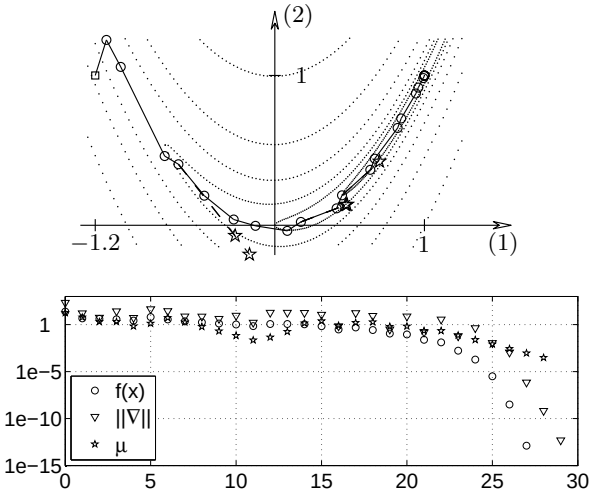



***Figure 3.4.*** *Damped Newton method applied to Rosenbrock's function. Top: iteration path. Bottom: $f(\boldsymbol{x}_k)$, $\|\nabla f(\boldsymbol{x}_k)\|_\infty$ and $\mu$.*

The six stars in the iteration path indicates points, where the attempted step was uphill, ie the current $\boldsymbol{x}$ is not changed, but $\mu$ is updated. After passing the bottom of the parabola, the damping parameter $\mu$ is decreased in most steps. As in the previous example we achieve superlinear final convergence. ∎

## 3.3. Quasi–Newton methods

The modifications discussed in the previous section make it possible to overcome the first three of the main disadvantages of Newton's method shown in Summary 3.6. The damped Newton method is globally convergent, ill-conditioning may be avoided, and minima are rapidly located. However, the fourth disadvantage remains: The user must supply second

derivatives, either by implementing analytically derived expressions or by means of automatic differentiation. For large or complicated problems this may be a costly affair, and it often happens that we only benefit from the quadratic convergence in the last few iterations.

In *quasi–Newton methods* the 2nd derivatives are not required. The Newton equation (3.2),

$$\boldsymbol{\nabla}^2 f(\boldsymbol{x})\, \boldsymbol{h}_{\rm n} \;=\; -\boldsymbol{\nabla} f(\boldsymbol{x}) \;,$$

is replaced by

$$\boldsymbol{B}\, \boldsymbol{h}_{\rm qn} \;=\; -\boldsymbol{\nabla} f(\boldsymbol{x}) \;, \tag{3.16}$$

where the matrix $\boldsymbol{B}$ is an approximation to the Hessian $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$. This matrix is modified during the iteration process, and many different schemes have been proposed.

Possibly the most straight-forward quasi–Newton method is obtained if the elements of the Hessian are approximated by finite differences, see Appendix A.3. This can give a good approximation to the Hessian, but it involves $n$ extra evaluations of the gradient in each iteration. Further, there is no guarantee that $\boldsymbol{B}$ is positive definite, and in order to get a robust algorithm, it might for instance be combined with the damped Newton method outlined in Algorithm 3.8.

The quasi–Newton methods that we shall discuss in the remainder of this chapter avoid these unfruitful extra evaluations of the gradient and the need for damping. First, however, we must mention that instead of generating a sequence of $\boldsymbol{B}$-matrices for use in (3.16) some quasi–Newton methods work with a sequence of $\boldsymbol{D}$-matrices such that the search direction $\boldsymbol{h}_{\rm qn}$ is given by

$$\boldsymbol{h}_{\rm qn} \;=\; -\boldsymbol{D}\, \boldsymbol{\nabla} f(\boldsymbol{x}) \;. \tag{3.17}$$

Thus, the current $\boldsymbol{D}$ is an approximation to the inverse of $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$. We shall discuss these two approaches in parallel. The modifications from the current $\boldsymbol{B}$ (or $\boldsymbol{D}$) are made by relatively simple *updates*, and for each updating formula for $\boldsymbol{B}$ there is a similar updating formula for $\boldsymbol{D}$ with the same computational cost. This means that the difference in cost is that with formulation (3.16) we have to solve a linear system in the $n$ unknown components of $\boldsymbol{h}_{\rm qn}$; this is a $O(n^3)$ process, whereas the multiplication with $\boldsymbol{D}$ in (3.17) is a $O(n^2)$ process, ie it is cheaper for large $n$.

The need for damping is avoided because the sequence of $\boldsymbol{B}$ (or $\boldsymbol{D}$) matrices are generated such that every matrix is positive definite, and

Theorem 2.15 tells us that then every $\boldsymbol{h}_{\text{qn}}$ is a descent direction. In order to find a good step length we can use line search or a trust region approach. We shall only discuss the former, and conclude this section by giving the following framework.

**Framework 3.18. Quasi–Newton method with updating and line search**

**begin**
  $\boldsymbol{x} := \boldsymbol{x}_0$; $k := 0$; *found* $:= \cdots$ $\{1^\circ\}$
  $\boldsymbol{B} := \boldsymbol{B}_0$ (or $\boldsymbol{D} := \boldsymbol{D}_0$ ) $\{2^\circ\}$
  **while** (**not** *found*) **and** $(k < k_{\max})$
    Solve $\boldsymbol{B}\boldsymbol{h}_{\text{qn}} = -\boldsymbol{\nabla} f(\boldsymbol{x})$ (or compute $\boldsymbol{h}_{\text{qn}} := -\boldsymbol{D}\,\boldsymbol{\nabla} f(\boldsymbol{x})$)
    $\boldsymbol{x}_{\text{new}} := \text{line\_search}(\boldsymbol{x}, \boldsymbol{h}_{\text{qn}})$ $\{3^\circ\}$
    Update *found* $\{1^\circ\}$
    Update $\boldsymbol{B}$ to $\boldsymbol{B}_{\text{new}}$ (or $\boldsymbol{D}$ to $\boldsymbol{D}_{\text{new}}$) $\{4^\circ\}$
    $\boldsymbol{x} := \boldsymbol{x}_{\text{new}}$
**end**

We have the following remarks:

1° Initialization. We recommend to use the stopping criteria (3.14).

2° The choice $\boldsymbol{B}_0 = \boldsymbol{I}$ (or $\boldsymbol{D}_0 = \boldsymbol{I}$) will make the iteration process start like the steepest descent method, which has good global convergence properties. With a good updating strategy the matrices will converge to good approximations to $\boldsymbol{\nabla}^2 f(\hat{\boldsymbol{x}})$ (or the inverse of this), and the process ends up somewhat like Newton's method.

3° Some methods demand exact line search, others work fine with very loose line search.

4° In the following we present the requirements to the updating and the techniques needed in order for the sequence of $\boldsymbol{B}$ (or $\boldsymbol{D}$) matrices to converge towards $\boldsymbol{\nabla}^2 f(\hat{\boldsymbol{x}})$ (or $\big(\boldsymbol{\nabla}^2 f(\hat{\boldsymbol{x}})\big)^{-1}$), respectively, $\hat{\boldsymbol{x}}$ being the minimizer.

### 3.3.1. Updating formulas

In this subsection we start by looking at the requirements that should be satisfied by an updating scheme for approximations to the Hessian or its inverse. Later we discuss how to build in other desirable features of the matrix sequence.

Let $\boldsymbol{x}$ and $\boldsymbol{B}$ be the current iterate and approximation to $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$. Given these, steps 1° and 2° of Framework 3.18 can be performed yielding $\boldsymbol{x}_{\text{new}}$. The objective is to calculate $\boldsymbol{B}_{\text{new}}$ by a correction of $\boldsymbol{B}$. The correction must contain some information about the second derivatives. Clearly, this information is only approximate, it is based on the gradients of $f$ at the two points. Now, consider the first order Taylor approximation of the gradient around $\boldsymbol{x}_{\text{new}}$,

$$\boldsymbol{\nabla} f(\boldsymbol{x}) \;\simeq\; \boldsymbol{\nabla} f(\boldsymbol{x}_{\text{new}}) + \boldsymbol{\nabla}^2 f(\boldsymbol{x}_{\text{new}})\,(\boldsymbol{x} - \boldsymbol{x}_{\text{new}}) \;\; .$$

By rearranging we get an expression similar to (2.35),

$$\boldsymbol{\nabla} f(\boldsymbol{x}_{\text{new}}) - \boldsymbol{\nabla} f(\boldsymbol{x}) \;\simeq\; \boldsymbol{\nabla}^2 f(\boldsymbol{x}_{\text{new}})\,(\boldsymbol{x}_{\text{new}} - \boldsymbol{x}) \;\; .$$

We require that the approximation $\boldsymbol{B}_{\text{new}}$ satisfies the corresponding equality

$$\begin{aligned} \boldsymbol{B}_{\text{new}}\boldsymbol{h} \;=\; \boldsymbol{y} \quad \text{with} \quad & \boldsymbol{h} = \boldsymbol{x}_{\text{new}} - \boldsymbol{x} \;, \\ & \boldsymbol{y} = \boldsymbol{\nabla} f(\boldsymbol{x}_{\text{new}}) - \boldsymbol{\nabla} f(\boldsymbol{x}) \;. \end{aligned} \tag{3.19}$$

This is the so-called *quasi–Newton condition*. The same arguments lead to the alternative formulation of the quasi–Newton condition,

$$\boldsymbol{D}_{\text{new}}\,\boldsymbol{y} \;=\; \boldsymbol{h} \;. \tag{3.20}$$

If $n=1$, then we see that $\boldsymbol{B}_{\text{new}}$ is an approximation to $f''$ at a point between the current and the new iterate, and (3.19) tells that this approximation is the slope of the line between the points $(x, f'(x))$ and $(x_{\text{new}}, f'(x_{\text{new}}))$ in the graph of $f'$. Therefore condition (3.19) is sometimes referred to as the *secant condition*.

For $n>1$ the quasi–Newton condition is an underdetermined linear system with $n$ equations and the $n^2$ elements in $\boldsymbol{B}_{\text{new}}$ (or $\boldsymbol{D}_{\text{new}}$) as unknowns. Therefore additional conditions are needed to get a well defined method. Different quasi–Newton methods are distinguished by the choice of these extra conditions. A common feature of the methods that we shall describe is that they have the form

$$\boldsymbol{B}_{\text{new}} \;=\; \boldsymbol{B} + \boldsymbol{W}_B \qquad (\textit{or} \;\; \boldsymbol{D}_{\text{new}} \;=\; \boldsymbol{D} + \boldsymbol{W}_D \;) \;.$$

We say that $\boldsymbol{B}_{\text{new}}$ is obtained by *updating* $\boldsymbol{B}$. The correction matrix $\boldsymbol{W}$ is constructed so that the the quasi-Newton condition is satisfied, and maybe ensures desirable features, such as preservation of symmetry

and positive definiteness. Also, it should be simple to compute the correction, and in most methods used in practice, $\boldsymbol{W}$ is either a *rank-one matrix*,

$$\boldsymbol{B}_{\text{new}} \;=\; \boldsymbol{B} + \boldsymbol{a}\,\boldsymbol{b}^T \;,$$

or a *rank-two matrix*,

$$\boldsymbol{B}_{\text{new}} \;=\; \boldsymbol{B} + \boldsymbol{a}\,\boldsymbol{b}^T + \boldsymbol{u}\,\boldsymbol{v}^T \;,$$

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

**Example 3.6.** One of the first updating formulas to be thoroughly discussed was *Broyden's rank-one formula*, [6],

$$\boldsymbol{B}_{\text{new}} \;=\; \boldsymbol{B} + \boldsymbol{a}\,\boldsymbol{b}^T \;,$$

The vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ are chosen so that they satisfy the quasi–Newton condition (3.19),

$$\left(\boldsymbol{B} + \boldsymbol{a}\boldsymbol{b}^T\right) \boldsymbol{h} \;=\; \boldsymbol{y} \;,$$

supplied with the condition that

$$\left(\boldsymbol{B} + \boldsymbol{a}\boldsymbol{b}^T\right) \boldsymbol{v} \;=\; \boldsymbol{B}\boldsymbol{v} \quad \text{for all} \quad \boldsymbol{v} \mid \boldsymbol{v}^T\boldsymbol{h} = 0 \;.$$

The rationale behind this is that we want to keep information already in $\boldsymbol{B}$, and after all we only have new information about 2nd derivatives in the direction given by $\boldsymbol{h}$. The reader can easily verify that the result is *Broyden's rank-one formula* for updating the approximation to the Hessian:

$$\boldsymbol{B}_{\text{new}} \;=\; \boldsymbol{B} + \frac{1}{\boldsymbol{h}^T\boldsymbol{h}}\big(\boldsymbol{y} - \boldsymbol{B}\,\boldsymbol{h}\big)\boldsymbol{h}^T \quad . \tag{3.21}$$

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

$$\boldsymbol{D}_{\text{new}} = \boldsymbol{D} + \frac{1}{\boldsymbol{y}^T\boldsymbol{y}}\big(\boldsymbol{h} - \boldsymbol{D}\,\boldsymbol{y}\big)\boldsymbol{y}^T \quad . \tag{3.22}$$

The observant reader will notice the symmetry between these two updating formulas. This is further discussed in Section 3.5.

Now, given some initial approximation $\boldsymbol{B}_0$ (or $\boldsymbol{D}_0$) (the choice of which shall be discussed later), we can use (3.21) or (3.22) to generate the sequence needed in the framework 3.18. However, two important features of the Hessian (or its inverse) would then be disregarded: We wish both matrices $\boldsymbol{B}$ and $\boldsymbol{D}$ to be symmetric and positive definite. This is not the case for (3.21) and (3.22), and thus the use of Broyden's formula may lead to steps which are not even downhill, and convergence towards saddle points or maxima will often occur. Therefore, these formulas are never used for unconstrained optimization.

The formulas were developed for solving systems of nonlinear equations, and they have several other applications, for instance in methods for least squares problems and minimax optimization. We return to (3.21) in Section 6.4. ∎

### 3.3.2. Symmetric updating

The Hessian and its inverse are symmetric, and therefore it is natural to require that also $\boldsymbol{B}$ and $\boldsymbol{D}$ be symmetric. We can obtain this for the $\boldsymbol{D}$-sequence if we start with a symmetric $\boldsymbol{D}_0$ and use rank-one updates of the form

$$\boldsymbol{D}_{\text{new}} \;=\; \boldsymbol{D} + \boldsymbol{u}\,\boldsymbol{u}^T \;.$$

The quasi–Newton condition (3.20) determines $\boldsymbol{u}$ uniquely:

$$\boldsymbol{h} \;=\; \boldsymbol{D}\,\boldsymbol{y} + \boldsymbol{u}\,\boldsymbol{u}^T\boldsymbol{y} \quad\Leftrightarrow\quad (\boldsymbol{u}^T\boldsymbol{y})\boldsymbol{u} \;=\; \boldsymbol{h} - \boldsymbol{D}\,\boldsymbol{y} \;,$$

leading to

$$\boldsymbol{D}_{\text{new}} \;=\; \boldsymbol{D} + \frac{1}{\boldsymbol{u}^T\boldsymbol{y}}\,\boldsymbol{u}\,\boldsymbol{u}^T \qquad \text{with} \quad \boldsymbol{u} \;=\; \boldsymbol{h} - \boldsymbol{D}\,\boldsymbol{y} \;.$$

A similar derivation gives the symmetric rank-one updating formula for approximations to $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$,

$$\boldsymbol{B}_{\text{new}} \;=\; \boldsymbol{B} + \frac{1}{\boldsymbol{h}^T\boldsymbol{v}}\,\boldsymbol{v}\,\boldsymbol{v}^T \qquad \text{with} \quad \boldsymbol{v} \;=\; \boldsymbol{y} - \boldsymbol{B}\,\boldsymbol{h} \;.$$

These formulas are known as the *SR1 formulas*. They have been used very little in practice. Notice that the updating breaks down if $\boldsymbol{u}^T\boldsymbol{y} = 0$. This occurs if $\boldsymbol{u} = \boldsymbol{0}$, in which case we just take $\boldsymbol{D}_{\text{new}} = \boldsymbol{D}$. It also occurs if a nonzero $\boldsymbol{u}$ is orthogonal to $\boldsymbol{y}$, and in this case there is no solution. In implementations this is handled by setting $\boldsymbol{D}_{\text{new}} = \boldsymbol{D}$ whenever $|\boldsymbol{u}^T\boldsymbol{y}|$ is too small.

**Example 3.7.** The SR1 formulas have some interesting properties. The most important is that a quasi–Newton method without line search based on SR1 updating will minimize a quadratic with positive definite Hessian in at most $n{+}1$ iterations, provided that the search directions are linearly independent and $\boldsymbol{y}^T\boldsymbol{u}$ remains positive. Further, in this case $\boldsymbol{D}_{\text{new}}$ equals $\boldsymbol{\nabla}^2 f(\boldsymbol{x}^*)^{-1}$ after $n{+}1$ steps. This important property is called *quadratic termination*, cf Section 2.6. ∎

The observant reader will have noticed a disappointing fact: a rank-one updating has not enough freedom to satisfy the quasi-Newton condition, preserve symmetry and the desirable property, that we mentioned

many times: If $\boldsymbol{B}$ (or $\boldsymbol{D}$) is positive definite, then we are sure that it is nonsingular, and that the direction $\boldsymbol{h}_{\text{qn}}$ computed by means of this matrix is sure to be downhill. This property can be achieved by rank-two updates, and and in the next two sections we describe two methods based on that.

## 3.4. The DFP formula

This updating formula was proposed by *Davidon* in 1959 and was later developed by *Fletcher* and *Powell* in 1963. A proper derivation of this formula is very lengthy, so we confine ourselves to the less rigorous presentation given by Fletcher in [19].

A symmetric rank-two formula for updating the inverse Hessian can be written as

$$\boldsymbol{D}_{\text{new}} \;=\; \boldsymbol{D} + \boldsymbol{u}\,\boldsymbol{u}^T + \boldsymbol{v}\,\boldsymbol{v}^T \;.$$

We insert this in the quasi–Newton condition (3.20) and get

$$\boldsymbol{h} \;=\; \boldsymbol{D}\,\boldsymbol{y} + \boldsymbol{u}\,\boldsymbol{u}^T\boldsymbol{y} + \boldsymbol{v}\,\boldsymbol{v}^T\boldsymbol{y} \;.$$

With two updating terms there is no unique determination of $\boldsymbol{u}$ and $\boldsymbol{v}$, but Fletcher points out that an obvious choice is to try

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

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

**Definition 3.23. DFP updating**

$$\boldsymbol{D}_{\text{new}} \;=\; \boldsymbol{D} + \frac{1}{\boldsymbol{h}^T\boldsymbol{y}}\,\boldsymbol{h}\,\boldsymbol{h}^T - \frac{1}{\boldsymbol{y}^T\boldsymbol{v}}\,\boldsymbol{v}\,\boldsymbol{v}^T \;,$$

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

The DFP formula dominated the field of quasi–Newton methods for more than a decade, and it was found to work well. Traditionally it is combined with exact line search, but it may also be used with soft line search as we shall see in a moment. In general it is more efficient than the conjugate gradient method, cf Section 2.7. A method with this updating formula and exact line search has the following important properties:

On quadratic objective functions with positive definite Hessian:

1° it terminates in at most $n$ iterations with $\boldsymbol{D}_{\text{new}} = \left(\boldsymbol{\nabla}^2 f(\hat{\boldsymbol{x}})\right)^{-1}$,

2° it generates conjugate directions,

3° it generates conjugate gradients if $\boldsymbol{D}_0 = \boldsymbol{I}$ .

On general functions:

4° it preserves positive definite $\boldsymbol{D}$-matrices if ${\boldsymbol{h}_{\text{qn}}}^T \boldsymbol{y} > 0$ in all steps,

5° it gives superlinear final convergence,

6° it gives global convergence for strictly convex objective functions.

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

$$\|\hat{\boldsymbol{x}} - \boldsymbol{x}_{\text{new}}\| \ll \|\hat{\boldsymbol{x}} - \boldsymbol{x} \quad \text{for} \quad k \to \infty \ ,$$

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

**Example 3.8.** The proof of property 4° in the above list is instructive, and therefore we give it here:

Assume that $\boldsymbol{D}$ is positive definite. Then its *Cholesky factorization* exists: $\boldsymbol{D} = \boldsymbol{C}^T \boldsymbol{C}$, and for any non-zero $\boldsymbol{z} \in \mathbb{R}^n$ we use Definition 3.23 to find

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

We introduce $\boldsymbol{a} = \boldsymbol{C}\, \boldsymbol{z}$, $\boldsymbol{b} = \boldsymbol{C}\, \boldsymbol{y}$ and $\theta = \angle(\boldsymbol{a}, \boldsymbol{b})$, cf (2.13), and get

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

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

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

$$\boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}_{\text{new}}) \ > \ \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) \ . \tag{3.24}$$

Notice that this condition will be satisfied when $\boldsymbol{x}_{\text{new}}$ is found by a successful line search: According to (2.17) – (2.18) we have

$$\boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}_{\text{new}}) \;=\; \gamma\, \varphi'(\alpha_{\text{s}}) \;>\; \gamma\, \varphi'(0) \;=\; \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) \;.$$

Here $\varphi$ is the line search function from $\boldsymbol{x}$ in direction $\boldsymbol{h}_{\text{cg}}$, cf Section 2.3, and $\gamma = \|\boldsymbol{h}\|/\|\boldsymbol{h}_{\text{cg}}\|$.

The DFP formula with exact line search works well in practice and has been widely used. When the soft line search methods were introduced, however, the DFP formula appeared less favourable because it sometimes fails with a soft line search. In the next section we give another rank-two updating formula which works better, and the DFP formula only has theoretical importance today. The corresponding formula for updating approximations to the Hessian itself is (see for instance [19, Section 3.2])

**Definition 3.25. DFP updating for $\boldsymbol{B}$**

$$\boldsymbol{B}_{\text{new}} \;=\; \boldsymbol{B} + \frac{1}{\boldsymbol{h}^T \boldsymbol{y}} \left( (1 + \frac{\boldsymbol{h}^T \boldsymbol{u}}{\boldsymbol{h}^T \boldsymbol{y}}) \boldsymbol{y}\, \boldsymbol{y}^T - \boldsymbol{y}\, \boldsymbol{u}^T - \boldsymbol{u}\, \boldsymbol{y}^T \right) \;,$$

where $\boldsymbol{h} = \boldsymbol{x}_{\text{new}} - \boldsymbol{x}$ , $\boldsymbol{y} = \boldsymbol{\nabla} f(\boldsymbol{x}_{\text{new}}) - \boldsymbol{\nabla} f(\boldsymbol{x})$ , $\boldsymbol{u} = \boldsymbol{B}\, \boldsymbol{h}$ .

Combining this with Definition 3.23, it is easy to see that if both $\boldsymbol{B}$ and $\boldsymbol{D}$ symmetric, then also $\boldsymbol{B}_{\text{new}}$ and $\boldsymbol{D}_{\text{new}}$ are symmetric. It is not difficult (but a bit tedious) to show that if $\boldsymbol{B}$ and $\boldsymbol{D}$ satisfy $\boldsymbol{B}\, \boldsymbol{D} = \boldsymbol{I}$, then also $\boldsymbol{B}_{\text{new}}$ and $\boldsymbol{D}_{\text{new}}$ are each other's inverse.

## 3.5. The BFGS formulas

The final updating formulas to be discussed in this chapter are known as the *BFGS formulas.* They were discovered independently by Broyden, Fletcher, Goldfarb and Shanno in 1970. These formulas are the most popular of all the updating formulas described in the literature.

As we saw with the DFP formula, the BFGS formulas are difficult to derive directly from the requirements. However, they arrive in a simple way through the concept of *duality*, which will be discussed briefly here. Remember the quasi–Newton conditions (3.19) – (3.20):

$$\boldsymbol{B}_{\text{new}}\, \boldsymbol{h} \;=\; \boldsymbol{y} \qquad \text{and} \qquad \boldsymbol{D}_{\text{new}}\, \boldsymbol{y} \;=\; \boldsymbol{h} \;.$$

These two equations have the same form, except that $\boldsymbol{h}$ and $\boldsymbol{y}$ are inter-

changed and $\boldsymbol{B}_{\text{new}}$ is replaced by $\boldsymbol{D}_{\text{new}}$. This implies that any updating formula for $\boldsymbol{B}$ which satisfies (3.19) can be transformed into an updating formula for $\boldsymbol{D}$. Further, any formula for $\boldsymbol{D}$ has a dual formula for $\boldsymbol{B}$ which is found by the substitution $\boldsymbol{D} \leftrightarrow \boldsymbol{B}$ and $\boldsymbol{h} \leftrightarrow \boldsymbol{y}$ . Performing this operation on the DFP formulas in definitions 3.23 and 3.23 yields the following updating formulas,

**Definition 3.26. BFGS updating for $\boldsymbol{B}$**

$$\boldsymbol{B}_{\text{new}} \;=\; \boldsymbol{B} + \frac{1}{\boldsymbol{h}^T\boldsymbol{y}}\,\boldsymbol{y}\,\boldsymbol{y}^T - \frac{1}{\boldsymbol{h}^T\boldsymbol{u}}\,\boldsymbol{u}\,\boldsymbol{u}^T \;,$$

where $\boldsymbol{h} = \boldsymbol{x}_{\text{new}} - \boldsymbol{x}$ , $\boldsymbol{y} = \boldsymbol{\nabla} f(\boldsymbol{x}_{\text{new}}) - \boldsymbol{\nabla} f(\boldsymbol{x})$ , $\boldsymbol{u} = \boldsymbol{B}\,\boldsymbol{h}$ .

**BFGS updating for $\boldsymbol{D}$**

$$\boldsymbol{D}_{\text{new}} \;=\; \boldsymbol{D} + \frac{1}{\boldsymbol{h}^T\boldsymbol{y}}\left(\big(1 + \frac{\boldsymbol{y}^T\boldsymbol{v}}{\boldsymbol{h}^T\boldsymbol{y}}\big)\boldsymbol{h}\,\boldsymbol{h}^T - \boldsymbol{h}\,\boldsymbol{v}^T - \boldsymbol{v}\,\boldsymbol{h}^T\right) \;,$$

where $\boldsymbol{h}$ and $\boldsymbol{y}$ are defined above and $\boldsymbol{v} = \boldsymbol{D}\,\boldsymbol{y}$ .

These updating formulas have much better performance than the DFP formulas; see [43] for an excellent explanation why this is the case. If we make the dual operation on the BFGS updates we return to the DFP updatings, as expected. The BFGS formula produces a sequence of $\boldsymbol{B}$-matrices which converges to $\boldsymbol{\nabla}^2 f(\hat{\boldsymbol{x}})$ and the first DFP formula produces $\boldsymbol{D}$-matrices which converge to $\big(\boldsymbol{\nabla}^2 f(\hat{\boldsymbol{x}})\big)^{-1}$.

The BFGS formulas are always used together with soft line search and as discussed above the line search should be initiated with the full quasi–Newton step in each iteration step, ie the initial $\alpha$ in Algorithm 2.20 should be one. Experiments show that it should be implemented with a very loose line search; typical values for the parameters in (2.18) are $\beta_1 = 10^{-4}$ and $\beta = 0.9$.

The properties $1^\circ$ – $6^\circ$ of the DFP formula also hold for the BFGS formulas. Moreover, Powell has proved a better convergence result for the latter formulas namely that they will also converge with a soft line search on convex problems. Unfortunately, convergence towards a stationary point has not been proved for neither the DFP nor the BFGS formulas on general nonlinear functions – no matter which type of line search. Still, BFGS with soft line search is known as the method which never fails to come out with a stationary point.

## 3.6. Implementation of a quasi–Newton method

In this section we shall give some details of the implementation of a quasi–Newton algorithm. We start by giving a version which is adequate for student exercises and preliminary research, and finish by discussing some modifications needed for professional codes. Based on the discussion in the previous sections we have chosen a BFGS updating formula for the inverse Hessian.

**Algorithm 3.27. Quasi–Newton method with BFGS updating**

**begin**
  $\boldsymbol{x} := \boldsymbol{x}_0$;  $\boldsymbol{D} := \boldsymbol{D}_0$;  $k := 0$;  $v := 0$  {1°}
  **while** $\|\boldsymbol{\nabla} f(\boldsymbol{x})\| > \varepsilon$ **and** $k < k_{\max}$ **and** $v < v_{\max}$
    $\boldsymbol{h}_{\mathrm{qn}} := \boldsymbol{D}\,(-\boldsymbol{\nabla} f(\boldsymbol{x}))$
    $[\boldsymbol{x}_{\mathrm{new}}, dv] := \mathrm{line\_search}(\boldsymbol{x}, \boldsymbol{h}_{\mathrm{qn}})$  {2°}
    $v := v + dv$;  $k := k{+}1$  {3°}
    $\boldsymbol{h} := \boldsymbol{x}_{\mathrm{new}} - \boldsymbol{x}$;  $\boldsymbol{y} = \boldsymbol{\nabla} f(\boldsymbol{x}_{\mathrm{new}}) - \boldsymbol{\nabla} f(\boldsymbol{x})$
    **if** $\boldsymbol{h}^T\boldsymbol{y} > \varepsilon_{\mathrm{M}}{}^{1/2}\,\|\boldsymbol{h}\|_2\,\|\boldsymbol{y}\|_2$  {4°}
      Use (3.26) to update $\boldsymbol{D}$
    $\boldsymbol{x} := \boldsymbol{x}_{\mathrm{new}}$
**end**

Remarks:

1° Initialize. $k$ and $v$ count iterations and number of function evaluations, respectively, and we are given upper bounds on each of these, $k_{\max}$ and $v_{\max}$.
It is traditionally recommended to use $\boldsymbol{D}_0 = \boldsymbol{I}$, the identity matrix. This matrix is, of course, positive definite and the first step will be in the steepest descent direction.

2° We recommend to use soft line search, for instance an implementation of Algorithm 2.20. It is important to notice that all the function evaluations take place during the line search, and we assume that the new function and gradient values are returned together with $\boldsymbol{x}_{\mathrm{new}}$ and the number $dv$ of function evaluations used during the line search.

3° Update iteration and function evaluation count.

4° $\varepsilon_{\mathsf{M}}$ is the computer accuracy. This sharpening of the curvature condition (3.24) is recommended by [14] as an attempt to keep the $\boldsymbol{D}$-matrices significantly positive definite.

For large values of $n$ it may be prohibitive to store and work with the $n \times n$ matrix $\boldsymbol{D}$. In such cases it is often a good idea to use a *limited memory BFGS* method. In such an algorithm one keeps a few, say $p$, $p \ll n$, of the most recent $\boldsymbol{h}$ and $\boldsymbol{y}$ vectors and use these to compute an approximation to $\boldsymbol{h}_{\text{qn}} = -\boldsymbol{D}\,\boldsymbol{\nabla} f(\boldsymbol{x})$. See [44, Chapter 9] for an excellent presentation of these methods.

**Example 3.9.** We have tried different updating formulas and line search methods to find the minimizer of Rosenbrock's function, cf Examples 2.8 and 3.5. The line search parameters were chosen as in Example 2.8.

With the starting point $\boldsymbol{x}_0 = \begin{pmatrix} -1.2 & 1 \end{pmatrix}^T$, the following numbers of iteration steps and evaluations of $f(\boldsymbol{x})$ and $\boldsymbol{\nabla} f(\boldsymbol{x})$ are needed to satisfy the stopping criterion $\|\boldsymbol{\nabla} f(\boldsymbol{x})\| \leq 10^{-10}$.

| Update by | Line search | # it. steps | # fct. evals |
|---|---|---|---|
| DFP | exact | 23 | 295 |
| DFP | soft | 31 | 93 |
| BFGS | exact | 23 | 276 |
| BFGS | soft | 36 | 40 |

The results are as expected: BFGS combined with soft line search needs the smallest number of function evaluations to find the solution. This choice is illustrated below. As in Figures 2.11 and 3.4 we show the iterates and the values of $f(\boldsymbol{x}_k)$ and $\|\boldsymbol{\nabla} f(\boldsymbol{x}_k)\|_\infty$. As with the Damped Newton method we have superlinear final convergence.

The numbers of iterations and function evaluations are both slightly larger than in Example 3.5. Note, however, that with Algorithm 3.27 each evaluation involves $f(\boldsymbol{x})$ and $\boldsymbol{\nabla} f(\boldsymbol{x})$, while each evaluation in the Damped Newton Method also involves the Hessian $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$. For many problems this is not available. If it is, it may be costly: we need to compute $\frac{1}{2}n(n+1)$ elements in the symmetric matrix $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$, while $\boldsymbol{\nabla} f(\boldsymbol{x})$ has $n$ elements only.

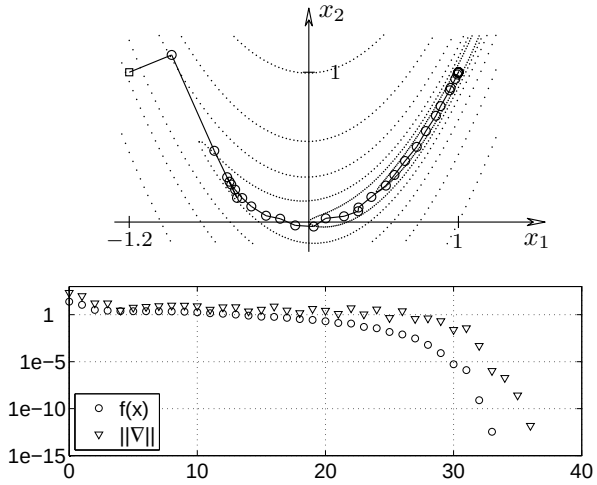



***Figure 3.5.*** *BFGS with soft line search, applied to Rosenbrock's function.*
*Top: iterates* $\boldsymbol{x}_k$. *Bottom:* $f(\boldsymbol{x}_k)$ *and* $\|\boldsymbol{\nabla} f(\boldsymbol{x}_k)\|_\infty$.

■

# Chapter 4

# Direct Search

## 4.1. Introduction

This chapter is a short introduction to a class of methods for unconstrained optimization, which use values of the objective function $f : \mathbb{R}^n \mapsto \mathbb{R}$, but rely on neither explicit nor implicit knowledge of the gradient. These methods are known as *direct search* methods, and special variants have names like pattern search methods, genetic algorithms, etc.

**Example 4.1.** Probably the simplest method is the so-called *coordinate search*. The basic ideas can be introduced by looking at the case $n = 1$. Given a starting point $x$ and a step-length $\Delta$, the basic algorithm is

**if** $f(x+\Delta) \geq f(x)$ **then** $\Delta := -\Delta$
**while** $f(x+\Delta) < f(x)$ **do** $x := x + \Delta$

This is illustrated in Figure 4.1.

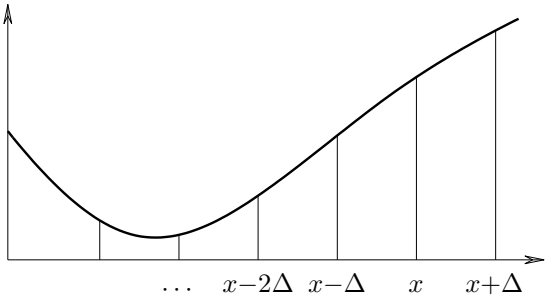



***Figure 4.1.*** *Basic coordinate search.*

It follows that if $|\Delta|$ is small, then then the location of the (local) minimizer may involve many function evaluations. Therefore one should start with a $\Delta$ that reflects the expectation on the distance from the starting guess $x$ to the minimizer. To get better accuracy the algorithm can be restarted from the currently best $x$ with a reduced value for the step-length. This is repeated until $\Delta$ is sufficiently small.

Coordinate search consists in using the above algorithm consecutively in the $n$ coordinate directions, ie when we work in the $j$th direction, we keep the other coordinates fixed. One has to loop several times through the coordinates since a change in one variable generally will change the minimizer with respect to the other variables. Typically, the step-length is kept constant during a loop through the components, and is reduced before the next loop.

The method is simple to implement, but it is not efficient. Also, there is a severe risk that the iterates are caught i *Stiefel's cage*, cf Examples 1.2 and 2.5. There are two reasons why we discuss this method. First, it gives a simple introduction to some of the ideas used in the more advanced methods described in the following sections. Second, we want to warn against using this method except, maybe, to get a crude approximation to the solution. ∎

## 4.2. The Simplex method

This method is also known as the *Nelder–Mead* method, named after the people who first gave a thorough discussion of it, [37]. An $n$-dimensional simplex is formed by a set of $n+1$ points in $\mathbb{R}^n$, the vertices. In a regular simplex all distances between any two vertices is the same. A regular simplex in $\mathbb{R}^2$ is an equilateral triangle.

The basic idea in the simplex method is to take the vertex with largest function value and get a new simplex by reflecting this point in the hyperplane spanned by the other points.

**Example 4.2.** Figure 4.2 illustrates the behaviour of the basic simplex method applied to *Rosenbrock's function* from Example 2.8. We use a regular simplex with side-length 0.25.

The point marked by a star is the first to be reflected in the opposite side, and the reflections are marked by dotted lines. Progress stops because the point marked by a box "wants" to be flipped back to its previous position. The vertex indicated by a circle has the smallest value, and we might proceed searching with a simplex that involves this vertex and has a smaller side.

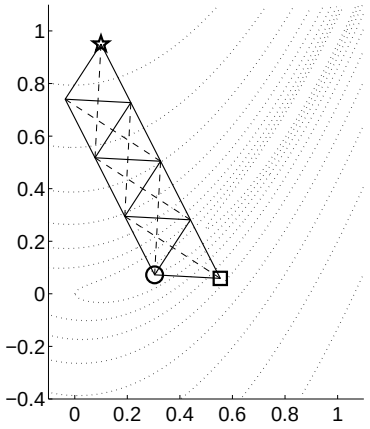



***Figure 4.2.*** *Basic simplex method.*

■

In $n$ dimensions let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n+1}$ denote the vertices of the current simplex and let $\boldsymbol{v}_p$ be the vertex with largest function value. Then

$$\boldsymbol{v}_c \;=\; \frac{1}{n} \sum_{j \neq p} \boldsymbol{v}_j$$

is the centroid of the other vertices and $\boldsymbol{v}_p$ is replaced by

$$\boldsymbol{v}_r \;=\; \boldsymbol{v}_c + \alpha\, \boldsymbol{h} \;, \quad \boldsymbol{h} = \boldsymbol{v}_c - \boldsymbol{v}_p \;, \tag{4.1}$$

where $\alpha = 1$ gives the reflected point. Advanced implementations allow the simplex to change form during the iterations. It may expand in directions along which further improvement is expected. If, for instance , $f(\boldsymbol{v}_r) < \min\{f(\boldsymbol{v}_i)\}$ in the current simplex, we might try a new vertex computed by (4.1) with $\alpha > 1$. Similarly, $0 < \alpha < 1$ leads to a contraction of the simplex.

**Example 4.3.** We consider the same problem and the same initial simplex as in Example 4.2, but now we use the ideas sketched above:

$\boldsymbol{v}_r := \boldsymbol{v}_c + \boldsymbol{h}$
**if** $f(\boldsymbol{v}_r) < \min\{f(\boldsymbol{v}_i)\}$ **then**
  $\boldsymbol{v}_s := \boldsymbol{v}_c + 2\boldsymbol{h}$
  **if** $f(\boldsymbol{v}_s) < f(\boldsymbol{v}_r)$ **then** $\boldsymbol{v}_r := \boldsymbol{v}_s$
**elseif** $f(\boldsymbol{v}_r) > \max_{j \neq p}\{f(\boldsymbol{v}_j)\}$ **then**
  $\boldsymbol{v}_s := \boldsymbol{v}_c + (1/3)\boldsymbol{h}$
  **if** $f(\boldsymbol{v}_s) < f(\boldsymbol{v}_r)$ **then** $\boldsymbol{v}_r := \boldsymbol{v}_s$
**end**

Figure 4.3 shows the results. The trial points are marked by dots. As

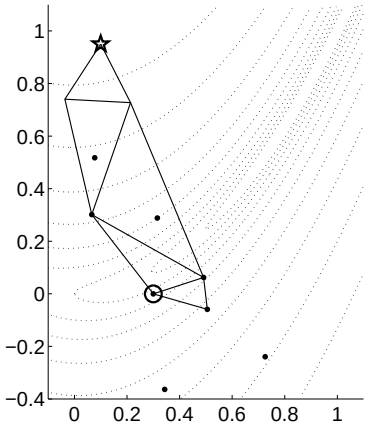



***Figure 4.3.*** *Simplex method with expansions and contractions.*

in Example 4.2 iterations stop when $\boldsymbol{v}_p$ is the new vertex in the new simplex, and we might start a new search with a simplex involving the vertex marked by a circle, and which has reduced side-lengths. We refer to [42, Section 2.6] for more information. ∎

## 4.3. The method of Hooke and Jeeves

This is the classical example of a *pattern search method*: The algorithm attempts to find a pattern in the computed function values and to exploit this in the determination of the minimizer. The method can be explained as follows.

Given a starting point $\boldsymbol{x}$ and a step length $\delta$, the algorithm explores the neighbourhood of $\boldsymbol{x}$ to find a better point $\tilde{\boldsymbol{x}}$. Next explore around the point $\boldsymbol{z} = \tilde{\boldsymbol{x}} + (\tilde{\boldsymbol{x}} - \boldsymbol{x})$, as indicated by the "pattern". The process is repeated as long as we get a better point. Eventually the function value stops reducing, but then a reduction of the step-length may lead to further progress. The combined algorithm may be expressed in terms of an auxiliary function explore and a main procedure move.

We have the following remarks:

1° $\boldsymbol{e}_j$ denotes the $j$th unit vector, equal to the $j$th column in the identity matrix $\boldsymbol{I}$.

2° Pattern move. $\tilde{\boldsymbol{x}}$ is the currently best point, and z is a so-called *base point.*

**Algorithm 4.2. Explore**
Given $\boldsymbol{x}$ and $\Delta$
**begin**
  $\widehat{\boldsymbol{x}} := \boldsymbol{x}$
  **for** j = 1 **to** n **do**
    $\widehat{x}_j := \text{argmin}\{f(\widehat{\boldsymbol{x}}-\Delta\boldsymbol{e}_j),\ f(\widehat{\boldsymbol{x}}),\ f(\widehat{\boldsymbol{x}}+\Delta\boldsymbol{e}_j)\}$ $\{1^\circ\}$
  **end**
**end**

**Algorithm 4.3. Move**
Given $\boldsymbol{x}$ and $\Delta$
**begin**
  $\widehat{\boldsymbol{x}} := \mathsf{explore}(\boldsymbol{x}, \Delta)$
  **repeat**
    **if** $f(\widehat{\boldsymbol{x}}) < f(\boldsymbol{x})$ **then**
      $\boldsymbol{z} := \widehat{\boldsymbol{x}} + (\widehat{\boldsymbol{x}} - \boldsymbol{x})$; $\boldsymbol{x} := \widehat{\boldsymbol{x}}$ $\{2^\circ\}$
    **else**
      $\boldsymbol{z} := \boldsymbol{x}$; $\Delta := \Delta/2$ $\{3^\circ\}$
    **end**
    $\widehat{\boldsymbol{x}} := \mathsf{explore}(\boldsymbol{z}, \Delta)$
  **until** STOP $\{4^\circ\}$
**end**

3° Progress stalled at $\boldsymbol{x}$; try a smaller step-length.

4° Simple stopping criteria are: stop when $\Delta$ is sufficiently small or when the number of function evaluations has exceeded a given limit.

**Example 4.4.** The method is illustrated in Figure 4.4 for a hypothetical function $f : \mathbb{R}^2 \mapsto \mathbb{R}$.

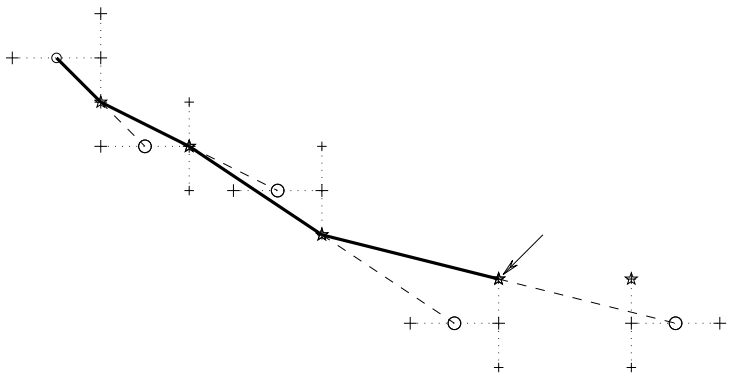


***Figure 4.4.*** *Hooke and Jeeves method.*

Base points and the $\widehat{\boldsymbol{x}}$-points are marked by $\circ$ and $\star$, respectively. Iteration starts at the upper left base point and it is seen that the steps from one $\boldsymbol{x}$ to the next may increase during the process. The last $\widehat{\boldsymbol{x}}$ has a larger function value than the currently best point, marked by an arrow, and a new search with reduced step-length can be started from that point. ■

When implementing the method care should be taken to avoid re-evaluation of $f$ for the same argument. At $1^\circ$ for $j > 1$ we already know $f(\widehat{\boldsymbol{x}})$ for the current $\widehat{\boldsymbol{x}}$, and explore should return not only the final $\widehat{\boldsymbol{x}}$ but also the corresponding function value. A good implementation should also allow the use of different step-lengths in different directions.

## 4.4. Final Remarks

There is an abundance of direct methods. Many of them use the basic ideas outlined in the previous sections and deal with better choices of the "*generating set*", for instance Rosenbrock's method, where the simple coordinate directions of Hooke and Jeeves are replaced by a set of orthogonal directions in $\mathbb{R}^n$, successively updated during the iterations; see [42, Section 2.5]. Other methods involve considerations of the computer architecture, for instance the parallel version of *multi-directional search* described in [52]. A very readable discussion of convergence properties of this type of methods is given in [53].

*Genetic algorithms* and *random search* methods have proven successful in some applications, see for instance [28] and [5]. These methods do not rely on a systematic generating set.

This is also the case for methods that can be listed under the heading *surrogate modelling*. Here the points $\boldsymbol{x}_k$ and function values $f(\boldsymbol{x}_k)$ computed during the search are used to build and successively refine a model of (an approximation to) the behaviour of the objective function; see for instance [4], [50] and [32].

# Chapter 5

# Linear Data Fitting

Given *data points* $(t_1, y_1), \ldots, (t_m, y_m)$, which are assumed to satisfy

$$y_i \;=\; \Upsilon(t_i) + e_i \;. \tag{5.1}$$

$\Upsilon$ is the so-called *background function* and the $\{e_i\}$ are (measurement) errors, often referred to as *"noise"*. We wish to find an approximation to $\Upsilon(t)$ in the domain $[a, b]$ spanned by the data abscissas. To do that we are given (or we choose) a *fitting model* $M(\boldsymbol{x}, t)$ with arguments $t$ and the *parameters* $\boldsymbol{x} = (x_1, \ldots, x_n)^T$. We seek $\hat{\boldsymbol{x}}$ such that $M(\hat{\boldsymbol{x}}, t)$ is the "best possible" approximation to $\Upsilon(t)$ for $t \in [a, b]$. In all data fitting problems the number of parameters is smaller than the number of data points, $n < m$.

**Example 5.1.** As a simple example consider the data

| $t_i$ | $y_i$ |
|---|---|
| −1.5 | 0.80 |
| −0.5 | 1.23 |
| 0.5 | 1.15 |
| 1.5 | 1.48 |
| 2.5 | 2.17 |

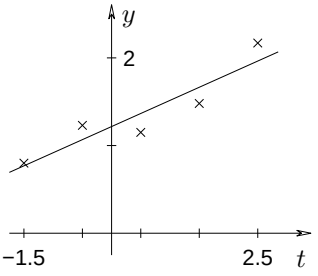



***Figure 5.1.***

Figure 5.1 shows the data points and the model $M(\boldsymbol{x}, t) = x_1 t + x_2$ with the parameter values $x_1 = 0.2990,\ x_2 = 1.2165$.

This problem is continued in Example 5.5. ∎

**Example 5.2.** The 45 data points shown by dots in Figure 5.2 can be fitted with the model

$$M(\boldsymbol{x}, t) \;=\; x_1 e^{-x_3 t} + x_2 e^{-x_4 t} \;. \tag{5.2}$$

In the figure we give the fit for two different choices of the parameters,

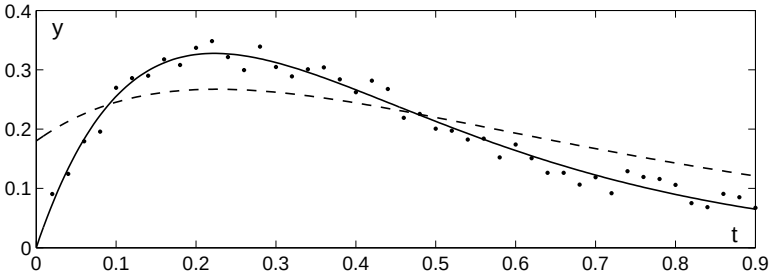



***Figure 5.2.*** *Data points and fitting model (5.2)* ,
*full line:* $\tilde{\boldsymbol{x}} = \begin{pmatrix} 4.00 & -4.00 & 4 & 5 \end{pmatrix}^T$ ,
*dashed line:* $\breve{\boldsymbol{x}} = \begin{pmatrix} 0.84, & -0.66, & 2, & 4 \end{pmatrix}^T$ .

The first choice of parameters seems to be better than the other. Actually, the data points were computed by adding noise to the values given by $\Upsilon(t_i) = M(\tilde{\boldsymbol{x}}, t_i)$, so in this case $\Upsilon(t) = M(\hat{\boldsymbol{x}}, t)$ with $\hat{\boldsymbol{x}} = \tilde{\boldsymbol{x}}$. ∎

**Example 5.3.** Now we consider the first 20 of the data points in Example 5.2, and try to fit them with a polynomial of degree $d$,

$$M(\boldsymbol{x}, t) \;=\; x_1 t^d + x_2 t^{d-1} + \cdots + x_d t + x_{d+1} \;. \tag{5.3}$$

(Note that the parameter vector has $n = d{+}1$ elements).

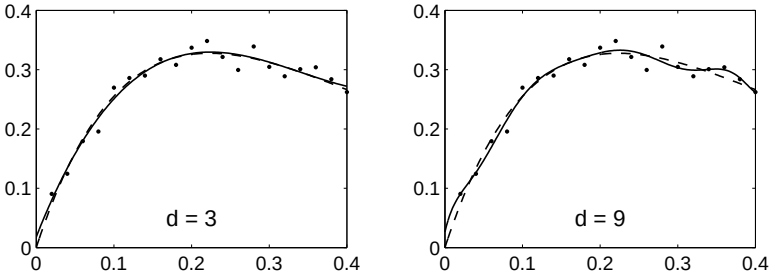



***Figure 5.3.*** *Data points and fit with polynomials.*

The figure shows the *"least squares fit"* (which is defined in Section 5.1) for two different choices of $d$. To the left we give the result for $d = 3$ (full line) and also show $\Upsilon(t)$ (dashed line). It is seen that in this domain a degree 3 polynomial is a very good approximation to $\Upsilon$. The other figure illustrates that if we take $d$ large, then we do not get the desired smoothing. The polynomial is flexible enough to follow not only the background function but also the noise. ∎

This chapter is focussed on *linear data fitting*, ie on how to compute a good estimate for the parameters in the model, when it depends linearly

on the parameters. This means that the fitting model has the form

$$M(\boldsymbol{x}, t) \;=\; x_1\phi_1(t) + \cdots + x_n\phi_n(t) \;, \tag{5.4}$$

where the $\{\phi_j\}$ are given, so-called *basis functions*. The model (5.2) is not linear since it depends nonlinearly on $x_3$ and $x_4$. We refer to Chapter 6 about estimation of the parameters in nonlinear models.

Before specifying what is meant by "a good estimate" $x^*$ for the parameters, we want to mention that data fitting has two slightly different applications,

**Parameter estimation.** Together with measurements of a physical phenomenon we are given the model $M(\boldsymbol{x}, t)$. Often the parameters have physical significance.

**Data representation.** We wish to approximate some data, which may come from experiments. In this case we are free to choose the model for approximating the background function, and should take the following points into consideration in the choice of $M$,

1° If $\boldsymbol{x}$ is close to $\hat{\boldsymbol{x}}$, then $M(\boldsymbol{x}, t)$ should be close to $\Upsilon(t)$.

2° The determination of $\hat{\boldsymbol{x}}$ should be robust against a few *wild points*, ie points with exceptionally large errors.

3° The determination of $\hat{\boldsymbol{x}}$ should simple.

4° The evaluation of $M(\boldsymbol{x}, t)$ for given $t$ should be simple.

5° Available software.

The two cases are closely related, especially as regards points 2°, 3° and 5°.

## 5.1. "Best" fit

A simple measure of the quality of the fit is provided by the vertical distances between the data points and the corresponding values of the model. These are the absolute values of the *residuals*, which depend on the parameters in $\boldsymbol{x}$,

$$r_i \;=\; r_i(\boldsymbol{x}) \;\equiv\; y_i - M(\boldsymbol{x}, t_i) \;. \tag{5.5}$$

By combining this with (5.1) we see that

$$r_i(\boldsymbol{x}) \;=\; \big(y_i - \Upsilon(t_i)\big) + \big(\Upsilon(t_i) - M(\boldsymbol{x}, t_i)\big) \;\equiv\; e_i + \alpha_i \;. \tag{5.6}$$

This shows that each residual is the sum of noise $e_i$ and *approximation error* $\alpha_i$. By choosing $\boldsymbol{x}$ so that some measure of the $\{r_i(\boldsymbol{x})\}$ is minimized, it is reasonable to expect that also the approximation errors are small.

In this section we use a *weighted p–norm* to measure the $\{r_i(\boldsymbol{x})\}$. The commonly used norms are

$$\begin{aligned} \|\boldsymbol{W}\,\boldsymbol{r}\|_1 &= |w_1 r_1| + \cdots |w_m r_m| \quad , \\ \|\boldsymbol{W}\,\boldsymbol{r}\|_2 &= \big(|w_1 r_1|^2 + \cdots |w_m r_m|^2\big)^{1/2} \quad , \\ \|\boldsymbol{W}\,\boldsymbol{r}\|_\infty &= \max\{|w_1 r_1|, \ldots |w_m r_m|\} \quad , \end{aligned}$$

and the "best" set of parameters is defined as

$$\boldsymbol{x}_{(\mathbf{w},p)} \;=\; \underset{\boldsymbol{x}\,\in\,\mathbb{R}^n}{\operatorname{argmin}} \{\; \Omega_p(\boldsymbol{W}, \boldsymbol{x}) \;\equiv\; \|\boldsymbol{W}\,\boldsymbol{r}\|_p \;\} \;. \tag{5.7}$$

Here, $\boldsymbol{W} = \operatorname{diag}(w_1, \ldots, w_m)$, where the weight $w_i \geq 0$ is small/large if $|e_i|$ is expected to be large/small. We return to this aspect in Section 5.4.

In the remainder of this section we shall assume that all weights are equal. It can be seen that $\boldsymbol{x}_{(\beta\mathbf{w},p)} = \boldsymbol{x}_{(\mathbf{w},p)}$ for any scalar $\beta$, so that we can use $\boldsymbol{W} = \boldsymbol{I}$ in all cases with equal weights. Therefore we shall omit $\boldsymbol{W}$ from the notation ( $\Omega_p(\boldsymbol{x}) = \Omega_p(\boldsymbol{I}, \boldsymbol{x})$ and $\boldsymbol{x}_{(p)} = \boldsymbol{x}_{(\mathbf{I},p)}$) and only need to discuss $p$. We say that $\boldsymbol{x}_{(p)}$ is the $L_p$ *estimator*.

The choice $p = 2$ is most widely used. This leads to the so-called *"least squares fit"*, since it corresponds to minimizing the sum of the squares of the residuals. In Example 5.5 and Section 5.2.1 we shall show why it is so popular. This choice is sometimes called the $L_2$-fit. In some applications the $L_1$*-fit* (*"least absolute deviation"*) or the $L_\infty$*-fit* (*"minimax-fit"*) is preferred, see Chapter 7. In many cases $\boldsymbol{x}_{(p)}$ is almost independent of $p$. In the remainder of this book we normally use the shorter notation $\hat{\boldsymbol{x}}$ for the least squares solution $\boldsymbol{x}_{(2)}$.

**Example 5.4.** Consider the simple case, where $n = 1$ and $M(\boldsymbol{x}, t) = x$, ie, $r_i = y_i - x$. Then

$$\begin{aligned} \big(\Omega_2(\boldsymbol{x})\big)^2 &= (y_1 - x)^2 + \cdots + (y_m - x)^2 \\ &= m\,x^2 - 2(\textstyle\sum y_i)x + \sum y_i^2 \;. \end{aligned}$$

This function (and therefore $\Omega_2(\boldsymbol{x})$) is clearly minimized by taking $x = x_{(2)}$ defined by

$$x_{(2)} \;=\; \tfrac{1}{m}\sum_{i=1}^{m} y_i \ .$$

This is recognized as the *mean* (or *average*) of the $\{y_i\}$. It is also easy to see that

$$x_{(\infty)} \;=\; \tfrac{1}{2}\left(\min\{y_i\} + \max\{y_i\}\right) \ ,$$

while it takes a bit more effort to see that

$$x_{(1)} \;=\; \text{median}(y_1, \ldots, y_m) \ ,$$

where the *median* is the middle value; for instance $\text{median}(5,1,2) = 2$ and $\text{median}(3,5,1,2) = 2.5$ .

These three estimates have different response to *wild points*. To see this, let $y_K = \max\{y_i\}$ and assume that this changes to $y_K + \Delta$, where $\Delta > 0$. Then the three minimizers change to $x_{(p)} + \delta_{(p)}$, and it follows from the above that

$$\delta_{(2)} \;=\; \frac{\Delta}{m} \ , \qquad \delta_{(\infty)} \;=\; \frac{\Delta}{2} \ , \qquad \delta_{(1)} \;=\; 0 \ .$$

This indicates why the $L_1$-estimator is said to be *robust*. ∎

**Example 5.5.** The straight line in Figure 5.1 is the least squares fit with the model $M(\boldsymbol{x}, t) = x_1 t + x_2$. We find

$$\boldsymbol{x}_{(2)} \;=\; \begin{pmatrix} 0.299 \\ 1.216 \end{pmatrix} \qquad \text{with} \quad \|r(\boldsymbol{x}_{(2)})\|_2 \;=\; 0.388 \ .$$

For the same data we find

$$\boldsymbol{x}_{(\infty)} \;=\; \begin{pmatrix} 0.313 \\ 1.190 \end{pmatrix} \qquad \text{with} \quad \|r(x_{(\infty)})\|_\infty \;=\; 0.197 \ ,$$

while the $L_1$-fit is not unique: All

$$\boldsymbol{x}_{(1)} \;=\; \begin{pmatrix} 0.227 \\ 1.140 \end{pmatrix} + \alpha \begin{pmatrix} 2 \\ 3 \end{pmatrix} \qquad \text{for} \;\; 0 \le \alpha \le 0.0579$$

give the same minimal value $\|r(\boldsymbol{x}_{(1)})\|_1 = 0.770$. This is illustrated in Figure 5.4. To the left we show the minimax fit; note that $r_2 = -r_3 = r_5 = \|\boldsymbol{r}(\boldsymbol{x}_{(\infty)})\|_\infty$. To the right the dashed lines correspond to the extreme values of the $L_1$-estimators. The solutions correspond to a line rotating around the first data point, and for all lines in that angle it is realized that $r_1 = 0$ and the sum of the absolute values of the other residuals is constant. The two extreme cases correspond to $r_4 = 0$ and $r_5 = 0$, respectively.

Figure 5.5 shows *level curves* of the functions $\Omega_p(\boldsymbol{x})$, $p$=1, 2, $\infty$, cf page 3. We give the curves corresponding to $\Omega_1 = 0.78 + .15k$, $\Omega_2 = 0.4 + 0.1k$ and $\Omega_\infty = 0.22 + 0.1k$ for $k = 0, 1, \ldots, 9$.

The figure shows that the least squares and the minimax problems have well defined minima, while the long "valley" for $p = 1$ reflects that there is not a unique $L_1$ minimizer. The figure also shows that $\Omega_2(\boldsymbol{x})$ is smooth, while the level curves for the other two cases have kinks. This makes it easier to compute $\boldsymbol{x}_{(2)}$ than the other two estimators.

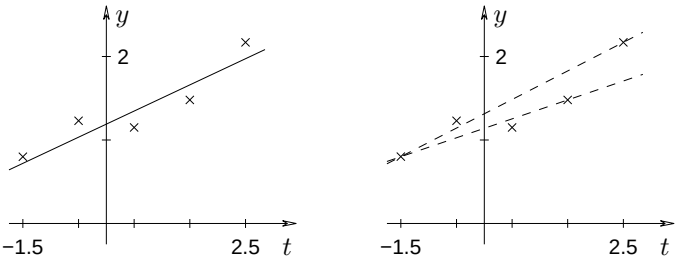



***Figure 5.4.*** *$L_\infty$ and $L_1$ fitting with a straight line.*

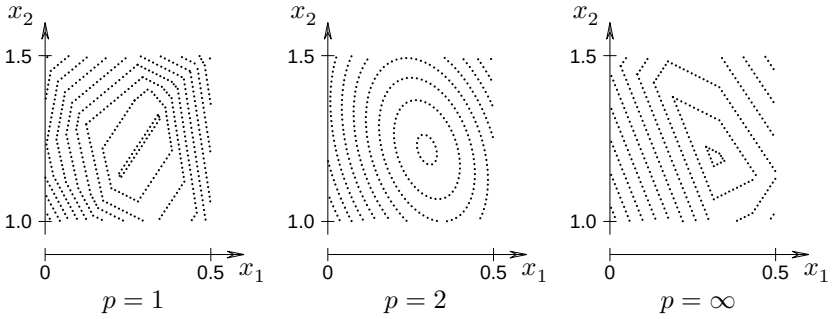



***Figure 5.5.*** *Contours for $\Omega_p(\boldsymbol{x}) = \|\boldsymbol{r}(\boldsymbol{x})\|_p$.*

Finally, Figure 5.6 shows the fits when the data point $(t_5, y_5)$ is changed from $(2.5, 2.17)$ to $(2.5, 4)$. The full and the dash-dotted lines are the least squares and minimax fits, respectively, and as in Figure 5.4 the dashed lines border the range of $L_1$-fits. The parameter values are

$$\boldsymbol{x}_{(2)} = \begin{pmatrix} 0.665 \\ 1.400 \end{pmatrix}, \quad \boldsymbol{x}_{(\infty)} = \begin{pmatrix} 0.800 \\ 1.140 \end{pmatrix},$$

and $x_{(1)}$ can be expressed as above, except that the $\alpha$-range is increased to $0 \le \alpha \le 0.102$. As in Example 5.4 we see that the $L_1$-fit is more robust than the other two.

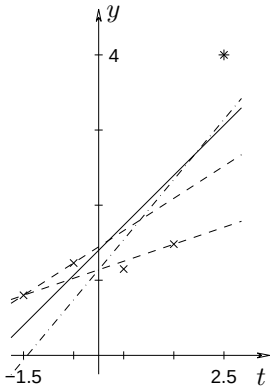



***Figure 5.6.*** *Wild point.*

■

## 5.2. Linear least squares fit

For a linear model (5.4),

$$M(\boldsymbol{x},t) \;=\; x_1\phi_1(t) + \cdots + x_n\phi_n(t)\ ,$$

the residuals (5.5) have the form

$$r_i(\boldsymbol{x}) \;=\; y_i - \big(x_1\phi_1(t_i) + \cdots + x_n\phi_n(t_i)\big)\ .$$

This implies that

$$\begin{pmatrix} r_1(x) \\ \vdots \\ r_i(x) \\ \vdots \\ r_m(x) \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_m \end{pmatrix} - \begin{pmatrix} \phi_1(t_1) & \dots & \phi_n(t_1) \\ \vdots & & \vdots \\ \phi_1(t_i) & \dots & \phi_n(t_i) \\ \vdots & & \vdots \\ \phi_1(t_m) & \dots & \phi_n(t_m) \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}\ ,$$

or

$$\boldsymbol{r}(\boldsymbol{x}) \;=\; \boldsymbol{y} - \boldsymbol{F}\,\boldsymbol{x}\ , \tag{5.8}$$

where $\boldsymbol{F}$ is the $m\times n$ matrix defined by $(\boldsymbol{F})_{ij} = \phi_j(t_i)$.

**Example 5.6.** The model (5.2) is linear if $x_3$ and $x_4$ are given. In that case we can use (5.8) with $n = 2$ and the $i$th row in $\boldsymbol{F}$ is $\boldsymbol{F}_{i,:} = (e^{-x_3t_i} \;\; e^{-x_4t_i})$. If the $\{t_i\}$ are equidistant with $t_{i+1} = t_i + h$, then $\boldsymbol{F}_{i+1,:} = \boldsymbol{g}^T \;.\!*\; \boldsymbol{F}_{i,:}$, where $\boldsymbol{g}^T = \big(e^{-x_3h} \;\; e^{-x_4h}\big)$ and `.*` denotes element-wise (or Hadamard) product.

The polynomial model (5.3) is linear; $n = d{+}1$, all $(\boldsymbol{F})_{i,n} = 1$, and

$$\boldsymbol{F}_{:,j} \;=\; \boldsymbol{t} \;.\!*\; \boldsymbol{F}_{:,j+1}\ , \quad j = d, d{-}1, \dots, 1\ ,$$

where $\boldsymbol{t}$ is the vector $\boldsymbol{t} = (t_1, \dots, t_m)^T$ and $\boldsymbol{F}_{:,j}$ is the $j$th column in $\boldsymbol{F}$. ∎

### 5.2.1. Normal equations

The linear data fitting problem can be reformulated as follows: Given the *overdetermined linear system*

$$\boldsymbol{F}\,\boldsymbol{x} \;\simeq\; \boldsymbol{y}\ , \qquad \boldsymbol{F}\in\mathbb{R}^{m\times n}, \quad \boldsymbol{y}\in\mathbb{R}^m, \quad m > n\ . \tag{5.9}$$

We seek a vector $\boldsymbol{x}\in\mathbb{R}^n$ such that $\boldsymbol{F}\boldsymbol{x}$ is as close as possible to $\boldsymbol{y}$ in the least squares sense. This is obtained when $\boldsymbol{x} = \hat{\boldsymbol{x}}$, a minimizer of the objective function

$$\begin{aligned} f(\boldsymbol{x}) \;&=\; \tfrac{1}{2}\,\|\boldsymbol{r}(\boldsymbol{x})\|_2^2 \;=\; \tfrac{1}{2}\,\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}) \\ &=\; \tfrac{1}{2}\,\boldsymbol{x}^T\,\boldsymbol{F}^T\boldsymbol{F}\,\boldsymbol{x} - \boldsymbol{x}^T\,\boldsymbol{F}^T\boldsymbol{y} + \tfrac{1}{2}\,\boldsymbol{y}^T\boldsymbol{y}\ . \end{aligned} \tag{5.10}$$

This is a function of the form discussed in Theorem 1.7. The matrix $\boldsymbol{A} = \boldsymbol{F}^T\boldsymbol{F}$ has the elements

$$(\boldsymbol{A})_{jk} \;=\; \boldsymbol{F}_{:,j}^T\boldsymbol{F}_{:,k} \;=\; \sum_{i=1}^{m} \phi_j(t_i)\phi_k(t_i) \;=\; (\boldsymbol{A})_{kj} \;. \tag{5.11}$$

This shows that $\boldsymbol{A}$ is *symmetric*, $\boldsymbol{A}^T = \boldsymbol{A}$, so $\boldsymbol{\nabla}^2 f(\boldsymbol{x}) = \boldsymbol{F}^T\boldsymbol{F}$ for all $\boldsymbol{x}$, and the requirement that $\hat{\boldsymbol{x}}$ be a stationary point takes the form

$$\boldsymbol{\nabla} f(\hat{\boldsymbol{x}}) \;=\; \boldsymbol{F}^T\boldsymbol{F}\,\hat{\boldsymbol{x}} - \boldsymbol{F}^T\boldsymbol{y} \;=\; \boldsymbol{0} \;. \tag{5.12}$$

Now, for any $\boldsymbol{u} \in \mathbb{R}^n$

$$\boldsymbol{u}^T\boldsymbol{A}\,\boldsymbol{u} \;=\; \boldsymbol{u}^T\boldsymbol{F}^T\boldsymbol{F}\,\boldsymbol{u} \;=\; \boldsymbol{v}^T\boldsymbol{v} \;\geq\; 0 \;. \tag{5.13}$$

This shows that $\boldsymbol{A}$ is *positive semidefinite*, cf Appendix A.2. Further, if $\boldsymbol{F} \in \mathbb{R}^{m\times n}$ (with $n < m$) has linearly independent columns ($\mathrm{rank}(\boldsymbol{F}) = n$), then $\boldsymbol{u} \neq \boldsymbol{0} \;\Rightarrow\; \boldsymbol{v} = \boldsymbol{F}\boldsymbol{u} \neq \boldsymbol{0}$, and (5.13) shows that $\boldsymbol{u}^T\boldsymbol{A}\,\boldsymbol{u} > 0$, ie, $\boldsymbol{A}$ is *positive definite* in this case, implying a unique solution to (5.12). Thus we have proved the following theorem.

**Theorem 5.14.** A least squares solution to the system (5.9) can be found as a solution $\hat{\boldsymbol{x}}$ to the so-called *normal equations,*

$$\boldsymbol{A}\,\boldsymbol{x} \;=\; \boldsymbol{b} \quad \text{with} \quad \boldsymbol{A} \;=\; \boldsymbol{F}^T\boldsymbol{F} \;,\quad \boldsymbol{b} \;=\; \boldsymbol{F}^T\boldsymbol{y} \;. \tag{5.15}$$

If $\boldsymbol{F}$ has linearly independent columns, then $\hat{\boldsymbol{x}}$ is unique.

Thus, when the model is linear the parameters of the least squares fit are determined by the linear, quadratic system of equations (5.15). The determination of $\boldsymbol{x}_{(p)}$ for any other choice of $p$ is a nonlinear problem and must be solved by iteration. This is part of the reason that least squares fitting is so popular, and in the remainder of this chapter we shall stick to that, but refer to Chapter 7 about other choices of norm. Another reason for the popularity is that the statistical aspects are much better understood in this case, see Section 5.3.

**Example 5.7.** With the data from Example 5.1 and the model $M(\boldsymbol{x}, t) = x_1 t + x_2$ we get

$$\boldsymbol{F} \;=\; \begin{pmatrix} -1.5 & 1 \\ -0.5 & 1 \\ 0.5 & 1 \\ 1.5 & 1 \\ 2.5 & 1 \end{pmatrix} \;,\quad \boldsymbol{y} \;=\; \begin{pmatrix} 0.80 \\ 1.23 \\ 1.15 \\ 1.48 \\ 2.17 \end{pmatrix} \;,$$

$$\boldsymbol{A} = \begin{pmatrix} 11.25 & 2.5 \\ 2.5 & 5 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 6.405 \\ 6.830 \end{pmatrix}.$$

As stated in Example 5.5, the solution to $\boldsymbol{Ax} = \boldsymbol{b}$ is $\hat{\boldsymbol{x}} = \begin{pmatrix} 0.2990 & 1.2165 \end{pmatrix}^T$. ∎

**Example 5.8.** The $j$th column of $\boldsymbol{F}$ contains the values $\phi_j(t_i),\ i = 1, \ldots, m$. Thus, a necessary condition for $\text{rank}(\boldsymbol{A}) = n$ is that the functions $\{\phi_j\}$ are linearly independent. This is not sufficient, however. As a counterexample let $\phi_j(t) = \sin j\pi t$, which are linearly independent, but if $t_i = i,\ i = 1, \ldots, m$, then $\boldsymbol{F} = \boldsymbol{0}$ with rank zero.

At the other extreme suppose that the basis functions are *orthogonal* over the data abscissas, ie,

$$\sum_{i=1}^m \phi_j(t_i)\phi_k(t_i) = 0 \quad \text{for } j \neq k .$$

According to (5.11) the matrix $\boldsymbol{A}$ is diagonal in this case, and the least squares fit is given by

$$x_j = b_j/a_{jj} = \left(\sum_{i=1}^m \phi_j(t_i)y_i\right) / \sum_{i=1}^m (\phi_j(t_i))^2 , \quad j = 1, \ldots, n .$$

Note that each component of $\boldsymbol{x} = \hat{\boldsymbol{x}}$ is independent of the other components. This is not true in the general case.

Provided that $\boldsymbol{A}$ is positive definite, the normal equations can be solved efficiently and accurately via a *Cholesky* or an $LDL^T$ factorization, cf Appendix A.2. ∎

It is instructive to give an alternative derivation of the normal equations: The vector $\boldsymbol{z} = \boldsymbol{Fx}$ lies in $\mathcal{R}(\boldsymbol{F})$, the *range* of $\boldsymbol{F}$. This is the subspace of $\mathbb{R}^m$ which is spanned by the columns of $\boldsymbol{F}$. The situation is illustrated below in the case $m = 3,\ n = 2$.

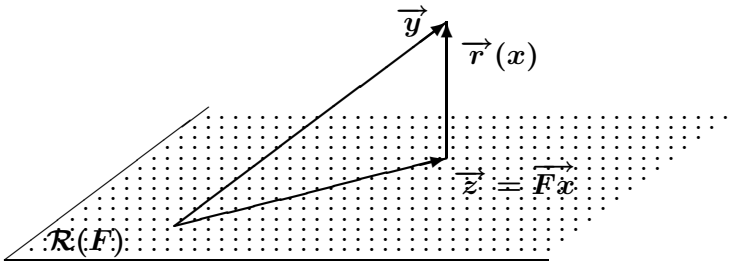



***Figure 5.7.*** *Range and residual vector.*

We wish to minimize the distance between the vectors $\boldsymbol{Fx}$ and $\boldsymbol{y}$ as measured by $\|\boldsymbol{r}\|_2 = \|\boldsymbol{y} - \boldsymbol{Fx}\|_2$. However, $\|\boldsymbol{u}\|_2$ is equal to the Euclidean length of the geometric vector $\vec{\boldsymbol{u}}$ whose coordinates are the elements of

the vector $\boldsymbol{u}$. The minimum length of $\overrightarrow{\boldsymbol{r}}$ is obtained if this vector is orthogonal to $\mathcal{R}(\boldsymbol{F})$, ie $\boldsymbol{z}^T\boldsymbol{r} = 0$ for every $\boldsymbol{z} \in \mathcal{R}(\boldsymbol{F})$. This is equivalent to the condition

$$\boldsymbol{F}^T\boldsymbol{r}(\boldsymbol{x}_{(2)}) \;=\; \boldsymbol{0} \;,$$

and by inserting (5.8) we recognize this condition as the normal equations.

### 5.2.2. Sensitivity

Suppose that (5.9) is changed from $\boldsymbol{F}\boldsymbol{x} \simeq \boldsymbol{y}$ to

$$(\boldsymbol{F} + \boldsymbol{\Delta})\boldsymbol{x} \;\simeq\; \boldsymbol{y} + \boldsymbol{\delta} \;, \qquad \boldsymbol{F}, \boldsymbol{\Delta} \in \mathbb{R}^{m\times n}, \quad \boldsymbol{y}, \boldsymbol{\delta} \in \mathbb{R}^m,$$

with least squares solution $\hat{\boldsymbol{x}}+\boldsymbol{\gamma}$. What can we say about $\boldsymbol{\gamma}$ if $\boldsymbol{\delta}$ and $\boldsymbol{\Delta}$ are not known explicitly, but we are given bounds on the sizes of their elements?

To answer that question we first note that from the above it follows that the least squares solution to an overdetermined system of equations is a linear function of the right-hand side. We shall express this as

$$\hat{\boldsymbol{x}} = \boldsymbol{F}^\dagger\boldsymbol{y} \;, \tag{5.16}$$

where $\boldsymbol{F}^\dagger \in \mathbb{R}^{n\times m}$ is the so-called *pseudo-inverse* of $\boldsymbol{F}$.

Next, for a general matrix $\boldsymbol{B} \in \mathbb{R}^{m\times n}$ we introduce the *condition number*

$$\kappa(\boldsymbol{B}) = \|\boldsymbol{B}\|_2\|\boldsymbol{B}^\dagger\|_2 \;, \tag{5.17}$$

where $\|\boldsymbol{B}\|_2$ is the *matrix norm*

$$\|\boldsymbol{B}\|_2 \;=\; \max_{\boldsymbol{x}\neq\boldsymbol{0}}\{\|\boldsymbol{B}\,\boldsymbol{x}\|_2/\|\boldsymbol{x}\|_2\} \;. \tag{5.18}$$

It can be shown that

$$\|\boldsymbol{B}^\dagger\|_2 \;=\; \max_{\boldsymbol{x}\neq\boldsymbol{0}}\{\|\boldsymbol{x}\|_2/\|\boldsymbol{B}\,\boldsymbol{x}\|_2\} \;.$$

The condition number is a measure of the linear independence of the columns of $\boldsymbol{B}$. If they are (almost) linearly dependent, then $\|\boldsymbol{B}\,\boldsymbol{x}\|_2$ can be much smaller than $\|\boldsymbol{x}\|_2$, so $\|\boldsymbol{B}^\dagger\|_2$ and therefore $\kappa(\boldsymbol{B})$ is (almost) $\infty$.

Now we are ready to answer the question at the start of this subsection:

**Theorem 5.19.** Let $\boldsymbol{F}\boldsymbol{x} \simeq \boldsymbol{y}$ and $(\boldsymbol{F}+\boldsymbol{\Delta})\boldsymbol{x} \simeq \boldsymbol{y}+\boldsymbol{\delta}$ have the least squares solutions $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{x}}+\boldsymbol{\gamma}$, respectively, and let

$$\eta \;\equiv\; \|\boldsymbol{F}^{\dagger}\|_2\|\boldsymbol{\Delta}\|_2 \;=\; \kappa(\boldsymbol{F})\,\frac{\|\boldsymbol{\Delta}\|_2}{\|\boldsymbol{F}\|_2}\;.$$

If $\boldsymbol{F}$ has full rank and $\eta < 1$ then also $\boldsymbol{F}+\boldsymbol{\Delta}$ has full rank, and

$$\frac{\|\boldsymbol{\gamma}\|_2}{\|\hat{\boldsymbol{x}}\|_2} \;\le\; \frac{\kappa}{1-\eta}\left(\frac{\|\boldsymbol{\delta}\|_2}{\|\boldsymbol{F}\hat{\boldsymbol{x}}\|_2}+\frac{\|\boldsymbol{\Delta}\|_2}{\|\boldsymbol{F}\|_2}\big(1+\kappa\,\frac{\|\hat{\boldsymbol{r}}\|_2}{\|\boldsymbol{F}\hat{\boldsymbol{x}}\|_2}\big)\right)\;,$$

where $\kappa = \kappa(\boldsymbol{F})$ and $\hat{\boldsymbol{r}} = \boldsymbol{y} - \boldsymbol{F}\hat{\boldsymbol{x}}$.

**Proof.** See [3, Section 1.4] or [30, Chapter 9]). □

The last term in the estimate of the relative change disappears if $\boldsymbol{\Delta} = \boldsymbol{0}$ or if the problem is *consistent*, ie, if $\hat{\boldsymbol{r}} = \boldsymbol{0} \;\Leftrightarrow\; \boldsymbol{y} \in \mathcal{R}(\boldsymbol{F})$, cf Figure 5.7. Otherwise, it should be noted that the relative change in the parameters may grow as $(\kappa(\boldsymbol{F}))^2$.

### 5.2.3. Solution via orthogonal transformation

In this section we shall discuss an alternative method for computing the least squares solution to the overdetermined system

$$\boldsymbol{F}\,\boldsymbol{x} \;\simeq\; \boldsymbol{y}\;, \qquad \boldsymbol{F} \in \mathbb{R}^{m\times n}, \quad \boldsymbol{y} \in \mathbb{R}^m, \quad m > n\;.$$

Given a vector $\boldsymbol{u} \in \mathbb{R}^m$ and an orthogonal matrix $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$. The vector $\widetilde{\boldsymbol{u}} = \boldsymbol{Q}^T\boldsymbol{u}$ is an *orthogonal transformation* of $\boldsymbol{u}$. In Appendix A.4 we show that an orthogonal transformation preserves the 2-norm, $\|\boldsymbol{Q}^T\boldsymbol{u}\|_2 = \|\boldsymbol{u}\|_2$, and by proper choice of $\boldsymbol{Q}$ this can be used to simplify the least squares problem.

This choice of $\boldsymbol{Q}$ is given by the *QR factorization* of the matrix $\boldsymbol{F}$, cf (A.21),

$$\boldsymbol{F} \;=\; \boldsymbol{Q}\begin{pmatrix}\boldsymbol{R}\\ \boldsymbol{0}\end{pmatrix} \;=\; \begin{pmatrix}\widehat{\boldsymbol{Q}} & \breve{\boldsymbol{Q}}\end{pmatrix}\begin{pmatrix}\boldsymbol{R}\\ \boldsymbol{0}\end{pmatrix} \;=\; \widehat{\boldsymbol{Q}}\boldsymbol{R}\;. \tag{5.20}$$

Here, $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$ is an orthogonal matrix and $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ is upper triangular. The two submatrices of $\boldsymbol{Q}$ consist respectively of the first $n$ and the last $m-n$ columns, $\widehat{\boldsymbol{Q}} = \boldsymbol{Q}_{:,1:n}$ and $\breve{\boldsymbol{Q}} = \boldsymbol{Q}_{:,n+1:m}$. The last expression in (5.20) is the so-called *economy sized* (or *thin*) QR-factorization.

The matrices $\widehat{\boldsymbol{Q}} \in \mathbb{R}^{m\times n}$ and $\breve{\boldsymbol{Q}} \in \mathbb{R}^{m\times(m-n)}$ have orthonormal columns. They satisfy

$$\widehat{\boldsymbol{Q}}^T\widehat{\boldsymbol{Q}} \;=\; \boldsymbol{I}_{(n)}, \qquad \breve{\boldsymbol{Q}}^T\breve{\boldsymbol{Q}} \;=\; \boldsymbol{I}_{(m-n)}, \qquad \widehat{\boldsymbol{Q}}^T\breve{\boldsymbol{Q}} \;=\; \boldsymbol{0}\;, \tag{5.21}$$

where the index on $\boldsymbol{I}$ gives the size of the identity matrix and $\boldsymbol{0}$ is the $n\times(m-n)$ matrix of all zeros.[1)]

Now we can formulate the alternative method for finding the least squares solution:

**Theorem 5.22.** Let the matrix $\boldsymbol{F}\in\mathbb{R}^{m\times n}$ have the QR factorization (5.20). Then the least squares solution to the overdetermined system $\boldsymbol{F}\boldsymbol{x}\simeq\boldsymbol{y}$ is found by back substitution in the upper triangular system

$$\boldsymbol{R}\,\hat{\boldsymbol{x}} \;=\; \widehat{\boldsymbol{Q}}^T\boldsymbol{y}\;, \tag{5.23}$$

and the corresponding residual satisfies

$$\|\boldsymbol{r}(\hat{\boldsymbol{x}})\|_2 \;=\; \|\breve{\boldsymbol{Q}}^T\boldsymbol{y}\|_2\;.$$

**Proof.** Using the norm-preserving property of an orthogonal transformation and the splitting of $\boldsymbol{Q}$ we see that

$$\|\boldsymbol{r}(\boldsymbol{x})\|_2 \;=\; \|\boldsymbol{Q}^T\,(\boldsymbol{y}-\boldsymbol{F}\,\boldsymbol{x})\,\|_2 \;=\; \left\|\begin{pmatrix}\widehat{\boldsymbol{Q}}^T\boldsymbol{y}-\boldsymbol{R}\boldsymbol{x}\\ \breve{\boldsymbol{Q}}^T\boldsymbol{y}\end{pmatrix}\right\|_2\;.$$

From this expression and the definition (A.1) of the 2-norm it follows that

$$\|\boldsymbol{r}(\boldsymbol{x})\|_2^2 \;=\; \|\widehat{\boldsymbol{Q}}^T\boldsymbol{y}-\boldsymbol{R}\boldsymbol{x}\|_2^2+\|\breve{\boldsymbol{Q}}^T\boldsymbol{y}\|_2^2\;, \tag{5.24}$$

and the minimum norm is obtained when $\widehat{\boldsymbol{Q}}^T\boldsymbol{y}-\boldsymbol{R}\boldsymbol{x}=\boldsymbol{0}$, which is equivalent to (5.23). The expression for the minimum norm of the residual also follows from (5.24). □

**Example 5.9.** The least squares solution defined by Theorem 5.22 is identical to the $\hat{\boldsymbol{x}}$ defined by the normal equations in Theorem 5.14,

$$\boldsymbol{A}\,\boldsymbol{x} \;=\; \boldsymbol{b} \quad\text{with}\quad \boldsymbol{A} \;=\; \boldsymbol{F}^T\boldsymbol{F}\;,\quad \boldsymbol{b} \;=\; \boldsymbol{F}^T\boldsymbol{y}\;.$$

We express $\boldsymbol{F}$ by means of (5.20), and since $\widehat{\boldsymbol{Q}}^T\widehat{\boldsymbol{Q}} \;=\; \boldsymbol{I}$, we get $\boldsymbol{A} \;=\; \boldsymbol{R}^T\widehat{\boldsymbol{Q}}^T\widehat{\boldsymbol{Q}}\,\boldsymbol{R}=\boldsymbol{R}^T\boldsymbol{R}$, so the normal equations are equivalent to

$$\left(\boldsymbol{R}^T\boldsymbol{R}\right)\hat{\boldsymbol{x}} \;=\; \boldsymbol{R}^T\widehat{\boldsymbol{Q}}^T\boldsymbol{y}\;.$$

We assume that $\boldsymbol{F}$ has full rank. Then $(\boldsymbol{R}^T)^{-1}$ exists, and by multiplying with it on both sides of the above equation we get (5.23). Thus, we have shown that the two ways to compute $x^*$ give the same result –

[1)] The two matrices are <u>not</u> orthogonal: both $\widehat{\boldsymbol{Q}}\,\widehat{\boldsymbol{Q}}^T\in\mathbb{R}^{m\times m}$ and $\breve{\boldsymbol{Q}}\,\breve{\boldsymbol{Q}}^T\in\mathbb{R}^{m\times m}$ are different from $\boldsymbol{I}_{(m)}$.

mathematically. The matrix $\boldsymbol{A}$ is symmetric and positive definite, and we can use a *Cholesky factorization* $\boldsymbol{A} = \boldsymbol{C}^T\boldsymbol{C}$ in the solution of the normal equations. $\boldsymbol{C}$ is an upper triangular matrix, and it can be shown that except for "row-wise sign" $\boldsymbol{C}$ is equal to $\boldsymbol{R}$.

When the computation is done in finite precision the solution found via orthogonal transformation is more accurate. In Appendix B.3 we show that

$$\kappa(\boldsymbol{A}) \;=\; (\kappa(\boldsymbol{F}))^2 \;. \tag{5.25}$$

If the problem is *ill-conditioned* (ie $\kappa(\boldsymbol{F})$ is large), this may lead to disastrous effects of rounding errors during the computation of $\hat{\boldsymbol{x}}$. Let $\varepsilon_{\mathsf{M}}$ denote the *machine precision*. With respect to this the problem must be considered as rank deficient if we use the normal equation with $\kappa(\boldsymbol{F}) \gtrsim 1/\sqrt{n\varepsilon_{\mathsf{M}}}$. If the computation is done via orthogonal transformation, the results are reliable if $\kappa(\boldsymbol{F}) \lesssim 1/(\sqrt{mn}\,\varepsilon_{\mathsf{M}})$. As an example let $\varepsilon_{\mathsf{M}} = 2^{-53} \simeq 1.11{\cdot}10^{-16}$, $m = 100$ and $n = 10$ the two conditions are $\kappa(\boldsymbol{F}) \lesssim 3{\cdot}10^7$ and $\kappa(\boldsymbol{F}) \lesssim 3{\cdot}10^{14}$, respectively. ∎

**Example 5.10.** In MATLAB, let `F` and `y` hold the matrix and right-hand side in the overdetermined system (5.9). Then the command

```
x = F \ y
```

returns the least squares solution, computed via orthogonal transformation. ∎

### 5.2.4. Fundamental subspaces

In this section the QR factorization is related to the geometric approach illustrated in Figure 5.7. The matrix $\boldsymbol{F} \in \mathbb{R}^{m\times n}$ can be thought of as the mapping matrix for a linear mapping $\mathbb{R}^n \mapsto \mathbb{R}^m$, and we introduce the following:[2)]

**Definition 5.26.** The *four fundamental subspaces* for the matrix $\boldsymbol{F} \in \mathbb{R}^{m\times n}$ are

$$\begin{aligned}
\textit{Range of } \boldsymbol{F}: &\quad \mathcal{R}(\boldsymbol{F}) &&= \{\boldsymbol{y} \mid \boldsymbol{y} = \boldsymbol{F}\boldsymbol{x},\;\; \boldsymbol{x} \in \mathbb{R}^n\} \\
\textit{Range of } \boldsymbol{F}^T: &\quad \mathcal{R}(\boldsymbol{F}^T) &&= \{\boldsymbol{x} \mid \boldsymbol{x} = \boldsymbol{F}^T\boldsymbol{y},\;\; \boldsymbol{y} \in \mathbb{R}^m\} \\
\textit{Nullspace of } \boldsymbol{F}: &\quad \mathcal{N}(\boldsymbol{F}) &&= \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{F}\boldsymbol{x} = \boldsymbol{0}\} \\
\textit{Nullspace of } \boldsymbol{F}^T: &\quad \mathcal{N}(\boldsymbol{F}^T) &&= \{\boldsymbol{y} \in \mathbb{R}^m \mid \boldsymbol{F}^T\boldsymbol{y} = \boldsymbol{0}\}
\end{aligned}$$

These spaces are subspaces of $\mathbb{R}^m$ or $\mathbb{R}^n$, and if $\boldsymbol{F}$ has *rank* $p$, $p \leq$

[2)] In Linear Algebra textbooks the nullspace is often denoted the *kernel*.

$\min\{m,n\}$, then the spaces have the following dimensions

$$\begin{aligned} \mathcal{R}(\boldsymbol{F}) &\subseteq \mathbb{R}^m, & \dim(\mathcal{R}(\boldsymbol{F})) &= p\ , \\ \mathcal{R}(\boldsymbol{F}^T) &\subseteq \mathbb{R}^n\ , & \dim(\mathcal{R}(\boldsymbol{F}^T)) &= p\ , \\ \mathcal{N}(\boldsymbol{F}) &\subseteq \mathbb{R}^n\ , & \dim(\mathcal{N}(\boldsymbol{F})) &= n-p\ , \\ \mathcal{N}(\boldsymbol{F}^T) &\subseteq \mathbb{R}^m, & \dim(\mathcal{N}(\boldsymbol{F}^T)) &= m-p\ . \end{aligned} \tag{5.27}$$

We shall discuss the two subspaces of $\mathbb{R}^m$ in the case where $m \geq n$ and $\boldsymbol{F}$ has full rank, $p = n$.

**Theorem 5.28.** Let the matrix $\boldsymbol{F} \in \mathbb{R}^{m\times n}$ have the QR factorization

$$\boldsymbol{F} \;=\; \boldsymbol{Q}\begin{pmatrix}\boldsymbol{R}\\ \boldsymbol{0}\end{pmatrix} \;=\; \begin{pmatrix}\widehat{\boldsymbol{Q}} & \breve{\boldsymbol{Q}}\end{pmatrix}\begin{pmatrix}\boldsymbol{R}\\ \boldsymbol{0}\end{pmatrix} \;=\; \widehat{\boldsymbol{Q}}\,\boldsymbol{R}\ ,$$

and assume that $\operatorname{rank}(\boldsymbol{F}) = n \leq m$. Then $\mathcal{R}(\boldsymbol{F})$ and $\mathcal{N}(\boldsymbol{F}^T)$ are spanned by the columns of $\widehat{\boldsymbol{Q}}$ and $\breve{\boldsymbol{Q}}$, respectively:

$$\mathcal{R}(\boldsymbol{F}) \;=\; \operatorname{span}(\boldsymbol{Q}_{:,1:n})\ , \qquad \mathcal{N}(\boldsymbol{F}^T) \;=\; \operatorname{span}(\boldsymbol{Q}_{:,n+1:m})\ .$$

Further, any vector $\boldsymbol{y} \in \mathbb{R}^m$ can be written as the sum of two orthogonal vectors, $\hat{\boldsymbol{y}} \in \mathcal{R}(\boldsymbol{F})$ and $\breve{\boldsymbol{y}} \in \mathcal{N}(\boldsymbol{F}^T)$,

$$\boldsymbol{y} \;=\; \hat{\boldsymbol{y}} + \breve{\boldsymbol{y}} \;=\; \widehat{\boldsymbol{Q}}\,\hat{\boldsymbol{u}} + \breve{\boldsymbol{Q}}\,\breve{\boldsymbol{u}}\ , \tag{5.29}$$

where the coordinates of $\boldsymbol{y}$ with respect to the basis given by the columns in $\boldsymbol{Q}$ can be expressed as follows,

$$\boldsymbol{u} \;=\; \boldsymbol{Q}^T\boldsymbol{y}\ , \qquad \hat{\boldsymbol{u}} \;=\; \widehat{\boldsymbol{Q}}^T\boldsymbol{y}\ , \quad \breve{\boldsymbol{u}} \;=\; \breve{\boldsymbol{Q}}^T\boldsymbol{y}\ .$$

**Proof.** See Appendix B.4. □

**Example 5.11.** Now let $\boldsymbol{y}$ be the right-hand side in the overdetermined system (5.9). The splitting (5.29) is illustrated in Figure 5.7 (for $m = 3$, $n = 2$, where $\vec{\boldsymbol{z}}$ is the geometric vector corresponding to $\hat{\boldsymbol{y}}$ and $\boldsymbol{r}(\hat{\boldsymbol{x}}) = \breve{\boldsymbol{y}}$. The coordinates for $\hat{\boldsymbol{y}}$ are

$$\hat{\boldsymbol{u}} \;=\; \boldsymbol{R}\,\hat{\boldsymbol{x}} \;=\; \widehat{\boldsymbol{Q}}^T\boldsymbol{y}\ .$$

This is seen to agree with (5.23). We return to the splitting in Section 5.3.

■

## 5.3. Statistical aspects

This section deals with statistical aspects of linear data fitting. The notation and basic concepts are introduced in Appendix A.7.

In linear data fitting we are given data points $(t_i, y_i)$, $i = 1, \ldots, m$ and basis functions $\phi_j : \mathbb{R} \mapsto \mathbb{R}$, $j = 1, \ldots, n$. We assume that

$$y_i \;=\; \Upsilon(t_i) + e_i \;, \quad i = 1, \ldots, m \;, \tag{5.30}$$

where $\Upsilon(t)$ is the background function and $e_i$ is an outcome of a random variable $E_i$. We wish to approximate $\Upsilon(t)$ by the linear model

$$M(\boldsymbol{x}, t) \;=\; x_1\phi_1(t) + \ldots + x_n\phi_n(t) \;. \tag{5.31}$$

The residual vector with components $r_i(\boldsymbol{x}) \;=\; y_i - M(\boldsymbol{x}, t_i)$ can be expressed as

$$\boldsymbol{r}(\boldsymbol{x}) \;=\; \boldsymbol{y} - \boldsymbol{F}\,x \;=\; \boldsymbol{\Upsilon} + \boldsymbol{e} - \boldsymbol{F}\,\boldsymbol{x} \;,$$

where $\boldsymbol{\Upsilon} = (\Upsilon(t_1) \;\cdots\; \Upsilon(t_m))^T$, $\boldsymbol{F} \in \mathbb{R}^{m\times n}$ has the elements $(\boldsymbol{F})_{ij} = \phi_j(t_i)$, and $\boldsymbol{e}$ is an outcome of a random $m$-vector $\boldsymbol{E}$ with

$$\mathsf{E}\big[\boldsymbol{E}\big] = \boldsymbol{0} \;, \qquad \mathsf{V}\big[\boldsymbol{E}\big] = \mathsf{E}\big[\boldsymbol{E}\,\boldsymbol{E}^T\big] = \boldsymbol{V} \;.$$

### 5.3.1. Unweighted fit

First we consider the simplest case where

*Assumption A0:* $\quad \mathsf{V}\big[\boldsymbol{E}\big] = \sigma^2\,\boldsymbol{I}$

is satisfied, ie the errors are uncorrelated and all the $E_i$ have the same variance, $\sigma^2$. It can be shown (see for instance [55]) that then the *maximum likelihood estimator* of $\boldsymbol{x}$ is $\hat{\boldsymbol{x}} = \operatorname{argmin}_{\boldsymbol{x}}\{\|\boldsymbol{r}(\boldsymbol{x})\|_2\}$, ie the least squares solution discussed in Section 5.2. The estimator satisfies

$$\boldsymbol{F}^T\boldsymbol{r}(\hat{\boldsymbol{x}}) \;=\; \boldsymbol{0} \;.$$

We want to analyze statistical properties of the solution. In order to simplify the analysis we introduce the QR factorization from Section 5.2.3,

$$\boldsymbol{F} \;=\; \boldsymbol{Q}\,\boldsymbol{R} \;=\; \begin{pmatrix} \widehat{\boldsymbol{Q}} & \breve{\boldsymbol{Q}} \end{pmatrix} \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix} \;=\; \widehat{\boldsymbol{Q}}\boldsymbol{R} \;.$$

According to (5.23) the vector $\hat{\boldsymbol{x}}$ solves the triangular system

$$\boldsymbol{R}\,\hat{\boldsymbol{x}} \;=\; \hat{\boldsymbol{u}} \;; \qquad \hat{\boldsymbol{u}} \;=\; \widehat{\boldsymbol{Q}}^T\boldsymbol{y} \;=\; \widehat{\boldsymbol{Q}}^T(\boldsymbol{\Upsilon} + \boldsymbol{e}) \;, \tag{5.32}$$

and from (A.31) and the orthonormality of the columns in $\widehat{\boldsymbol{Q}}$ it follows that

$$\mathsf{E}\big[\hat{\boldsymbol{u}}\big] \;=\; \widehat{\boldsymbol{Q}}^T\boldsymbol{\Upsilon}\;, \quad \mathsf{V}\big[\hat{\boldsymbol{u}}\big] \;=\; \widehat{\boldsymbol{Q}}^T\left(\sigma^2\boldsymbol{I}_{(m)}\right)\widehat{\boldsymbol{Q}} \;=\; \sigma^2\boldsymbol{I}_{(n)}\;. \tag{5.33}$$

The indices indicate the different sizes of the identity matrices. Further, from (5.32) we see that $\hat{\boldsymbol{x}} = \boldsymbol{R}^{-1}\hat{\boldsymbol{u}}$, and (A.31) and (5.33) imply

$$\mathsf{E}\big[\hat{\boldsymbol{X}}\big] \;=\; \boldsymbol{R}^{-1}\mathsf{E}\big[\hat{\boldsymbol{U}}\big] \;=\; \boldsymbol{R}^{-1}\widehat{\boldsymbol{Q}}^T\boldsymbol{\Upsilon}\;,$$

and by applying (A.30) and exploiting that $\boldsymbol{R}^{-1}$ is upper triangular we get

$$\mathsf{V}\big[\hat{X}_j\big] \;=\; \sigma^2\left([\boldsymbol{R}^{-1}]_{jj}^2 + \cdots + [\boldsymbol{R}^{-1}]_{jn}^2\right)\;.$$

### 5.3.2. Estimating the variance

The residual $\hat{\boldsymbol{r}} = \boldsymbol{r}(\hat{\boldsymbol{x}})$ is orthogonal to the range of $\boldsymbol{F}$, and as in Section 5.2.4 we see that it lies in span($\breve{\boldsymbol{Q}}$),

$$\hat{\boldsymbol{r}} \;=\; \breve{\boldsymbol{\Upsilon}} + \breve{\boldsymbol{Q}}\,\breve{\boldsymbol{u}}\;, \quad \text{where} \quad \breve{\boldsymbol{\Upsilon}} \;=\; \breve{\boldsymbol{Q}}\,\breve{\boldsymbol{Q}}^T\Upsilon\;, \;\; \breve{\boldsymbol{u}} \;=\; \breve{\boldsymbol{Q}}^T\boldsymbol{e}\;.$$

Similar to (5.33) we see that the vector $\breve{\boldsymbol{u}} \in \mathbb{R}^{m-n}$ satisfies

$$\mathsf{E}\big[\breve{\boldsymbol{U}}\big] \;=\; \boldsymbol{0}\;, \qquad \mathsf{V}\big[\breve{\boldsymbol{U}}\big] \;=\; \sigma^2\,\boldsymbol{I}_{(m-n)}\;, \tag{5.34}$$

and by further application of the rules from Section A.7 we get

$$\mathsf{E}\big[\hat{\boldsymbol{r}}\big] \;=\; \breve{\boldsymbol{\Upsilon}}\;, \tag{5.35}$$

$$\begin{aligned}
\mathsf{E}\big[\hat{\boldsymbol{r}}^T\hat{\boldsymbol{r}}\big] \;&=\; \mathsf{E}\big[\breve{\boldsymbol{\Upsilon}}^T\breve{\boldsymbol{\Upsilon}} + 2\breve{\boldsymbol{\Upsilon}}^T\breve{\boldsymbol{Q}}\,\breve{\boldsymbol{u}} + (\breve{\boldsymbol{Q}}\,\breve{\boldsymbol{u}})^T(\breve{\boldsymbol{Q}}\,\breve{\boldsymbol{u}})\big] \\
&=\; \breve{\boldsymbol{\Upsilon}}^T\breve{\boldsymbol{\Upsilon}} + \boldsymbol{0} + \mathsf{E}\big[\breve{\boldsymbol{u}}^T\breve{\boldsymbol{Q}}^T\breve{\boldsymbol{Q}}\,\breve{\boldsymbol{u}}\big] \\
&=\; \|\breve{\boldsymbol{\Upsilon}}\|_2^2 + \mathsf{E}\big[\|\breve{\boldsymbol{u}}\|_2^2\big] \;=\; \|\breve{\boldsymbol{\Upsilon}}\|_2^2 + (m{-}n)\sigma^2\;.
\end{aligned} \tag{5.36}$$

The vector $\breve{\boldsymbol{\Upsilon}}$, the part of $\boldsymbol{\Upsilon}$ which is orthogonal to $\mathcal{R}(\boldsymbol{F})$, is identified as the vector of *approximation errors*. If this is zero, it follows that

$$\sigma^2 \;\simeq\; v_n \;\equiv\; \frac{\|\boldsymbol{r}(\hat{\boldsymbol{x}})\|_2^2}{m-n}\;. \tag{5.37}$$

If $\breve{\boldsymbol{\Upsilon}} \neq \boldsymbol{0}$, then (5.36) shows that the estimate $v_n$ defined by (5.37) can be expected to give a value larger than $\sigma^2$.

**Example 5.12.** We have used polynomials of degree $0, 1, \ldots, 9$ (ie $n = 1, 2, \ldots, 10$ parameters) to fit the 20 data points in Example 5.3. Figure 5.8 shows the values for the estimated variance, as computed by (5.37).

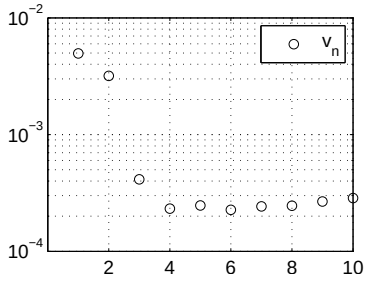



***Figure 5.8.*** *Variance estimates.*

We see that approximation error has marked influence on the first three values, while $v_n \simeq 2.2{\cdot}10^{-4}$ for $n \geq 4$.

We give more examples of the behaviour of $v_n$ in Sections 5.6 and 5.7. ∎

## 5.4. Weighted least squares

In this section we replace Assumption A0 by

$$\textit{Assumption A1:} \qquad \mathsf{V}\big[\boldsymbol{E}\,\boldsymbol{E}^T\big] \;=\; \mathrm{diag}(\sigma_1^2, \ldots, \sigma_m^2)\;,$$

where the $\sigma_i^2$ are known, except maybe for a common factor. Thus, we still assume uncorrelated errors. We introduce *weights*

$$\boldsymbol{W} \;=\; \mathrm{diag}(w_1, \ldots, w_m) \quad \text{with} \quad w_i \;=\; \frac{K}{\sigma_i}\;,$$

where $K$ is some constant, and see that the transformed problem

$$\boldsymbol{W}\,\boldsymbol{y} \;=\; \boldsymbol{W}\,\boldsymbol{\Upsilon} + \boldsymbol{W}\,\boldsymbol{e}$$

satisfies $\mathsf{E}\big[(\boldsymbol{W}\,\boldsymbol{E})(\boldsymbol{W}\,\boldsymbol{E})^T\big] = K^2\,\boldsymbol{I}$. Now we fit the data (5.30) with the model (5.31) by means of *weighted least squares*,

$$\hat{\boldsymbol{x}}_{\mathbf{w}} \;=\; \mathrm{argmin}_{\boldsymbol{x}}\{\|\boldsymbol{W}(\boldsymbol{y} - \boldsymbol{F}\,\boldsymbol{x})\|_2\}\;. \tag{5.38}$$

Proceeding as in Section 5.2.3 we compute the QR factorization of the weighted matrix

$$\boldsymbol{W}\boldsymbol{F} \;=\; \widehat{\boldsymbol{Q}}\boldsymbol{R}\;,$$

and find the solution by back substitution in

$$\boldsymbol{R}\,\hat{\boldsymbol{x}}_{\mathbf{w}} \;=\; \widehat{\boldsymbol{Q}}^T\boldsymbol{W}\,\boldsymbol{y}\;.$$

Further, we find that (5.33)–(5.34) hold with $\sigma^2$ replaced by $K^2$. This means that

$$K^2 \;\simeq\; v_n \;\equiv\; \frac{1}{m-n}\sum_{i=1}^{m} w_i^2\,(y_i - M(\hat{\boldsymbol{x}}_{\mathbf{w}}, t_i))^2\;. \tag{5.39}$$

This estimate presumes that the approximation error is zero. If not, then (5.39) is an overestimate of $K^2$.

We end this section by some examples that may be helpful in the proper *choice of weights*.

**Example 5.13. Known variance.** If the $\{y_i\}$ can be expected to have the same absolute or the same relative error, then the relevant choice of weights is $w_i = 1$ and $w_i = 1/y_i$, respectively. If the errors can be expected to follow a *Poisson distribution* (which, for instance, is the case with results from a Geiger counter), the relevant weights are $w_i = 1/\sqrt{y_i}$. ∎

**Example 5.14. Estimate variance.** The following algorithm, inspired by [45, Section 3], may sometimes be useful for finding appropriate weights.

If the data points are sufficiently close, relative to the variation in the background function, then a *piecewise quadratic* can give a good approximation to the local behaviour of $\Upsilon$. To determine the piecewise quadratic we select $q$, for instance $q=2$ or $q=3$, and for $i = q{+}1, \ldots, m{-}q$ we fit a parabola to the data points $(t_j, y_j)$, $j = i{-}q, \ldots, i{-}q$. We can assume that the "noise level is almost constant in this region, and use unweighted fitting to determine the parabola, and (5.37) to compute $s_i^2$, an estimate of the local variance.

Now, we have points $(t_i, s_i)$, $i = q{+}1, \ldots, m{-}q$, where $s_i = \sigma_i + \varepsilon_i$, with $\sigma_i$ being the local *standard deviation*, and $\varepsilon_i$ comes from the errors in the data. We can fit the points $(t_i, s_i)$ by some model $S(\boldsymbol{z}, t)$, and use this to compute weights

$$w_i \;=\; \frac{1}{S(\hat{\boldsymbol{z}}, t_i)}\;, \quad i = 1, \ldots, m\;.$$

As a specific example consider the data[3)] in Figure 5.9.

[3)] The data is an OSL (optically stimulated luminescence) measurement performed at Risø National Laboratory for Sustainable Energy.

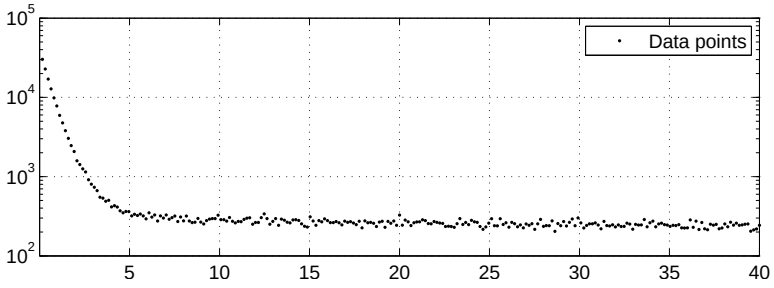



***Figure 5.9.*** *OSL data. Notice the logarithmic scale.*

This kind of data is often fitted with weights $w_i = 1/\sqrt{y_i}$, cf Example 5.13. However, the long, flat tail seems to include some wild points with exceptionally small $y_i$-values, and they would get undesirable large weights.

Figure 5.10 shows the result of applying the above algorithm with $q=3$ and the model

$$S(\boldsymbol{z},t) \;=\; z_1 + z_2 e^{z_3 t} \;.$$

This is a nonlinear model, and we used the method from Example 6.20 to get the least squares fit: $\hat{\boldsymbol{z}} = \begin{pmatrix} 19.7 & 1490 & -1.89 \end{pmatrix}^T$.

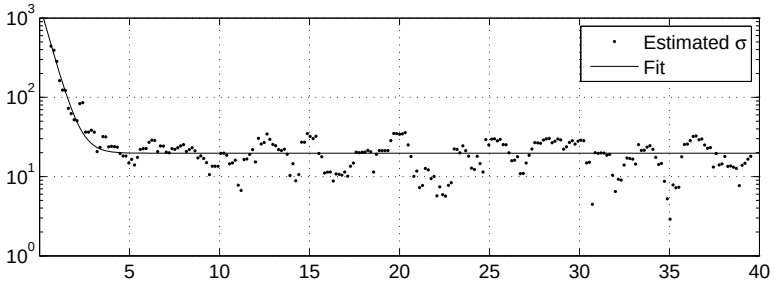



***Figure 5.10.*** *Estimated standard deviation.*

We return to this OSL problem in Example 6.22. ∎

**Example 5.15. Transformation of variables.** Instead of $\{(t_i, y_i)\}_{i=1}^m$ we fit $\{(t_i, \eta_i)\}_{i=1}^m$, where $\eta_i = \psi(y_i)$.

We still assume that $\mathsf{E}\big[Y_i\big] = \Upsilon(t_i)$, and (A.30) implies

$$\mathsf{V}\big[\psi(Y_i)\big] \;=\; \big\{\psi'(\Upsilon(t_i))\big\}^2\, \mathsf{V}\big[Y_i\big] \;.$$

Therefore, if $\mathsf{E}\big[w_i^2 E_i^2\big] = K^2$, then

$$\omega_i \;=\; \frac{w_i}{|\psi'(y_i)|} \;, \quad i = 1, \ldots, m \tag{5.40}$$

are the relevant weights for the transformed problem. ∎

**Example 5.16. Exponential fit.** Suppose that all the $y_i > 0$, and we want to use the nonlinear model $M(x,t) = x_1 e^{x_2 t}$, ie we want to find the parameters so that

$$x_1 e^{x_2 t_i} \simeq y_i , \quad i = 1, \ldots, m . \tag{5.41}$$

Taking the natural logarithm on both sides we get the linear problem

$$\log x_1 + x_2\, t_i \simeq \log y_i , \quad i = 1, \ldots, m . \tag{5.42}$$

Thus, we use $\psi(y) = \log y \;\Rightarrow\; \psi'(y) = 1/y$. Therefore, if $w_i$ is the appropriate weight for (5.41), then $\omega_i = w_i y_i$ is the appropriate weight for (5.42). ∎

**Example 5.17. Uncertainty in the abscissas.** Instead of $\{(\tau_i, \Upsilon_i)\}$ we are given $\{(t_i, y_i)\}$ with

$$\begin{aligned} t_i &= \tau_i + d_i, \qquad \mathsf{E}\big[D_i\big] = 0, \;\; \mathsf{E}\big[D_i^2\big] = \chi_i^2 , \\ y_i &= \Upsilon(\tau_i) + e_i, \;\; \mathsf{E}\big[E_i\big] = 0, \;\; \mathsf{E}\big[E_i^2\big] = \sigma_i^2 , \end{aligned}$$

cf Figure 5.11 below.

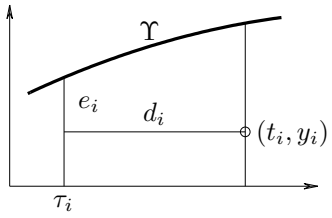



***Figure 5.11.*** *Uncertain abscissa.*

By Taylor's theorem we get

$$y_i \;=\; \Upsilon(\tau_i) + e_i \;\simeq\; \Upsilon(t_i) - d_i \Upsilon'(t_i) + e_i ,$$

so that $\mathsf{V}\big[Y_i\big] \simeq \mathsf{V}\big[E_i\big] + \big(\Upsilon'(t_i)\big)^2\, \mathsf{V}\big[D_i\big]$. Therefore, in this case the relevant weights are given by

$$w_i \;=\; \frac{K}{\sqrt{\sigma_i^2 + [\Upsilon'(t_i)]^2 \chi_i^2}} , \quad i = 1, \ldots, m . \tag{5.43}$$

In practice we do not know $\Upsilon'(t_i)$ but have to approximate it by $M'(\hat{\boldsymbol{x}}_{\mathbf{w}}, t_i)$. This implies that the determination of $\hat{\boldsymbol{x}}_{\mathbf{w}}$ is a nonlinear problem, which must be solved by iteration.

Instead of using the above approach this type of data fitting problem is often handled by the "*total least squares*" method, based on minimizing the sum of squares of orthogonal distances between data points and fitting curve. In the unweighted cases this leads to

$$\min_{\boldsymbol{x},\boldsymbol{d}} \sum_{i=1}^{m} \Big[ \big(y_i - M(\boldsymbol{x}, t_i + d_i)\big)^2 + d_i^2 \Big] .$$

See for instance [27]. ∎

**Example 5.18. Fit with constraints.** Suppose that the fit $M(\hat{\boldsymbol{x}}, t)$ has to satisfy the constraints

$$\begin{aligned} M(\hat{\boldsymbol{x}}, t_i) &= \Upsilon_i, \quad i = m{+}1, \ldots, m{+}q \ , \\ M'(\hat{\boldsymbol{x}}, t_i) &= \Upsilon'_i, \quad i = m{+}q{+}1, \ldots, m{+}q{+}p \ , \end{aligned} \tag{5.44}$$

where prime denotes differentiation with respect to $t$ and the $\{(t_i, \Upsilon_i)\}_{i=m+1}^{m+q}$ and $\{(t_i, \Upsilon'_i)\}_{i=m+q+1}^{m+q+p}$ are given. We assume that the constraints are linearly independent and that $q{+}p < n$, so that there still is freedom to determine $\boldsymbol{x}$ so as to get close to the data points. The "best" $\boldsymbol{x}$ can be found by an algorithm for general constrained optimization, see Chapter **??**. However, there are simple methods for this special case.

**Method 1.** (Assumes $p = 0$, ie, no constraint on $M'$): Extend the set of data $\{(t_i, y_i), w_i\}_{i=1}^{m}$ with $\{(t_i, \Upsilon_i), w_i\}_{i=m+1}^{m+q}$, where $w_i = w_\infty$ for $i > m$. In theory $w_\infty$ should be infinitely large, so that $\hat{\boldsymbol{x}}_{\mathbf{w}}$ computed by (5.38) gives $\Upsilon_i - M(\hat{\boldsymbol{x}}_{\mathbf{w}}, t_i) = 0$ for $i > m$. In practice it suffices that $w_\infty > \varepsilon_{\mathsf{M}}{}^{-1} \cdot \max_{i \le m}\{w_i\}$, where $\varepsilon_{\mathsf{M}}$ is the machine precision. See [49] for a discussion of practical aspects.

**Method 2.** Let $\begin{aligned} M(\boldsymbol{x}, t) &= \mu(t) + \nu(t) N(\boldsymbol{x}, t) \quad \Rightarrow \\ M'(\boldsymbol{x}, t) &= \mu'(t) + \nu'(t) N(\boldsymbol{x}, t) + \nu(t) N'(\boldsymbol{x}, t) \ , \end{aligned}$

with the functions $\mu(t)$ and $\nu(t)$ chosen so that the constraints (5.44) are satisfied for all $N(\boldsymbol{x}, t)$. If $\nu(t_i) = 0$ for $i > m{+}q$, then we do not need to consider $N'$. $\hat{\boldsymbol{x}}_{\mathbf{w}}$ is found as the weighted least squares fit to

$$\{(t_i, \eta_i),\ \omega_i\} = \{\left(t_i, \frac{y_i - \mu(t_i)}{\nu(t_i)}\right),\ |\nu(t_i)| \cdot w_i\}, \quad i = 1, \ldots, m \ .$$

The weights were found by (5.40). Note that $\nu(t_i) = 0 \ \Rightarrow \ \omega_i = 0$, which implies that the data point is ignored and we do not have problems with the corresponding $\eta_i$ being undefined.

**Method 3.** Combination of Methods 1 and 2: Use $w_\infty$ in points where $\Upsilon'$ is not given.

If, for instance, we are given the constraints $\Upsilon(0) = b$, $\Upsilon'(0) = c$, $\Upsilon(1) = d$, we may use

$$\mu(t) = b, \quad \nu(t) = t \quad \text{and an extra data point} \quad (1, d), w_\infty \ ,$$

or – without extra data points:

$$\mu(t) = (d{-}b{-}c)t^2 + ct + b, \quad \nu(t) = t^2(t{-}1) \ .$$

■

## 5.5. Generalized least squares

In the general case the variance–covariance matrix

$$\mathsf{V}\big[\boldsymbol{E}\big] \;=\; \mathsf{E}\big[\boldsymbol{E}\,\boldsymbol{E}^T\big] \;=\; \sigma^2\,\boldsymbol{V}$$

is not diagonal, but the following theorem holds.

**Theorem 5.45.** The *variance–covariance matrix* $\sigma^2\,\boldsymbol{V}$ is SPSD: symmetric and positive semidefinite.

**Proof.** Symmetry follows from $\mathsf{E}\big[E_iE_j\big] = \mathsf{E}\big[E_jE_i\big]$, and applying the rules from Section A.7 to the linear function

$$u(\boldsymbol{y}) \;=\; \boldsymbol{a}^T\boldsymbol{y} \;=\; \boldsymbol{a}^T\overline{\boldsymbol{y}} + z \quad \text{with} \quad z \;=\; \boldsymbol{a}^T\boldsymbol{e}\;,$$

we find

$$\mathsf{E}\big[Z^2\big] \;=\; \mathsf{E}\big[\boldsymbol{a}^T\boldsymbol{E}\;\boldsymbol{E}^T\boldsymbol{a}\big] \;=\; \boldsymbol{a}^T\,\mathsf{E}\big[\boldsymbol{E}\,\boldsymbol{E}^T\big]\,\boldsymbol{a} \;=\; \sigma^2\;\boldsymbol{a}^T\boldsymbol{V}\,\boldsymbol{a}\;.$$

This is nonnegative, ie $\boldsymbol{a}^T\boldsymbol{V}\,\boldsymbol{a} \geq 0$ for all $\boldsymbol{a}$, and this is the condition for $\boldsymbol{V}$ to be positive semidefinite. □

Because $\boldsymbol{V}$ is SSPD, its Cholesky factorization

$$\boldsymbol{V} \;=\; \boldsymbol{C}^T\boldsymbol{C}$$

exists, cf Appendix A.2. Now assume that $\boldsymbol{C}$ is nonsingular (equivalent to V being SPD), then we can use the "de-correlation transformation"transformation, de-correlation

$$\tilde{\boldsymbol{y}} \;=\; \boldsymbol{C}^{-T}\boldsymbol{y} \;=\tilde{\boldsymbol{\Upsilon}} + \tilde{\boldsymbol{e}}\;, \tag{5.46}$$

where $\tilde{\boldsymbol{\Upsilon}} = \boldsymbol{C}^{-T}\boldsymbol{\Upsilon}$ and $\tilde{\boldsymbol{e}} = \boldsymbol{C}^{-T}\boldsymbol{e}$. The rules from Section A.7 imply that $\mathsf{E}\big[\tilde{\boldsymbol{E}}\big] \;=\; \boldsymbol{0}$ and

$$\mathsf{E}\big[\tilde{\boldsymbol{E}}\,\breve{E}^T\big] \;=\; \boldsymbol{C}^{-T}\mathsf{E}\big[E\,E^T\big]\boldsymbol{C}^{-1} \;=\; \boldsymbol{C}^{-T}\;(\sigma^2\boldsymbol{V})\;\boldsymbol{C}^{-1} \;=\; \sigma^2\,\boldsymbol{I}\;.$$

Therefore, the generalized least squares solution to the overdetermined system $\boldsymbol{F}\boldsymbol{x} \simeq \boldsymbol{y}$ can be found by applying orthogonal transformation, to the modified system

$$\tilde{\boldsymbol{F}}\boldsymbol{x} \;\simeq\; \tilde{\boldsymbol{y}} \quad \text{with} \quad \tilde{\boldsymbol{F}} \;=\; \boldsymbol{C}^{-T}\boldsymbol{F}\;, \quad \tilde{\boldsymbol{y}} \;=\; \boldsymbol{C}^{-T}\boldsymbol{y}\;.$$

Note that the modified matrices are computed by forward substitution in the lower triangular systems

$$\boldsymbol{C}^T\,\tilde{\boldsymbol{F}} \;=\; \boldsymbol{F}\;, \qquad \boldsymbol{C}^T\,\tilde{\boldsymbol{y}} \;=\; \boldsymbol{y}\;.$$

In cases where the correlation matrix $\boldsymbol{V}$ is singular, the computation of the "best" fit is more complex, see for instance [3, Chapter 2].

## 5.6. Fitting with polynomials

We finish this chapter by discussing two choices of basis functions for data representation: polynomials and cubic splines. Both choices satisfy requirements 3° – 5° on page 77. Further, we shall assume that the background function is a "smooth" function, and such a function can be approximated well with both choices, so also requirement 1° can be satisfied.

We already discussed fitting with polynomials in several examples. Given data points $\{(t_i, y_i)\}_{i=1}^m$, we want to find a least squares fit with a polynomial

$$M(\boldsymbol{x}, t) \;=\; x_1 p_{n-1}(t) + x_2 p_{n-2}(t) + \cdots + x_n p_0(t) \;, \tag{5.47}$$

where each $p_k$ is a polynomial of degree $k$, implying that $M(\boldsymbol{x}, t)$ is a polynomial of degree at most $n-1$. There is nothing special about this problem if $n$ were known: we just compute the least squares fit with basis functions $\phi_j(t) = p_{n-j}(t)$. In this section, however, we shall concentrate on the data representation aspect, where $n$ is not given beforehand.

If $n$ is small, the polynomial may be too "stiff" to follow the variations in the background function $\Upsilon$, and (some of) the residuals, cf (5.6),

$$r_i(\boldsymbol{x}) \;=\; \big(y_i - \Upsilon(t_i)\big) + \big(\Upsilon(t_i) - M(\boldsymbol{x}, t_i)\big) \;\equiv\; e_i + \alpha_i \;,$$

will be dominated by approximation errors $\alpha_i$. For large $n$ the polynomial is so "elastic" that it follows not only $\Upsilon$ but also the noise, cf Figure 5.3. The aim is to find $n$ so that the level of approximation errors is of the same order of magnitude as the "noise level".[4)]

This goal may be reached interactively: Start with a low order polynomial and increase $n$ as long as a visual inspection of the data and fit indicate a better approximation. In this context a plot of the residuals may be helpful, it acts like a "magnifying glass".

**Example 5.19.** Consider the data from Example 5.3. Figure 5.12 gives results for increasing values of $n$.

The 1st degree polynomial ($n = 2$) is too stiff. One way to realize that is that in most cases two neighbouring residuals have the same sign – we say that there are *"trends"*.

[4)] The *"noise level"* will be made more precise on page **??**.

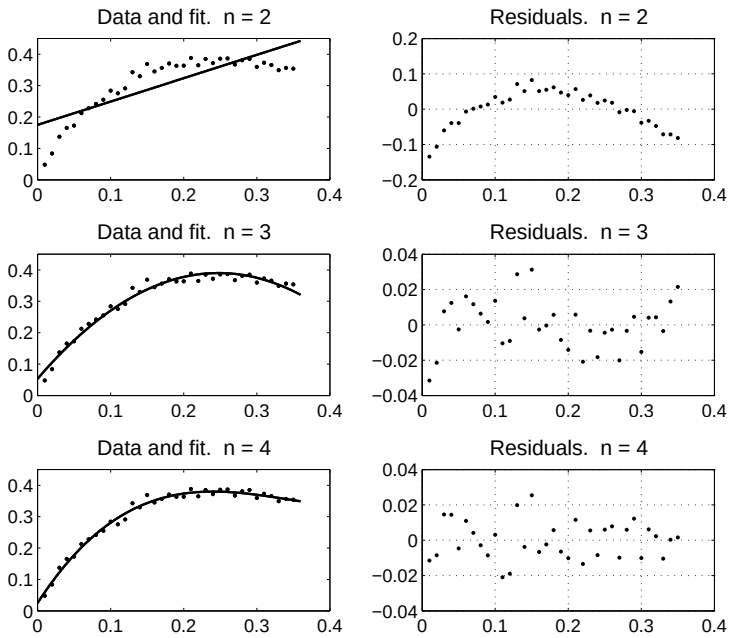



***Figure 5.12.*** *Fit with polynomials of degree 1, 2 and 3.*

The trends seem to have disappeared for $n = 4$, so this value might be a good choice.

This choice is confirmed by the approximation errors shown alongside. The approximation $P_3$ is significantly better than $P_2$ and has maximum error slightly smaller than $P_4$.

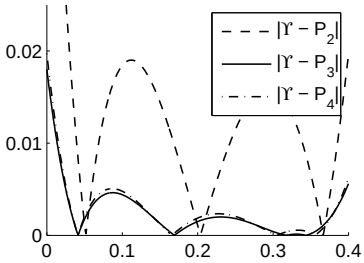



***Figure 5.13.*** *Approximation errors.*

■

The proper choice of $n$ can be quantified: As discussed in Section 5.3 we can compute the variance estimate (5.37)

$$v_n \;=\; \frac{1}{m-n}\|\hat{\boldsymbol{r}}\|_2^2 \;=\; \frac{1}{m-n}\sum_{i=1}^{m}\left(r_i(\hat{\boldsymbol{x}})\right)^2 \quad .$$

If $n$ is too small, then $v_n > v$, but (under certain statistical assumptions about the $\{e_i\}$) the value decreases for larger $n$ and settles at an almost constant value when we have passed the optimal value of $n$. This is

where the level of approximation errors is below the noise level.

There are a number of statistical tests for trends. We shall only present the following intuitive method, which assumes that the data points are ordered by increasing value of $t_i$: Compute the ratio between the average product of two consecutive residuals and the variance estimate,

$$R_n \;=\; \frac{1}{(m-1)v_n} \sum_{i=1}^{m-1} \hat{r}_i \hat{r}_{i+1} \;.$$

Statistics tell us that there is a trend if this ratio is larger than the threshold $1/\sqrt{2(m-1)}$, so we shall use the following *measure of trend*

$$T_n \;=\; \sqrt{2(m-1)}\, R_n \;. \tag{5.48}$$

A value $T_n \geq 1$ indicates that the residuals are dominated by approximation errors. In other words, the polynomial is too stiff and should be "softened" by increasing the degree.

**Example 5.20.** For the data in Examples 5.3 and 5.19 we have computed $v_n$ and $T_n$ for $n = 1, \ldots, 10$. The results are shown in Figure 5.14.

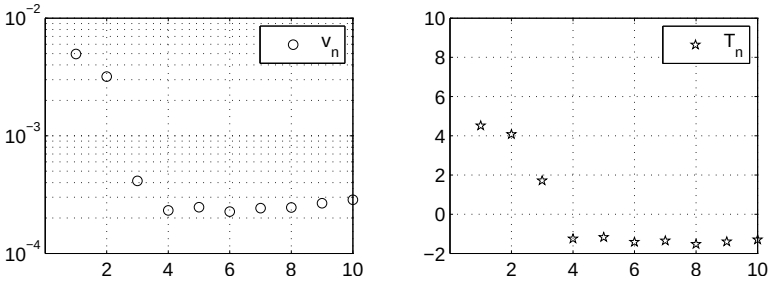



***Figure 5.14.*** *Values of $v_n$ (5.37), and $T_n$ (5.48). The first 20 data points from Figure 5.2.*

As expected, $v_n$ starts by decreasing. For $n \geq 4$ all the $v_n$ are almost constant. Thus, this method gives the same optimal value $n = 4$ as the interactive method in the previous example. Also the trend measure suggests $n = 4$ as the best value, since this is the smallest $n$-value for which $T_n < 1$.

A similar analysis with all the data from Figure 5.2 gives the results shown in Figure 5.15.

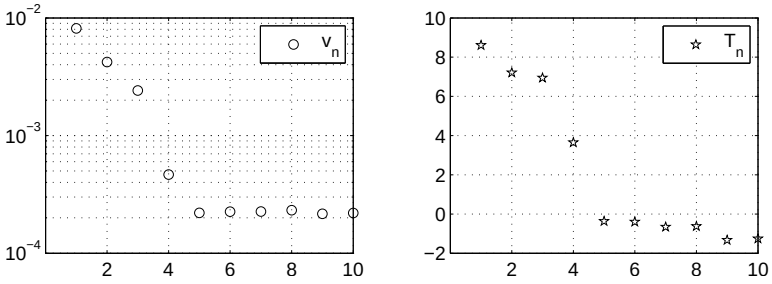



***Figure 5.15.*** *Values of $v_n$ (5.37), and $T_n$ (5.48). All 45 data points from Figure 5.2.*

Here the estimated variance and trend measure agree on the choice $n = 5$. The approximation errors for the cases $n = 4, 5, 6$ are shown in Figure 5.16. $n = 6$ seems to be slightly better than $n = 5$. As in Figure 5.13 the error is largest close to the ends of the domain of data abscissas; this is typical for data fitting problems.

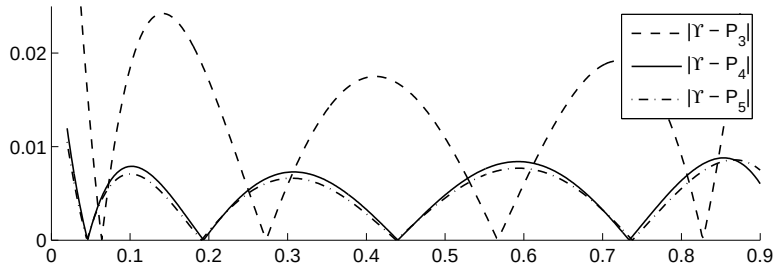



***Figure 5.16.*** *Approximation errors.* ■

Each $p_k$ in (5.47) is a polynomial of exact degree $k$, and since $m > n$ it follows that the matrix $\boldsymbol{F}$ has full rank, $\kappa(\boldsymbol{F}) < \infty$. The above discussion is independent of how we represent these basis functions, but for practical computation this aspect is important, as we shall demonstrate for data abscissas in the domain

$$[a, b] \;=\; [c-d, c+d]; \quad c \;=\; \tfrac{1}{2}(a+b), \quad d \;=\; \tfrac{1}{2}(b-a)\;. \tag{5.49}$$

The simplest choice is to take *simple polynomials*, $p_k(t) = t^k$. Figure 5.17 shows them in the two domains $[a, b] = [-1, 1]$ and $[a, b] = [99, 101]$.

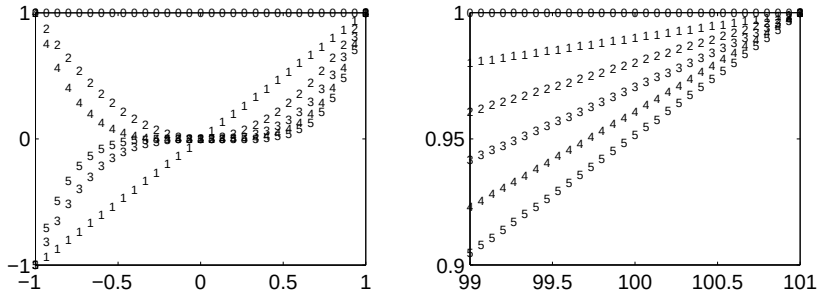



***Figure 5.17.*** *Simple polynomials on $[a, b]$, normalized by division with $p_k(b)$.*

This illustrates that if $|c| \gg d$, then all $t^k$ for $k \geq 1$ behave almost like a first order polynomial for $t \in [a, b]$. Therefore the columns of $\boldsymbol{F}$ are almost linearly dependent (if $n > 2$) and $\kappa(\boldsymbol{F})$ will be very large. The discussion in Section 5.2.2 shows that then the computed results are unreliable.

**Example 5.21.** Suppose we have $m = 100$ data points with the abscissas equidistant in $[a, b]$. Then the condition numbers $\kappa(\boldsymbol{F})$ are as follows (with "–" denoting that $\kappa(\boldsymbol{F}) > 0.1/\varepsilon_{\mathsf{M}} \simeq$ `4.5e+15`. This indicates that the *efficient rank* of $\boldsymbol{F}$ is less than $n$).

| $n$ | $[-1, 1]$ | $[0, 2]$ | $[9, 11]$ | $[99, 101]$ | $[999, 1001]$ |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1.7e+00 | 3.7e+00 | 1.7e+02 | 1.7e+04 | 1.7e+06 |
| 3 | 3.7e+00 | 1.8e+01 | 3.4e+04 | 3.3e+08 | 3.3e+12 |
| 4 | 8.0e+00 | 1.0e+02 | 6.9e+06 | 6.4e+12 | – |
| 5 | 1.8e+01 | 6.0e+02 | 1.4e+09 | – | – |
| 6 | 4.2e+01 | 3.8e+03 | 3.0e+11 | – | – |
| 7 | 9.8e+01 | 2.4e+04 | 6.4e+13 | – | – |
| 8 | 2.3e+02 | 1.6e+05 | – | – | – |
| 9 | 5.3e+02 | 1.1e+06 | – | – | – |
| 10 | 1.2e+03 | 7.4e+06 | – | – | – |
| 11 | 2.9e+03 | 5.1e+07 | – | – | – |

***Table 5.6.1.*** *$\kappa(\boldsymbol{F})$ for simple polynomials.* ∎

A simple cure is to replace the simple polynomials by

$$p_k(t) \;=\; \left(\frac{t-c}{d}\right)^k, \quad k = 0, 1, \ldots, n-1 \;, \tag{5.50}$$

where $c = \frac{1}{2}(\min\{t_i\} + \max\{t_i\})$ and $d = \frac{1}{2}(\max\{t_i\} - \min\{t_i\})$, cf (5.49). Essentially this corresponds to simple polynomials on $[-1, 1]$ irrespective of the domain of data abscissas. Figure 5.17 and Example 5.21 indicate that here the problems with almost linear dependency is much less pronounced.

It is possible to generate $\{p_k\}$ so that they are *orthogonal* over the given $\{t_i\}$, cf Example 5.8, see for instance [18, Section 9.5]. This would be advantageous in the present context where we have to recompute the fit for several values of $n$. As a *rule of thumb*, however, we should never use a polynomial of degree larger than 10 ($n \leq 11$), and Example 5.21 indicates that the polynomials (5.50) are sufficiently linearly independent. Further, it is relatively simple to update the QR factorization of $\boldsymbol{F}$ when $n$ is increased, ie when an extra column is added to $\boldsymbol{F}$.

## 5.7. Fitting with cubic splines

In this section we shall discuss a tool which is suited for data fitting in cases where we are not given a fitting model and where polynomials are not suited. Throughout the section we shall use the data points in Figure 5.18 as a representative example.

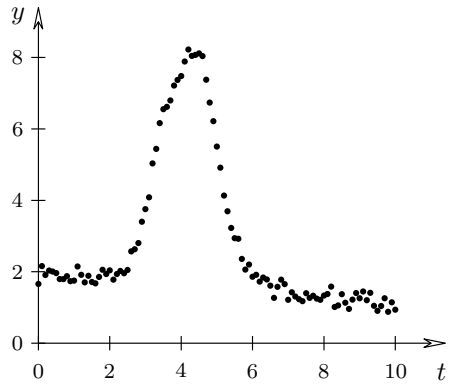



***Figure 5.18.*** *Data points from a function with different nature in different regions.*

**Example 5.22.** If we try to use polynomial fitting, we get the variance estimates and trend measures shown n Figure 5.19.

Even for $n = 20$ the $v_n$ have not settled, and $T_{20} \simeq 3.7$ indicates that there still is a trend. The (small) jump between $v_{13}$ and $v_{14}$ makes us look closer at the case $n = 14$, ie the least squares fit with a polynomial of degree 13. This is shown in Figure 5.20, and we see that the middle part is quite well approximated, except that we miss some details close to the peak; we would need a polynomial of higher degree to fix that. At both ends, however, the fit exhibits unexpected oscillations, indicating that the degree is too high.

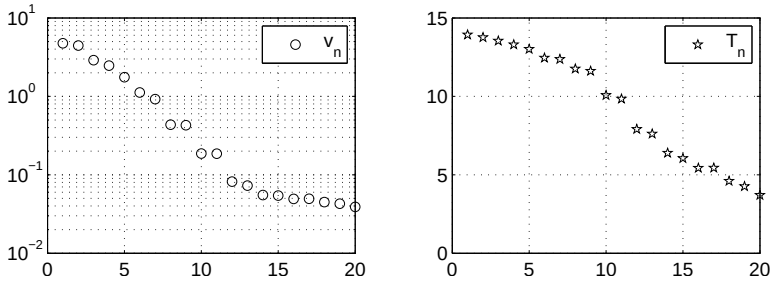



***Figure 5.19.*** *Values of $v_n$ (5.37), and $T_n$ (5.48). The 101 data points from Figure 5.18.*

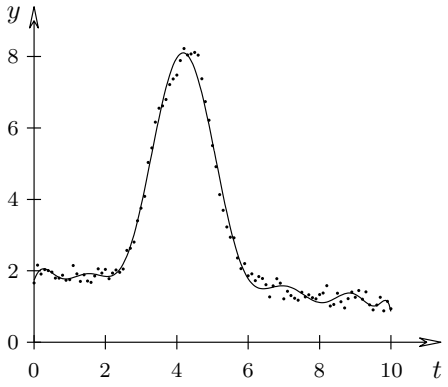



***Figure 5.20.*** *Least squares fit to the data in Figure 5.18. Degree 13 polynomial.*

■

The slow convergence of the variance estimates and trend measures, and the simultaneous occurrence of subregions where the degree is too low and other subregions where the degree is too high is typical for polynomial fits to data where the background function has different nature in different subregions.

For such problems it is better to use *cubic splines*.[5)] We start by giving a short introduction here. More details are given in Appendix A.8, and for even more information see [18, Sections 5.11 – 5.12] or the monographs [13, 15]

[5)] For short: *splines*. More precisely: *cubic spline functions*.

A cubic spline $s$ is a piecewise cubic with breakpoints, so-called *knots*

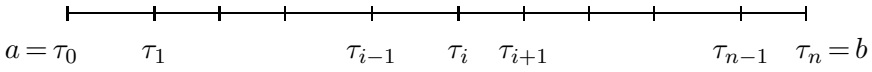



***Figure 5.21.*** *Knots.*

The spline and its first two derivatives are continuous on the whole interval $[\tau_0, \tau_n]$, implying that we get a smooth curve, but in general the piecewise constant $s'''(t)$ jumps as $t$ passes one of the interior knots $\tau_1, \ldots, \tau_{n-1}$. This property enables the spline to focus on the local nature of $\Upsilon$, with less influence from the global behaviour than a polynomial has.

The spline with $n$ knot intervals has $n+3$ *degrees of freedom*, and it can be expressed as

$$s(t) \;=\; \sum_{j=1}^{n+3} c_j \, B_j(\boldsymbol{\tau}, t) \;, \quad \tau_0 \le t \le \tau_n \;, \tag{5.51}$$

where the $B_j$ are so-called *basis splines*, *B-splines* for short. The values of $B_j(\boldsymbol{\tau}, t)$ depend on $t$ and the knots $\boldsymbol{\tau}$. The formulation (5.51) is convenient for determining a spline as a least squares fit to data points $\{(t_i, y_i)\}_{i=1}^m$ : The coefficients $\boldsymbol{c} = \begin{pmatrix} c_1 & \ldots & c_{n+3} \end{pmatrix}^T$ are the least squares solution to the overdetermined system

$$\boldsymbol{F}\,\boldsymbol{c} \;\simeq\; \boldsymbol{y} \;, \qquad \big(\boldsymbol{F}\big)_{ij} = B_j(\boldsymbol{\tau}, t_i) \;. \tag{5.52}$$

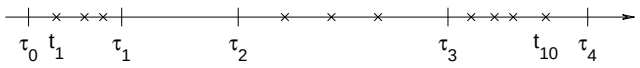
Thus, for given knots, this is a problem of the form discussed in Section 5.2. The proof of the following theorem is given in [10].

**Theorem 5.53.** Problem (5.52) has a unique least squares solution $\boldsymbol{c}^*$ if $m \ge n+3$ and a subset $\widetilde{t}_1, \ldots, \widetilde{t}_{n+3}$ of the data abscissas $t_1, \ldots, t_m$ satisfies

$$\widetilde{t}_k < \tau_k < \widetilde{t}_{k+4} \quad \text{for} \quad k = 1, \ldots, n-1 \;.$$

In the quadratic case $m = n+3$ these conditions are known as the *Schoenberg-Whitney conditions.*

**Example 5.23.** The B-splines have local support: $B_j(\boldsymbol{\tau}, t)$ is nonzero only for $\tau_{j-4} < t < \tau_j$. Assuming that the data points are ordered such that $t_i > t_{i-1}$, the matrix $\boldsymbol{F}$ has a "generalized" band structure. For the distribution of knots and data abscissas shown below

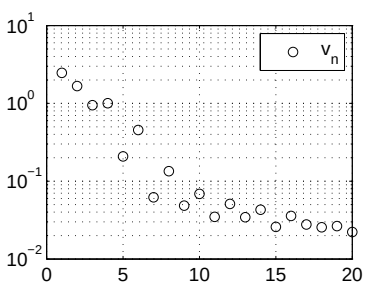



the nonzero structure of $\boldsymbol{F}$ is

$$\boldsymbol{F} \;=\; \begin{pmatrix} + & + & + & + & & & & & \\ + & + & + & + & & & & & \\ + & + & + & + & & & & & \\ & & & + & + & + & + & & \\ & & & + & + & + & + & & \\ & & & + & + & + & + & & \\ & & & & + & + & + & + & \\ & & & & + & + & + & + & \\ & & & & + & + & + & + & \\ & & & & + & + & + & + & \end{pmatrix}.$$

This structure can be exploited in the least squares solution. ∎

**Example 5.24.** We want to fit the data in Figure 5.18 for $0 \le t \le 10$, and the simplest choice is to take *equidistant knots*, $\tau_j = \frac{10j}{n}$, $j = 0, 1, \ldots, n$. We have tried this for $n = 1, \ldots, 20$, and used (5.37) to get the variance estimates shown to the left in Figure 5.22.




$$v_n \;=\; \frac{\|\boldsymbol{y} - \boldsymbol{F}\,\boldsymbol{c}^*\|^2}{m - (n{+}3)}$$

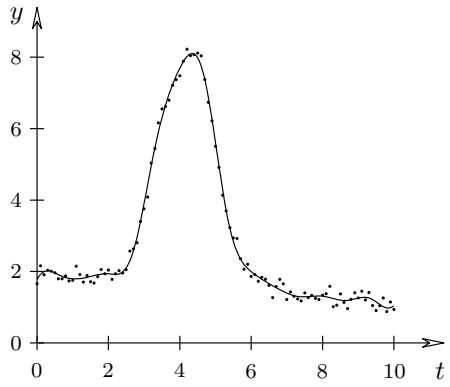



***Figure 5.22.*** *Equidistant knots spline fit to the data in Figure 5.18.*

$n = 15$ seems to be a good choice, and the corresponding fit is shown to the right. Comparing with the polynomial fit in Figure 5.20 we see some improvement, but we still miss some details close to the peak and there are undesirable oscillations at the ends. ∎

In the remainder of this chapter we shall concentrate on the choice of knots: their number and their positions. In applications $\tau_0 = a$ and $\tau_n = b$ will always be fixed, so it is the number and distribution of the interior knots that we shall discuss.

The problem can be treated as a nonlinear least squares problem:

$$\hat{\boldsymbol{x}} \;=\; \underset{\boldsymbol{x}\,\in\,\mathbb{R}^{2n+2}}{\operatorname{argmin}} \left\{ f(\boldsymbol{x}) \equiv \tfrac{1}{2} \sum_{i=1}^{m} \left( y_i - \sum_{j=1}^{n+3} c_j \, B_j(\boldsymbol{\tau}, t_i) \right)^2 \right\} \;,$$

where $\boldsymbol{x}$ is the vector

$$\boldsymbol{x} \;=\; \begin{pmatrix} \boldsymbol{\tau}_{1:n-1} \\ \boldsymbol{c} \end{pmatrix} \;.$$

This type of problem can be solved by one of the methods described in Chapter 6. Experience shows, however, that quite often the results are disappointing: either the computed $\hat{\boldsymbol{x}}$ is a local rather than global minimizer and/or the conditions in Theorem 5.53 are violated.

**Example 5.25.** Before discussing a robust algorithm for computing a good choice of knots we should mention that it often happens that the desired fit has to satisfy some boundary conditions, for instance

$$\begin{aligned} d_{1,1} s(\tau_0) + d_{1,2} s'(\tau_0) + d_{1,3} s''(\tau_0) \;&=\; d_{1,4} \;, \\ d_{2,1} s(\tau_n) + d_{2,2} s'(\tau_n) + d_{2,3} s''(\tau_n) \;&=\; d_{2,4} \;, \end{aligned} \tag{5.54}$$

where the $\{d_{k,j}\}$ are given. Inserting (5.51) and exploiting (see Appendix A.8) that at the knot $\tau_0$ the only nonzero values of the B-splines and their first two derivatives are $B^{(p)}_{1:3}(\boldsymbol{\tau}, \tau_0)$ we see that the first condition in (5.54) is equivalent to

$$\alpha_{1,1} c_1 + \alpha_{1,2} c_2 + \alpha_{1,3} c_3 \;=\; d_{1,4} \;,$$

where

$$\alpha_{1,j} \;=\; d_{1,1} B_j(\boldsymbol{\tau}, \tau_0) + d_{1,2} B'_j(\boldsymbol{\tau}, \tau_0) + d_{1,3} B''_j(\boldsymbol{\tau}, \tau_0) \;.$$

Now, assume that $\alpha_{1,1} \neq 0$. Then it follows that the spline coefficients must satisfy

$$c_1 \;=\; \gamma_1 - \beta_{1,2} c_2 - \beta_{1,3} c_3 \;, \quad \gamma_1 \;=\; \frac{d_{1,4}}{\alpha_{1,1}} \;, \quad \beta_{1,j} \;=\; \frac{\alpha_{1,j}}{\alpha_{1,1}} \;. \tag{5.55}$$

Similarly, $c_{n+3}$ can be expressed in terms of $d_{2,4}$ and $c_{n+1:n+2}$, and when we introduce these expressions in (5.52), we see that the reduced coefficient vector $\boldsymbol{z} = \boldsymbol{c}_{2:n+2}$ is the least squares solution to the modified system

$$\tilde{\boldsymbol{F}} \, \boldsymbol{z} \;\simeq\; \tilde{\boldsymbol{y}} \;, \tag{5.56}$$

where $\tilde{\boldsymbol{y}} \in \mathbb{R}^m$ and $\tilde{\boldsymbol{F}} \in \mathbb{R}^{m \times (n+1)}$ are given by

$$\begin{aligned} \tilde{\boldsymbol{y}} &:= \boldsymbol{y} - \gamma_1 \, \boldsymbol{F}_{:,1} - \gamma_2 \, \boldsymbol{F}_{:,n+3} \;, \\ \tilde{\boldsymbol{F}} &:= \boldsymbol{F}_{:,2:n+2} \;, \\ \tilde{\boldsymbol{F}}_{:,1:2} &:= \tilde{\boldsymbol{F}}_{:,1:2} - \boldsymbol{F}_{:,1} \boldsymbol{\beta}_{1,1:2} \;, \\ \tilde{\boldsymbol{F}}_{:,n:n+1} &:= \tilde{\boldsymbol{F}}_{:,n:n+1} - \boldsymbol{F}_{:,n+3} \boldsymbol{\beta}_{2,1:2} \;. \end{aligned}$$

After having computed $\hat{\boldsymbol{z}}$, the least squares solution to (5.56), the com-

plete coefficient vector is given by $\hat{\boldsymbol{c}}_{2:n+2} = \hat{\boldsymbol{z}}$, and $\hat{c}_1$ and $\hat{c}_{n+3}$ are then computed by (5.55) and the similar expression at the other end. The variance estimate is

$$v_n \;=\; \frac{\|\tilde{\boldsymbol{y}} - \tilde{\boldsymbol{F}}\,\hat{\boldsymbol{z}}\|^2}{m-(n+1)} \;.$$

The difference between the denominator here and in Figure 5.22 is caused by the conditions that reduce the number of degrees of freedoms by 2.

If we set all the $d_{i,j} = 0$ except for $d_{1,3} = d_{2,3} = 1$ in (5.54), we get a so-called *natural spline*, ie a spline that satisfies $s''(\tau_0) = s''(\tau_n) = 0$. This choice would dampen the oscillations at the ends in Figure 5.22. ∎

## 5.8. An algorithm for choice of knots

If the background function were known, the following theorem gives a hint about a good distribution of the knots:

**Theorem 5.57.** Let the function $\Upsilon$ be four times continuously differentiable in the interval $[\tau_{i-1}, \tau_i]$, and let $s$ be a cubic spline approximation. Then

$$\max_{\tau_{j-1}\le t\le\tau_j} |\Upsilon(t) - s(t)| \;\le\; \tfrac{1}{384} M_j h_j^4 + \tfrac{1}{4} E_j' h_j + E_j \;,$$

where

$$h_j = \tau_j - \tau_{j-1} \;, \qquad M_j = \max_{\tau_{j-1}\le t\le\tau_j} |\Upsilon^{(4)}(t)| \;,$$

$$E_j = \max_{i=j-1,j} |\Upsilon(\tau_i) - s(\tau_i)| \;, \quad E_j' = \max_{i=j-1,j} |\Upsilon'(\tau_i) - s'(\tau_i)| \;.$$

This theorem shows that the best we can hope for is a *local error* proportional to $h_4^4 M_j$. In other words, the knots should be close where the function $\Upsilon$ varies fast, while we can use larger knot spacing where the variation is slower.

In data fitting we only know the data points. As in (5.6) we can write

$$r_i \;=\; y_i - s(t_i) \;=\; y_i - \Upsilon(t_i) + \Upsilon(t_i) - s(t_i) \;,$$

which we recognize as the sum of data error $y_i - \Upsilon(t_i)$ and approximation error $\Upsilon(t_i) - s(t_i)$. The goal is to find $n$ and the interior knots so that the spline is sufficiently lax to follow the variations in $\Upsilon$, but not so lax that the effect of the errors $y_i - \Upsilon(t_i)$ dominate in any part of the region. Put another way: we want to find the number and the distribution of the

interior knots such that the "noise" and the approximation errors are of the same order of magnitude, both locally and globally. The algorithm that we present shares the main ideas with [47] and [11].

The basic idea is as follows: For a given $n$ and knots $\boldsymbol{\tau}_{0:n}$ compute the fitting spline and estimate *trends*, ie indication of whether approximation error dominates locally, Subsection 5.8.1. If this is the case, insert a new knot (Subsection 5.8.2), ie increase $n$ and we have a new knot set with which the process is repeated.

### 5.8.1. Trends

The trend measure (5.48) is global, ie it includes all data point. We need a measure for the local trend: Again we assume that the data points are ordered by increasing value of $t_i$. In the current knot set let $p_j$ denote the number of data points in the $j$th knot interval, $\mathcal{I}_j = [\tau_{j-1}, \tau_j]$. The *local correlation* of the residuals is defined as

$$R_j \;=\; \begin{cases} 0 & \text{if } p_j < 2\ , \\ \dfrac{1}{(p_j-1)V_j} \displaystyle\sum_{t_k, t_{k+1} \in \mathcal{I}_j} r_k r_{k+1} & \text{otherwise,} \end{cases} \tag{5.58}$$

where $V_j$ is the estimate of *local variance*,

$$V_j \;=\; \max\left\{V,\ \frac{1}{p_j} \sum_{t_k \in \mathcal{I}_j} r_k^2\right\}\ , \qquad V \;=\; \frac{1}{m-\nu} \sum_{i=1}^{m} r_i^2\ .$$

Here, $V$ is the *global variance*, and $\nu = n{+}1$ in case of fitting with two boundary conditions, cf Example 5.25, otherwise $\nu = n{+}3$.

Proceeding as we did in deriving (5.48), there is a local trend if $R_j \geq 1/\sqrt{2(p_j-1)}$. In other words, we get the following *measure of trend* in the $j$th knot interval,

$$T_j \;=\; \begin{cases} 0 & \text{if } p_j \leq 1 \\ \sqrt{2(p_j-1)}\, R_j & \text{otherwise} \end{cases}\ . \tag{5.59}$$

A value $T_j \geq 1$ indicates that locally the spline is too stiff. It may be "softened" by inserting an extra knot in the interval $[\tau_{j-1}, \tau_j]$.

### 5.8.2. Knot insertion

There exist a number of strategies for inserting extra knots. Algorithm 5.60 below describes a simple version, which often gives good results.

**Algorithm 5.60. Knot insertion**
Given data $\{t_i, y_i\}_{i=1}^m$ and range $[a, b]$ (with all $t_i \in [a, b]$).
Choose initial $n$ and knots (with $\tau_0 = a$ and $\tau_n = b$)
**repeat**
  Compute fitting spline {Section 5.7}
  **if** $V_{\text{new}} > V_{\text{old}}$
    **return** with previous knot set
  Compute trends $T_{1:n}$ by (5.59)
  $M := \text{argmax}\,\{T_j\}$
  **if** $T_M \geq 1$
    adjust knots and $n$ {See below}
**until** $T_M < 1$

Basically, the knot adjustment is to choose an interval – the $k$th (see below) – insert a new knot in the middle of it (thus increasing $n$ by one) and maybe move the "old" knots $\tau_{k-1}$ and $\tau_k$ a little, according to the algorithm

$$\textbf{if } T_{k-1} \geq 1 \textbf{ then } \tau_{k-1} := \tau_{k-1} - 0.1h_{k-1}$$
$$\textbf{if } T_{k+1} \geq 1 \textbf{ then } \tau_k := \tau_k + 0.1h_{k+1}$$

Artificial variables $T_0 = T_{n+1} = 0$ ensure that the end knots stay fixed.

The interval chosen has $T_k > 1$, and if one or both of its neighbours also show a need for "softening", then this strategy attempts to satisfy the needs also for the neighbour(s). In order to enhance this possibility we compute

$$L := \text{argmax}\,\{T_j \mid T_{j-1} \geq 1 \text{ and } T_{j+1} \geq 1\} ,$$

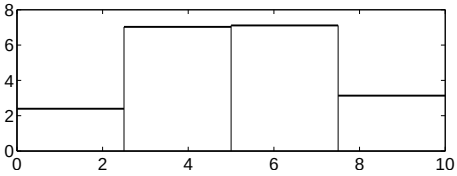
and choose

$$k = \begin{cases} L & \text{if } T_L \geq \max\left\{1, \frac{1}{2}T_M\right\} \\ M & \text{otherwise} \end{cases} .$$

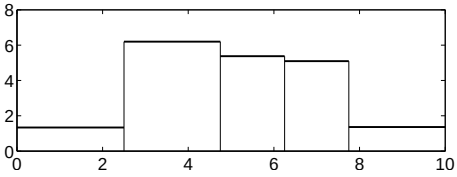
It should be noted that the trend measure (5.59) favours intervals with many data points. This enhances the probability that the reduced knot intervals still contain a fair amount of data points.

**Example 5.26.** We have used this algorithm to fit a natural spline to the data of Figure 5.18. As the starting point we chose $n = 4$ and equidistant knots. We got the following results; the plots show the trend measures.

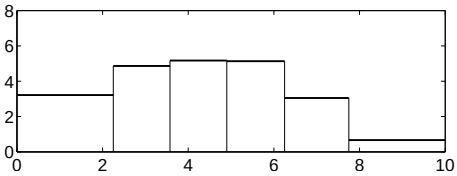
$n = 4:$ $V = 1.4003$
$M = L = k = 3.$



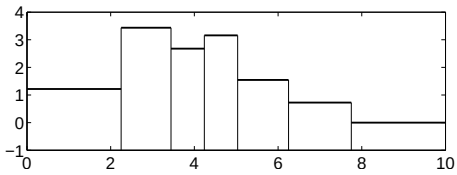
$n = 5:$ $V = 0.3326$
$M = L = k = 2.$



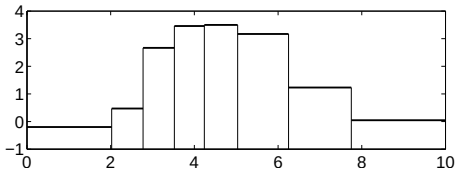
$n = 6:$ $V = 0.3306$
$M = 2,\ L = k = 3.$



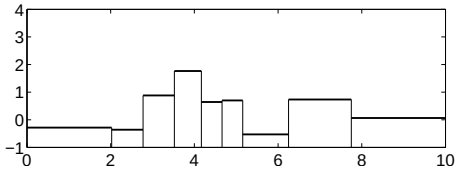
$n = 7:$ $V = 0.0779$
$M = L = k = 2.$



$n = 8:$ $V = 0.0622$
$M = 5,\ L = k = 5.$



$n = 9:$ $V = 0.0252$
$M = k = 4.$



$n = 10:$ $V = 0.0235$
$M = k = 9.$

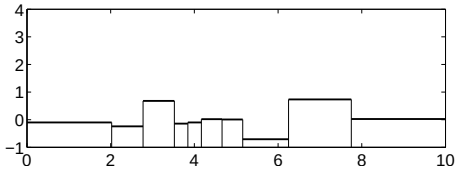


***Figure 5.23.*** *Performance of Algorithm 5.60.*

The algorithm stops because all $T_j < 1$. The final spline is shown in Figure 5.24. Comparing this with Figures 5.20 and 5.22 we see that we got rid of the oscillations at the ends and have captured details at the peak.

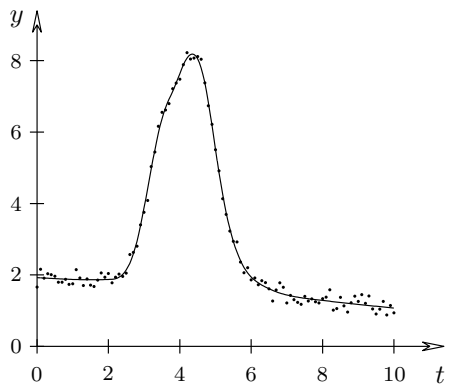



***Figure 5.24.*** *Resulting spline fit to the data in Figure 5.18.*

The data points were generated by adding noise with variance $\sigma^2 = 0.0225$ to values of the function

$$\Upsilon(t) \;=\; 2 - 0.1t + \frac{2(t-1)}{(t-4)^4 + 1} \;.$$

The final value for $V$ agrees well with $\sigma^2$. Figure 5.25 shows the residuals and the error curve $s(t) - \Upsilon(t)$. Note that the maximum error is close to the standard deviation $\sqrt{V} = 0.153$.

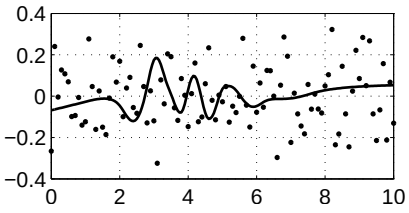



***Figure 5.25.*** *Residuals $y_i - s(t_i)$ (dots) and error $s(t) - \Upsilon(t)$.*

■

# Chapter 6

# Nonlinear Least Squares Problems

In this chapter we consider the problem:

**Definition 6.1. Least squares problem**

Let $\boldsymbol{r} : \mathbb{R}^n \mapsto \mathbb{R}^m$ be a vector function and $m \leq n$. Find $\hat{\boldsymbol{x}}$, a local minimizer for

$$f(\boldsymbol{x}) \;=\; \tfrac{1}{2}\,\|\boldsymbol{r}(\boldsymbol{x})\|^2 \;=\; \tfrac{1}{2}\,\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}) \;=\; \tfrac{1}{2}\sum_{i=1}^{m}\left(r_i(\boldsymbol{x})\right)^2 \;. \qquad (6.2)$$

Note that in this chapter $\|\cdot\|$ denotes the 2-norm. The factor $\frac{1}{2}$ in the definition of $f(\boldsymbol{x})$ has no effect on $\hat{\boldsymbol{x}}$. It is introduced for convenience, see page 115.

**Example 6.1.** An important source of least squares problems is *data fitting*, where we are given data points $(t_1, y_1), \ldots, (t_m, y_m)$ and a *fitting model* $M(\boldsymbol{x}, t)$ that depends on parameters $\boldsymbol{x} = (x_1, \ldots, x_n)^T$. The $r_i(\boldsymbol{x})$ are the *residuals*

$$r_i(\boldsymbol{x}) \;=\; y_i - M(\boldsymbol{x}, t_i)\;, \quad i = 1, \ldots, m\;,$$

and $\hat{\boldsymbol{x}}$ is the least squares solution introduced in Section 5.1. By comparison with (5.7) we see that $f(\boldsymbol{x}) = \frac{1}{2}\left(\Omega_2(\boldsymbol{I}, \boldsymbol{x})\right)^2$. In this chapter we shall discuss methods for problems where $M$ and therefore $r_i$ depends nonlinearly on $\boldsymbol{x}$. ∎

**Example 6.2.** Another application is in the solution of nonlinear systems of equations,

$$\boldsymbol{r}(\hat{\boldsymbol{x}}) = \boldsymbol{0}\;, \quad \text{where} \quad \boldsymbol{r} : \mathbb{R}^n \mapsto \mathbb{R}^n\;. \qquad (6.3)$$

This must be solved by iteration, and if we are given a good starting guess $\boldsymbol{x}_0$, and the Jacobian $\boldsymbol{J}(\hat{\boldsymbol{x}})$, see (6.5), of $\boldsymbol{r}(\hat{\boldsymbol{x}})$ is nonsingular, then *Newton–Raphson's method* is a fast and accurate method, see for instance [18, Section 4.8]. If these conditions are not satisfied, however, we are not sure that the iterations converge, and if it does, the convergence may be slow.

We can reformulate the problem in a way that enables us to use all the "tools" that we are going to present in this chapter: A solution of (6.3) is a global minimizer of the function $f$ given in Definition 6.1, since $f(\hat{\boldsymbol{x}}) = 0$ and $f(\boldsymbol{x}) > 0$ if $\boldsymbol{r}(\boldsymbol{x}) \neq \boldsymbol{0}$.

A simple approach is to combine Newton–Raphson's method with *line search*. The typical iteration step is

$$\begin{aligned} &\text{Solve } \ \boldsymbol{J}(\boldsymbol{x}_k)\,\boldsymbol{h} \ = -\boldsymbol{r}(\boldsymbol{x}_k) \quad \text{for } \ \boldsymbol{h} \ , \\ &\boldsymbol{x}_{k+1} \ = \ \boldsymbol{x}_k + \alpha_k\,\boldsymbol{h} \ , \end{aligned} \tag{6.4}$$

where $\alpha_k$ is found by line search applied to the function $\varphi(\alpha) = f(\boldsymbol{x}_k + \alpha\,\boldsymbol{h})$.

As a specific example we shall consider the following problem, taken from [48],

$$\boldsymbol{r}(\boldsymbol{x}) \ = \ \begin{pmatrix} x_1 \\ \frac{10x_1}{x_1+0.1} + 2x_2^2 \end{pmatrix} \ ,$$

with $\hat{\boldsymbol{x}} = \boldsymbol{0}$ as the only solution. The Jacobian is

$$\boldsymbol{J}(\boldsymbol{x}) \ = \ \begin{pmatrix} 1 & 0 \\ (x_1+0.1)^{-2} & 4x_2 \end{pmatrix} \ ,$$

which is singular at the solution.

If we take $\boldsymbol{x}_0 = \begin{pmatrix} 3 & 1 \end{pmatrix}^T$ and use the above algorithm with exact line search, then the iterates converge to $\boldsymbol{x}_{\mathrm{c}} \simeq \begin{pmatrix} 1.8016 & 0 \end{pmatrix}^T$, which is <u>not</u> a solution. On the other hand, if we use the simple Newton–Raphson algorithm, ie (6.4) with $\alpha_k = 1$ in every step, then it is easily seen that the iterates are $\boldsymbol{x}_k = \begin{pmatrix} 0 & y_k \end{pmatrix}^T$ with $y_{k+1} = \frac{1}{2}\,y_k$, ie we have linear convergence to the solution.

We shall return to this problem in a number of examples, to see how different methods handle it. ∎

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

In all the methods described in this chapter we assume that every $r_i(\boldsymbol{x})$ is continuously differentiable with respect to each of the indepen-

dent variables $x_j$, and define the *Jacobian* $\boldsymbol{J}(\boldsymbol{x}) \in \mathbb{R}^{m\times n}$ with elements

$$(\boldsymbol{J}(\boldsymbol{x}))_{ij} = \frac{\partial r_i}{\partial x_j}(\boldsymbol{x}) \ . \tag{6.5}$$

**Example 6.3.** The $i$th row of $\boldsymbol{J}(\boldsymbol{x})$ is the transpose of the gradient of the function $r_i \ \mathbb{R}^n \mapsto \mathbb{R}$, $\boldsymbol{J}_{i,:} = (\boldsymbol{\nabla} r_i)^T$.

In Example 5.2 we used the fitting model $M(\boldsymbol{x}, t) = x_1 e^{-x_3 t} + x_2 e^{-x_4 t}$. The $i$th row in the Jacobian is

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

If the fitting model is linear, then the residual vector has the form $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{F}\,\boldsymbol{x}$, cf (5.8), and then $\boldsymbol{J}(\boldsymbol{x}) = -\boldsymbol{F}$ for all $\boldsymbol{x}$. ∎

Provided that the $r_i$ have continuous second partial derivatives, we can write the *Taylor expansion* from $\boldsymbol{x}$ of $f$

$$f(\boldsymbol{x}+\boldsymbol{h}) \ = \ f(\boldsymbol{x}) + \boldsymbol{h}^T \boldsymbol{\nabla} f(\boldsymbol{x}) + \tfrac{1}{2}\, \boldsymbol{h}^T \boldsymbol{\nabla}^2 f(\boldsymbol{x})\, \boldsymbol{h} + O(\|\boldsymbol{h}\|^3) \ ,$$

where $\boldsymbol{\nabla} f$ and $\boldsymbol{\nabla}^2 f$ are the gradient and Hessian of $f$, respectively. From the first formulation in Definition 6.1 it follows that[1)]

$$\big(\boldsymbol{\nabla} f(\boldsymbol{x})\big)_j \ = \ \frac{\partial f}{\partial x_j}(\boldsymbol{x}) \ = \ \sum_{i=1}^{m} r_i(\boldsymbol{x}) \frac{\partial r_i}{\partial x_j}(\boldsymbol{x}) \ .$$

This shows that the *gradient* of $f$ is

$$\boldsymbol{\nabla} f(\boldsymbol{x}) \ = \ \boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x}) \ . \tag{6.6}$$

Next,

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

showing that the *Hessian* of $f$ is

$$\boldsymbol{\nabla}^2 f(\boldsymbol{x}) \ = \ \boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{J}(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x}) \boldsymbol{\nabla}^2 r_i(\boldsymbol{x}) \ . \tag{6.7}$$

[1)] If we had not used the factor $\frac{1}{2}$ in the definition of $f$, we would have got an annoying factor 2 in many expressions.

**Example 6.4.** For a linear data fitting problem $\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{y} - \boldsymbol{F}\,\boldsymbol{x}$, implying that $\boldsymbol{\nabla}^2 r_i(\boldsymbol{x}) = \boldsymbol{0}$, and by use of the result from Example 6.1 we see that in that case we get

$$\boldsymbol{\nabla} f(\boldsymbol{x}) \;=\; -\boldsymbol{F}^T(\boldsymbol{y} - \boldsymbol{F}\,\boldsymbol{x})\;, \qquad \boldsymbol{\nabla}^2 f(\boldsymbol{x}) \;=\; \boldsymbol{F}^T\boldsymbol{F}\;.$$

The gradient is zero at the minimizer, and from the above expression we see that $\boldsymbol{\nabla} f(\boldsymbol{x}) = \boldsymbol{0}$ is equivalent to the normal equations (5.15). ∎

The methods presented in this chapter are descent methods, ie they are of the form given in the generic algorithm 2.9, and they take advantage of the special form of the objective function $f$.

## 6.1. The Gauss–Newton method

This method is the basis of the very efficient methods we shall describe in the following sections. 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 Chapter 3.

The *Gauss–Newton* method is based on an affine approximation to the components of $\boldsymbol{r}$ (an *affine model* of $\boldsymbol{r}$) in the neighbourhood of $\boldsymbol{x}$: Provided that $\boldsymbol{r}$ has continuous second partial derivatives, we can write its Taylor expansion as

$$\boldsymbol{r}(\boldsymbol{x}+\boldsymbol{h}) \;=\; \boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} + O(\|\boldsymbol{h}\|^2)\;.$$

This shows that

$$\boldsymbol{r}(\boldsymbol{x}+\boldsymbol{h}) \;\simeq\; \boldsymbol{\ell}(\boldsymbol{h}) \;\equiv\; \boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} \tag{6.8}$$

is a good approximation for sufficiently small $\boldsymbol{h}$. Inserting this in the definition (6.2) of $f$ we see that

$$\begin{aligned} f(\boldsymbol{x}+\boldsymbol{h}) \;\simeq\; L(\boldsymbol{h}) \;&\equiv\; \tfrac{1}{2}\,\boldsymbol{\ell}(\boldsymbol{h})^T\boldsymbol{\ell}(\boldsymbol{h}) \\ &=\; \tfrac{1}{2}\,\boldsymbol{r}^T\boldsymbol{r} + \boldsymbol{h}^T\boldsymbol{J}^T\boldsymbol{r} + \tfrac{1}{2}\,\boldsymbol{h}^T\boldsymbol{J}^T\boldsymbol{J}\,\boldsymbol{h} \\ &=\; f(\boldsymbol{x}) + \boldsymbol{h}^T\boldsymbol{J}^T\boldsymbol{r} + \tfrac{1}{2}\,\boldsymbol{h}^T\boldsymbol{J}^T\boldsymbol{J}\,\boldsymbol{h}\;, \end{aligned} \tag{6.9}$$

where $\boldsymbol{r} = \boldsymbol{r}(\boldsymbol{x})$ and $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{x})$. The so-called *Gauss–Newton step* $\boldsymbol{h}_{\rm gn}$ minimizes $L(\boldsymbol{h})$,

$$\boldsymbol{h}_{\rm gn} \;=\; \operatorname{argmin}_{\boldsymbol{h}}\{L(\boldsymbol{h})\}\;.$$

The approximation $L$ is a *model* of the behaviour of $f$ close to the current iterand. The model is of the form treated in Theorem 1.7, and the gradient is

$$\boldsymbol{\nabla} L(\boldsymbol{h}) \;=\; \boldsymbol{J}^T\boldsymbol{r} + \big(\boldsymbol{J}^T\boldsymbol{J}\big)\,\boldsymbol{h}\;, \tag{6.10}$$

where again $\boldsymbol{r}=\boldsymbol{r}(\boldsymbol{x})$ and $\boldsymbol{J}=\boldsymbol{J}(\boldsymbol{x})$. The gradient is zero at a minimizer for $L$, so we get the condition

$$(\boldsymbol{J}^T\boldsymbol{J})\boldsymbol{h}_{\mathrm{gn}} \;=\; -\boldsymbol{J}^T\boldsymbol{r}\;. \tag{6.11}$$

Comparing this with (5.15) we see that (6.11) is the normal equations for the overdetermined, linear system

$$\boldsymbol{\ell}(\boldsymbol{h}) \;\simeq\; \boldsymbol{0} \quad\Leftrightarrow\quad \boldsymbol{J}\,\boldsymbol{h} \;\simeq\; -\boldsymbol{r}\;.$$

From the discussion in Section 5.2 it follows that if the columns in $\boldsymbol{J}$ are linearly independent, then the matrix $\boldsymbol{A}=\boldsymbol{J}^T\boldsymbol{J}$ is positive definite. This implies that $L(\boldsymbol{h})$ has a unique minimizer, which can be found by solving (6.11). This does not, however, guarantee that $\boldsymbol{x}+\boldsymbol{h}_{\mathrm{gn}}$ is a minimizer of $f$, after all $L(\boldsymbol{h})$ is only an approximation to $f(\boldsymbol{x}+\boldsymbol{h})$, but Theorem 2.15 guarantees that the solution $\boldsymbol{h}_{\mathrm{gn}}$ is a descent direction. Thus, we can use $\boldsymbol{h}_{\mathrm{gn}}$ for $\boldsymbol{h}_{\mathrm{d}}$ in Algorithm 2.9. The typical step is

$$\begin{aligned}&\text{Solve}\quad (\boldsymbol{J}^T\boldsymbol{J})\boldsymbol{h}_{\mathrm{gn}} = -\boldsymbol{J}^T\boldsymbol{r}\\ &\boldsymbol{x} := \boldsymbol{x} + \alpha\boldsymbol{h}_{\mathrm{gn}}\end{aligned}$$

where $\alpha$ 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

1° $\{\boldsymbol{x} \mid f(\boldsymbol{x}) \le f(\boldsymbol{x}_0)\}$ is bounded, and

2° the Jacobian $\boldsymbol{J}(\boldsymbol{x})$ has full rank in all steps.

Newton's method for optimization has quadratic final convergence, cf Theorem 3.4. This is normally not the case with the Gauss–Newton method. To see that, we compare the search directions $\boldsymbol{h}_{\mathrm{n}}$ and $\boldsymbol{h}_{\mathrm{gn}}$ used in the two methods,

$$\boldsymbol{\nabla}^2 f(\boldsymbol{x})\boldsymbol{h}_{\mathrm{n}} \;=\; -\boldsymbol{\nabla} f(\boldsymbol{x})\;, \qquad \boldsymbol{\nabla}^2 L(\boldsymbol{0})\boldsymbol{h}_{\mathrm{gn}} \;=\; -\boldsymbol{\nabla} L(\boldsymbol{0})\;.$$

From (6.6) and (6.10) we see that the two right-hand sides are identical,

$$\boldsymbol{\nabla} f(\boldsymbol{x}) \;=\; \boldsymbol{\nabla} L(\boldsymbol{0}) \;=\; \boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x})\;,$$

but (6.7) and (6.11) show that the coefficient matrices differ:

$$\boldsymbol{\nabla}^2 f(\boldsymbol{x}) \;=\; \boldsymbol{\nabla}^2 L(\mathbf{0}) + \sum_{i=1}^{m} r_i(\boldsymbol{x})\boldsymbol{\nabla}^2 r_i(\boldsymbol{x}) \;. \tag{6.12}$$

Therefore, if $\boldsymbol{r}(\hat{\boldsymbol{x}}) = \mathbf{0}$, then $\boldsymbol{\nabla}^2 L(\mathbf{0}) \simeq \boldsymbol{\nabla}^2 f(\boldsymbol{x})$ for $\boldsymbol{x}$ close to $\hat{\boldsymbol{x}}$, and we get quadratic convergence also with the Gauss–Newton method. We can expect superlinear convergence if the functions $\{r_i\}$ have small curvatures or if the $\{|r_i(\hat{\boldsymbol{x}})|\}$ are small, but in general we must expect linear convergence. It is remarkable that the value of $f(\hat{\boldsymbol{x}})$ controls the convergence speed.

**Example 6.5.** Consider the simple problem with $n = 1$, $m = 2$

$$\boldsymbol{r}(x) \;=\; \begin{pmatrix} x+1 \\ \lambda x^2 + x - 1 \end{pmatrix}. \quad f(x) = \tfrac{1}{2}(x+1)^2 + \tfrac{1}{2}(\lambda x^2 + x - 1)^2 \;.$$

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 $f''(0) > 0$, so $x = 0$ is a local minimizer – actually, it is the global minimizer.

The Jacobian is

$$\boldsymbol{J}(x) = \begin{pmatrix} 1 \\ 2\lambda x + 1 \end{pmatrix} \;,$$

and the classical Gauss–Newton method gives

$$x_{\text{new}} \;=\; x - \frac{2\lambda^2 x^3 + 3\lambda x^2 - 2(\lambda - 1)x}{2 + 4\lambda x + 4\lambda^2 x^2} \;.$$

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

$$x_{\text{new}} \;=\; x + (\lambda - 1)x + O(x^2) \;=\; \lambda x + O(x^2) \;.$$

Thus, if $|\lambda| < 1$, we have linear convergence. If $\lambda < -1$, then the classical Gauss–Newton method cannot find the minimizer. For instance with $\lambda = -2$ and $x^{[0]} = 0.1$ we get a seemingly chaotic behaviour of the iterates,

| $k$ | $x_k$ |
|---|---|
| 0 | 0.1000 |
| 1 | −0.3029 |
| 2 | 0.1368 |
| 3 | −0.4680 |
| ⋮ | ⋮ |

Finally, if $\lambda = 0$, then $x_{\text{new}} = x - x = 0$, ie we find the solution in one step. The reason is that in this case $\boldsymbol{r}$ is an affine function. ∎

**Example 6.6.** In the data fitting problem from Examples 5.2 and 6.3 we used the fitting model $M(\boldsymbol{x}, t) = x_1 e^{-x_3 t} + x_2 e^{-x_4 t}$. The $i$th row in the Jacobian is

$$\boldsymbol{J}(\boldsymbol{x})_{i,:} \;=\; \begin{pmatrix} -e^{-x_3 t_i} & -e^{-x_4 t i} & x_1 t_i e^{-x_3 t_i} & x_2 t_i e^{-x_4 t_i} \end{pmatrix} .$$

If the problem is *consistent* (ie $\boldsymbol{r}(\hat{\boldsymbol{x}}) = \boldsymbol{0}$), then the Gauss–Newton method with line search will have quadratic final convergence, provided that $\hat{x}_3$ is significantly different from $\hat{x}_4$. If $\hat{x}_3 = \hat{x}_4$, then $\operatorname{rank}(\boldsymbol{J}(\hat{\boldsymbol{x}})) \leq 2$, and the Gauss–Newton method fails.

If one or more measurement errors are exceptionally large, then $\boldsymbol{r}(\hat{\boldsymbol{x}})$ has some large components, and this may slow down the convergence. If, on the other hand, the errors behave like *white noise*, we can expect partial cancelling of the terms $r_i(\hat{\boldsymbol{x}})\boldsymbol{\nabla}^2 r_i(\hat{\boldsymbol{x}})$ in the difference between $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ and $\boldsymbol{\nabla}^2 L(\boldsymbol{0})$, and get superlinear final convergence.

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

```
function  [r, J] = fitexp(x, ty)
t = ty(:,1); x = x(:);
E = exp(t * (-x(3:4)'));
r = ty(:,2) - E*x(1:2);
J = [-E (t*x(1:2)').*E];
```

∎

**Example 6.7.** If we use Newton–Raphson's method to solve the problem from Example 6.2, $\boldsymbol{r}(\hat{\boldsymbol{x}}) = \boldsymbol{0}$ with $\boldsymbol{r} : \mathbb{R}^n \mapsto \mathbb{R}^n$, the typical iteration step is

$$\text{Solve} \;\; \boldsymbol{J}(\boldsymbol{x})\boldsymbol{h}_{\text{nr}} \;=\; -\boldsymbol{r}(\boldsymbol{x}); \qquad \boldsymbol{x} := \boldsymbol{x} + \boldsymbol{h}_{\text{nr}} \; .$$

The Gauss–Newton method applied to the minimization of $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{r}(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x})$ has the typical step

$$\text{Solve} \;\; \left(\boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{J}(\boldsymbol{x})\right) \boldsymbol{h}_{\text{gn}} = -\boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x}); \qquad \boldsymbol{x} := \boldsymbol{x} + \boldsymbol{h}_{\text{gn}} \; .$$

In this case $\boldsymbol{J}(\boldsymbol{x})$ is a square matrix, and if it is nonsingular, then $\left(\boldsymbol{J}(\boldsymbol{x})\right)^{-T}$ exists, and it follows that $\boldsymbol{h}_{\text{gn}} = \boldsymbol{h}_{\text{nr}}$. Therefore, when applied to Powell's problem from Example 6.2, 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 Sections 6.2 and 6.3 we give two methods with superior global performance.

## 6.2. The Levenberg–Marquardt method

Levenberg and later Marquardt, [31, 36] suggested to use a *damped Gauss–Newton method*, cf Section 3.2. The step $\boldsymbol{h}_{\rm lm}$ is defined by the following modification to (6.11),

$$\begin{aligned} \big(\boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{I}\big)\,\boldsymbol{h}_{\rm lm} \;&=\; -\boldsymbol{J}^T\boldsymbol{r} \\ \text{with}\quad \boldsymbol{J} = \boldsymbol{J}(\boldsymbol{x}),&\quad \boldsymbol{r} = \boldsymbol{r}(\boldsymbol{x}),\quad \mu \geq 0\ . \end{aligned} \tag{6.13}$$

The damping parameter $\mu$ has several effects:

1° For all $\mu > 0$ the coefficient matrix is positive definite, cf Appendix A.2, and this ensures that $\boldsymbol{h}_{\rm lm}$ is a descent direction, cf Thedorem 2.15.

2° For large values of $\mu$ we get

$$\boldsymbol{h}_{\rm lm} \;\simeq\; -\frac{1}{\mu}\,\boldsymbol{J}^T\boldsymbol{r} \;=\; -\frac{1}{\mu}\,\boldsymbol{\nabla} f(\boldsymbol{x})\ ,$$

ie a short step in the steepest descent direction. This is good if the current iterate is far from the solution.

3° If $\mu$ is very small, then $\boldsymbol{h}_{\rm lm} \simeq \boldsymbol{h}_{\rm gn}$, which is a good step in the final stages of the iteration, when $\boldsymbol{x}$ is close to $\hat{\boldsymbol{x}}$. If $f(\hat{\boldsymbol{x}}) = 0$ (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 $\mu$-value should be related to the size of the elements in $\boldsymbol{A}^{[0]} = \boldsymbol{J}(\boldsymbol{x}_0)^T\boldsymbol{J}(\boldsymbol{x}_0)$, for instance by letting

$$\mu_0 \;=\; \tau \cdot \max_i\{a_{ii}^{[0]}\}\ , \tag{6.14}$$

where $\tau$ is chosen by the user. The algorithm is not very sensitive to the choice of $\tau$, but as a rule of thumb, one should use a small value, for instance $\tau = 10^{-6}$ if $\boldsymbol{x}_0$ is believed to be a good approximation to $\hat{\boldsymbol{x}}$. Otherwise, use $\tau = 10^{-3}$ or even $\tau = 1$. During iteration the size of $\mu$ can be updated as described in Section 3.2. The updating is controlled by the *gain ratio*

$$\varrho \;=\; \frac{f(\boldsymbol{x}) - f(\boldsymbol{x}_{\rm new})}{L(\boldsymbol{0}) - L(\boldsymbol{h}_{\rm lm})}\ ,$$

where $\boldsymbol{x}_{\rm new} = \boldsymbol{x} + \boldsymbol{h}_{\rm lm}$. In order to reduce *cancellation error* the numerator

$$\delta f \;=\; f(\boldsymbol{x}) - f(\boldsymbol{x}_{\rm new}) \;=\; \tfrac{1}{2}\,\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}) - \tfrac{1}{2}\,\boldsymbol{r}(\boldsymbol{x}_{\rm new})^T\boldsymbol{r}(\boldsymbol{x}_{\rm new})$$

should be computed by the mathematically equivalent expression

$$\delta f \;=\; \tfrac{1}{2}\left(\boldsymbol{r}(\boldsymbol{x}) - \boldsymbol{r}(\boldsymbol{x}_{\rm new})\right)^T \left(\boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{r}(\boldsymbol{x}_{\rm new})\right) \;. \tag{6.15}$$

The denominator is the gain predicted by the affine model (6.9),

$$\begin{aligned}
\delta L &= L(\boldsymbol{0}) - L(\boldsymbol{h}_{\rm lm}) \\
&= -\boldsymbol{h}_{\rm lm}^T \boldsymbol{J}^T \boldsymbol{r} - \tfrac{1}{2}\,\boldsymbol{h}_{\rm lm}\,\boldsymbol{J}^T \boldsymbol{J}\,\boldsymbol{h}_{\rm lm} \\
&= -\tfrac{1}{2}\,\boldsymbol{h}_{\rm lm}^T \left(2\boldsymbol{J}^T \boldsymbol{r} + (\boldsymbol{J}^T \boldsymbol{J} + \mu \boldsymbol{I} - \mu \boldsymbol{I})\boldsymbol{h}_{\rm lm}\right) \\
&= \tfrac{1}{2}\,\boldsymbol{h}_{\rm lm}^T \left(\mu\,\boldsymbol{h}_{\rm lm} - \boldsymbol{\nabla} f)\right) \quad .
\end{aligned} \tag{6.16}$$

Note that both $\boldsymbol{h}_{\rm lm}^T \boldsymbol{h}_{\rm lm}$ and $-\boldsymbol{h}_{\rm lm}^T \boldsymbol{\nabla} f$ are positive, so $L(\boldsymbol{0}) - L(\boldsymbol{h}_{\rm lm})$ is guaranteed to be positive.

A large value of $\varrho$ indicates that $L(\boldsymbol{h}_{\rm lm})$ is a good approximation to $f(\boldsymbol{x}+\boldsymbol{h}_{\rm lm})$, and we can decrease $\mu$ so that the next Levenberg-Marquardt step is closer to the Gauss–Newton step. If $\varrho$ is small (maybe even negative), then $L(\boldsymbol{h}_{\rm lm})$ is a poor approximation, and we should increase $\mu$ 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, see Algorithm 6.18 and Example 6.7 below.

As discussed in connection with (2.10) – (2.11) we suggest to use the following *stopping criteria* for the algorithm,

$$\begin{aligned}
&\|\boldsymbol{\nabla} f(\boldsymbol{x})\|_\infty \;\leq\; \varepsilon_1 \;, \\
&\|\boldsymbol{x}_{\rm new} - \boldsymbol{x}\|_2 \;\leq\; \varepsilon_2(\|\boldsymbol{x}\|_2 + \varepsilon_2) \;, \\
&k \;\geq\; k_{\max} \;,
\end{aligned} \tag{6.17}$$

where $\varepsilon_1$ are small, positive numbers and $k_{\max}$ is an integer; all of these are chosen by the user.

The last two criteria come into effect, for instance, if $\varepsilon_1$ 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 affine model $L$, and will result in $\mu$ being augmented in every step. The updating strategy for $\mu$ is based on (3.12), but modified so that consecutive failures of getting a smaller $f$-value gives fast growth of $\mu$, resulting in small $\|\boldsymbol{h}_{\rm lm}\|$, and the process will be stopped by the second criterion in (6.17).

The algorithm is summarized below. We shall refer to it as the *L–M method*.

**Algorithm 6.18. Levenberg–Marquardt method**

Given $\boldsymbol{x}_0$, $\tau$, $\varepsilon_1$, $\varepsilon_2$, $k_{\max}$

**begin**
  $k := 0$; $\nu := 2$; $\boldsymbol{x} := \boldsymbol{x}_0$
  $\boldsymbol{A} := \boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{J}(\boldsymbol{x})$; $\boldsymbol{g} := \boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x})$
  *found* := $(\|\boldsymbol{g}\|_\infty \le \varepsilon_1)$; $\mu := \tau * \max\{a_{ii}\}$
  **while** (**not** *found*) **and** $(k < k_{\max})$
    $k := k+1$; Solve $(\boldsymbol{A} + \mu\boldsymbol{I})\boldsymbol{h}_{\text{lm}} = -\boldsymbol{g}$
    **if** $\|\boldsymbol{h}_{\text{lm}}\| \le \varepsilon_2(\|\boldsymbol{x}\| + \varepsilon_2)$
      *found* := **true**
    **else**
      $\boldsymbol{x}_{\text{new}} := \boldsymbol{x} + \boldsymbol{h}_{\text{lm}}$
      $\varrho := \delta f/\delta L$ {computed by (6.15) and (6.16)}
      **if** $\varrho > 0$ {step acceptable}
        $\boldsymbol{x} := \boldsymbol{x}_{\text{new}}$
        $\boldsymbol{A} := \boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{J}(\boldsymbol{x})$; $\boldsymbol{g} := \boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x})$
        *found* := $(\|\boldsymbol{g}\|_\infty \le \varepsilon_1)$
        $\mu := \mu * \max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}$; $\nu := 2$
      **else**
        $\mu := \mu * \nu$; $\nu := 2 * \nu$
      **end**
    **end**
  **end**
**end**

**Example 6.8.** We already saw in connection with (6.11) that the Gauss–Newton step $\boldsymbol{h}_{\text{gn}}$ is the least squares solution to the linear problem

$$\boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} \simeq \boldsymbol{0} .$$

Similarly, the L-M equations in (6.13) are the normal equations for the augmented linear problem

$$\begin{pmatrix} \boldsymbol{r}(\boldsymbol{x}) \\ \boldsymbol{0} \end{pmatrix} + \begin{pmatrix} \boldsymbol{J}(\boldsymbol{x}) \\ \sqrt{\mu}\boldsymbol{I} \end{pmatrix} \boldsymbol{h} \simeq \boldsymbol{0} .$$

As discussed in Section 5.2, the most accurate solution is found via orthogonal transformation. However, the solution $\boldsymbol{h}_{\text{lm}}$ 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" to compute, this method is often used. ∎

**Example 6.9.** The Rosenbrock problem from Examples 2.8, 3.5 and 3.9 can be formulated as a system of nonlinear equations,

$$\boldsymbol{r}(\boldsymbol{x}) \;=\; \sqrt{2}\cdot\begin{pmatrix} 10\cdot\left(x_2 - x_1^2\right) \\ 1 - x_1 \end{pmatrix} \;.$$

It is easy to see that this system has the unique solution $\hat{\boldsymbol{x}} = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$, and this is the unique minimizer for $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{x})\|^2$.

In Figure 6.1 we illustrate the performance of Algorithm 6.18 applied to this problem with the input $\boldsymbol{x}_0 = \begin{pmatrix} -1.2 & 1 \end{pmatrix}^T$, $\tau = 10^{-3}$, $\varepsilon_1 = 10^{-8}$, $\varepsilon_2 = 10^{-12}$ $k_{max} = 100$.

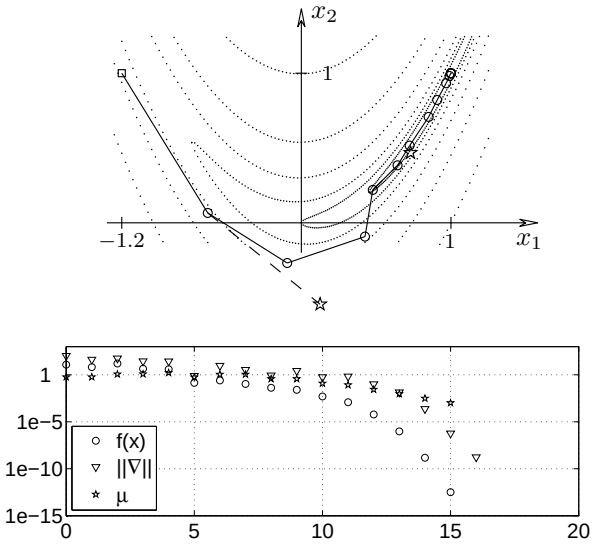



***Figure 6.1.*** *Rosenbrock function. Iteration path and performance parameters.*

The step from $\boldsymbol{x}_0$ is good, but from $\boldsymbol{x}_1$ the attempted step was uphill, so a new step has to be computed with an increased value for $\mu$. This also happens at $\boldsymbol{x}_5$. Apart from that the iteration progresses nicely, and stops after 15 computations of $\boldsymbol{h}_{\mathrm{lm}}$ because it has reached a point $\boldsymbol{x} = \hat{\boldsymbol{x}} - 10^{-9}\cdot\begin{pmatrix} 4.1 & 8.2 \end{pmatrix}^T$ with $\|\boldsymbol{\nabla} f(\boldsymbol{x})\|_\infty \simeq 1.7{\cdot}10^{-9} \leq \varepsilon_1$.

Rosenbrock's problem is consistent, $\boldsymbol{r}(\hat{\boldsymbol{x}}) = \boldsymbol{0}$, and therefore we can expect quadratic final convergence. This is seen by the behavior of $f(\boldsymbol{x})$ and $\|\boldsymbol{\nabla} f(\boldsymbol{x})\|$ in the last few iterations. ∎

**Example 6.10.** We have used Algorithm 6.18 on the data fitting problem from Examples 5.2 and 6.6 with the fitting model

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

Figure 5.2 indicates that both $x_3$ and $x_4$ are positive and that $M(\hat{\boldsymbol{x}}, 0) \simeq 0$. These conditions are satisfied by $\boldsymbol{x}_0 = \begin{pmatrix} 1 & -1 & 1 & 2 \end{pmatrix}^T$. Further, we used

$\tau = 10^{-3},\ \varepsilon_1 = 10^{-8},\ \varepsilon_2 = 10^{-14}\ \ k_{max} = 100$. The algorithm stopped after 62 iteration steps with $\boldsymbol{x} \simeq \begin{pmatrix} 4 & -4 & 4 & 5 \end{pmatrix}^T$, corresponding to the fit shown by full line in Figure 5.2 The performance is illustrated below.

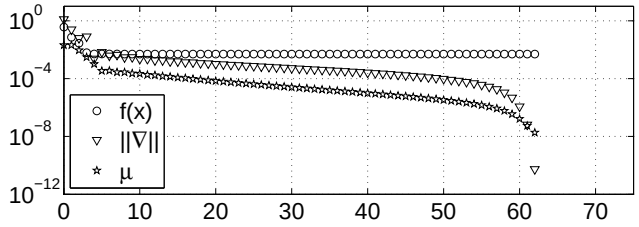



***Figure 6.2.*** *L-M method applied to the problem from Example 5.2.*

This problem is not consistent, so we could expect linear final convergence. The final iterations indicate a much better (superlinear) convergence. The explanation is that the $\boldsymbol{\nabla}^2 r_i(\boldsymbol{x})$ are slowly varying functions of $t_i$, and the $r_i(\hat{\boldsymbol{x}})$ have "random" sign, so that the contributions to the "forgotten term" in (6.12) almost cancel each other. Such a situation occurs frequently in data fitting applications.

For comparison, Figure 6.3 shows the performance with the classical updating strategy for $\mu$, (3.11). From step 5 to step 68 we see that each decrease in $\mu$ 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 6.2.

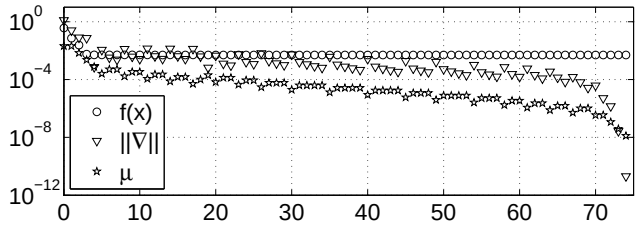



***Figure 6.3.*** *L-M method with classical $\mu$-updating strategy.* ∎

**Example 6.11.** Figure 6.4 illustrates the performance of Algorithm 6.18 applied to Powell's problem from Examples 6.2 and 6.7. The starting point is $\boldsymbol{x}_0 = \begin{pmatrix} 3 & 1 \end{pmatrix}^T$, and we use $\tau = 1,\ \varepsilon_1 = \varepsilon_2 = 10^{-15},\ \ k_{max} = 100$.

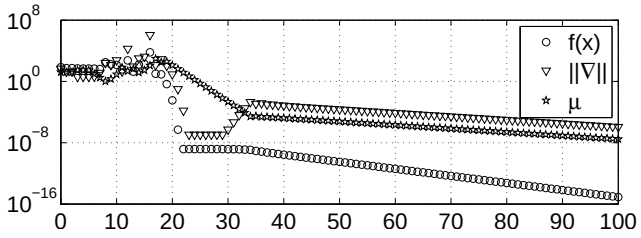



***Figure 6.4.*** *The L-M method applied to Powell's problem.*

The iteration process seems to stall between steps 22 and 30. This is an effect of the (almost) singular Jacobian matrix. After that there seems to be linear convergence. The iteration is stopped by the "safeguard" at the point $\boldsymbol{x} = \begin{pmatrix} \texttt{-3.82e-08} & \texttt{-1.38e-03} \end{pmatrix}^T$. This is a better approximation to $\hat{\boldsymbol{x}} = \boldsymbol{0}$ than we found in Example 6.7, but we want to be able to do even better; see Example 6.10. ∎

## 6.3. Powell's Dog Leg method

As the Levenberg–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. Powell's name is connected to the algorithm because he proposed how to find an approximation to $\boldsymbol{h}_{\text{tr}}$, defined by (2.28):

$$\boldsymbol{h}_{\text{tr}} \;=\; \underset{\boldsymbol{h}\,\in\,\mathcal{T}}{\text{argmin}}\{q(\boldsymbol{h})\}\;; \quad \mathcal{T} \;=\; \{\boldsymbol{h} \mid \|\boldsymbol{h}\| \le \Delta\}, \quad \Delta > 0\;,$$

where $q(\boldsymbol{h})$ is an approximation to $f(\boldsymbol{x}+\boldsymbol{h})$.

Given $\boldsymbol{r} : \mathbb{R}^n \mapsto \mathbb{R}^m$. At the current iterate $\boldsymbol{x}$ the Gauss–Newton step $\boldsymbol{h}_{\text{gn}}$ is a least squares solution to the linear system

$$\boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} \;\simeq\; -\boldsymbol{r}(\boldsymbol{x})\;.$$

The steepest descent direction is given by

$$\boldsymbol{h}_{\text{sd}} \;=\; -\boldsymbol{g} \;=\; -\boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x})\;.$$

This is a direction, not a step, and to see how far we should go, we look at the affine model

$$\begin{aligned} \boldsymbol{r}(\boldsymbol{x}+\alpha\boldsymbol{h}_{\text{sd}}) \;&\simeq\; \boldsymbol{r}(\boldsymbol{x}) + \alpha\boldsymbol{J}(\boldsymbol{x})\boldsymbol{h}_{\text{sd}} \\ \Downarrow \\ f(\boldsymbol{x}+\alpha\boldsymbol{h}_{\text{sd}}) \;&\simeq\; \tfrac{1}{2}\|\boldsymbol{r}(\boldsymbol{x}) + \alpha\boldsymbol{J}(\boldsymbol{x})\boldsymbol{h}_{\text{sd}}\|^2 \\ &=\; f(\boldsymbol{x}) + \alpha\boldsymbol{h}_{\text{sd}}^T\,\boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}) + \tfrac{1}{2}\alpha^2\|\boldsymbol{J}(\boldsymbol{x})\boldsymbol{h}_{\text{sd}}\|^2\;. \end{aligned}$$

This function of $\alpha$ is minimal for

$$\alpha \;=\; -\,\frac{\boldsymbol{h}_{\rm sd}^T\,\boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x})}{\|\boldsymbol{J}(\boldsymbol{x})\boldsymbol{h}_{\rm sd}\|^2} \;=\; \frac{\|\boldsymbol{g}\|^2}{\|\boldsymbol{J}(\boldsymbol{x})\boldsymbol{g}\|^2}\;. \tag{6.19}$$

Now we have two candidates for the step to take from the current point $\boldsymbol{x}$: $\boldsymbol{a}=\alpha\boldsymbol{h}_{\rm sd}$ and $\boldsymbol{b}=\boldsymbol{h}_{\rm gn}$. Powell suggested to use the following strategy for choosing the step, when the trust region has radius $\Delta$. The last case in the strategy is illustrated in Figure 6.5.

$$\begin{aligned}
&\textbf{if } \|\boldsymbol{h}_{\rm gn}\| \le \Delta \\
&\quad \boldsymbol{h}_{\rm dl} := \boldsymbol{h}_{\rm gn} \\
&\textbf{elseif } \|\alpha\boldsymbol{h}_{\rm sd}\| \ge \Delta \\
&\quad \boldsymbol{h}_{\rm dl} := (\Delta/\|\boldsymbol{h}_{\rm sd}\|)\boldsymbol{h}_{\rm sd} \\
&\textbf{else} \\
&\quad \boldsymbol{h}_{\rm dl} := \alpha\boldsymbol{h}_{\rm sd} + \beta(\boldsymbol{h}_{\rm gn} - \alpha\boldsymbol{h}_{\rm sd}) \\
&\qquad \text{with } \beta \text{ chosen so that } \|\boldsymbol{h}_{\rm dl}\| = \Delta\;.
\end{aligned} \tag{6.20}$$

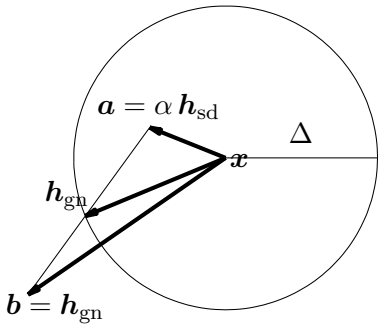



***Figure 6.5.*** *Trust region and Dog Leg step.*[2)]

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

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

We seek a root for this second degree polynomial, and note that $P\to+\infty$ for $\beta\to-\infty$; $P(0) = \|\boldsymbol{a}\|^2-\Delta^2 < 0$; $P(1) = \|\boldsymbol{h}_{\rm gn}\|^2-\Delta^2 > 0$. Thus, $P$ has one negative root and one root in $]0,1[$. We seek the latter, and the

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

most accurate computation of it is given by

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

As in the Levenberg-Marquardt method we can use the gain ratio

$$\varrho = \left( f(\boldsymbol{x}) - f(\boldsymbol{x}+\boldsymbol{h}_{\mathrm{dl}}) \right) / \left( L(\boldsymbol{0}) - L(\boldsymbol{h}_{\mathrm{dl}}) \right)$$

to monitor the iteration. Again, $L$ is the linear model

$$L(\boldsymbol{h}) = \tfrac{1}{2} \|\boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\boldsymbol{h}\|^2 \ .$$

In the L-M method we used $\varrho$ to control the size of the damping parameter. Here, we use it to control the radius $\Delta$ of the trust region. A large value of $\varrho$ indicates that the linear model is good. We can increase $\Delta$ and thereby take longer steps, and they will be closer to the Gauss–Newton direction. If $\varrho$ is small, maybe even negative, then we reduce $\Delta$, implying smaller steps, closer to the steepest descent direction. Below we summarize the algorithm.

We have the following remarks.

1° Initialization. $\boldsymbol{x}_0$ and $\Delta_0$ should be supplied by the user.

2° We use the stopping criteria (6.17) supplemented with $\|\boldsymbol{r}(\boldsymbol{x})\|_\infty \le \varepsilon_3$, reflecting that $\boldsymbol{r}(\hat{\boldsymbol{x}}) = \boldsymbol{0}$ in case of $m = n$, ie a nonlinear system of equations.

3° See Example 6.12 below.

4° Corresponding to the three cases in (6.20) we can show that

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

5° The strategy in Algorithm 2.30 is used to update the trust region radius.

**Algorithm 6.21. Dog Leg method**

Given $\boldsymbol{x}_0$, $\Delta_0$, $\varepsilon_1$, $\varepsilon_2$, $\varepsilon_3$, $k_{\max}$ 1°

**begin**

$k := 0$; $\boldsymbol{x} := \boldsymbol{x}_0$; $\Delta := \Delta_0$; $\boldsymbol{g} := \boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x})$

*found* := $(\|\boldsymbol{r}(\boldsymbol{x})\|_\infty \le \varepsilon_3)$ **or** $(\|\boldsymbol{g}\|_\infty \le \varepsilon_1)$ 2°

**while** (**not** *found*) **and** $(k < k_{\max})$

$k := k+1$; Compute $\alpha$ by (6.19)

$\boldsymbol{h}_{\mathrm{sd}} := -\alpha \boldsymbol{g}$; Solve $\boldsymbol{J}(\boldsymbol{x})\boldsymbol{h}_{\mathrm{gn}} \simeq -\boldsymbol{r}(\boldsymbol{x})$ 3°

Compute $\boldsymbol{h}_{\mathrm{dl}}$ by (6.20)

**if** $\|\boldsymbol{h}_{\mathrm{dl}}\| \le \varepsilon_2(\|\boldsymbol{x}\| + \varepsilon_2)$

*found* := **true**

**else**

$\boldsymbol{x}_{\mathrm{new}} := \boldsymbol{x} + \boldsymbol{h}_{\mathrm{dl}}$

$\varrho := (f(\boldsymbol{x}) - f(\boldsymbol{x}_{\mathrm{new}}))/(L(\boldsymbol{0}) - L(\boldsymbol{h}_{\mathrm{dl}}))$ 4°

**if** $\varrho > 0$

$\boldsymbol{x} := \boldsymbol{x}_{\mathrm{new}}$; $\boldsymbol{g} := \boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x})$

*found* := $(\|\boldsymbol{r}(\boldsymbol{x})\|_\infty \le \varepsilon_3)$ **or** $(\|\boldsymbol{g}\|_\infty \le \varepsilon_1)$

**if** $\varrho > 0.75$ 5°

$\Delta := \max\{\Delta, 3*\|\boldsymbol{h}_{\mathrm{dl}}\|\}$

**elseif** $\varrho < 0.25$

$\Delta := \Delta/2$; *found* := $(\Delta \le \varepsilon_2(\|\boldsymbol{x}\| + \varepsilon_2))$ 6°

**end**

6° Extra stopping criterion. If $\Delta \le \varepsilon_2(\|\boldsymbol{x}\| + \varepsilon_2)$, then (6.17) will surely be satisfied in the next step.

**Example 6.12.** In Example 6.8 we briefly discussed the computation of the step $\boldsymbol{h}_{\mathrm{lm}}$ and argued that we might as well compute it via the normal equations formulation. Provided that $\mu$ is not very small, the matrix is reasonably well conditioned, and there will be no excessive effects of rounding errors.

The Dog Leg method is intended perform well also on nonlinear systems of equations, ie where the system $\boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} \simeq -\boldsymbol{r}(\boldsymbol{x})$ is a square system of linear equations

$$\boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} \;=\; -\boldsymbol{r}(\boldsymbol{x}) \;,$$

with the solution $\boldsymbol{h} = \boldsymbol{h}_{\mathrm{NR}}$, the Newton–Raphson step, cf Example 6.2. The Jacobian $\boldsymbol{J}$ may be ill-conditioned (even singular), in which case rounding errors tend to dominate the solution. This problem is worsened if we use the formulation (6.11) to compute $\boldsymbol{h}_{\mathrm{gn}}$.

In the implementation `dogleg` in `immoptibox` the vector $\boldsymbol{h}_{\text{gn}}$ is computed with respect to these problems. If the columns of $\boldsymbol{J}(\boldsymbol{x})$ are not significantly linearly independent, then the least squares solution $\boldsymbol{h}$ is not unique, and $\boldsymbol{h}_{\text{gn}}$ is computed as the $\boldsymbol{h}$ with minimum norm. Some details of this computation are given in Appendix A.6. ∎

**Example 6.13.** Figure 6.6 illustrates the performance of Algorithm 6.21 applied to Powell's problem from Examples 6.2, 6.7 and 6.11. The starting point is $\boldsymbol{x}_0 = \begin{pmatrix} 3 & 1 \end{pmatrix}^T$, and we use $\Delta_0 = 1$, and the stopping criteria given by $\varepsilon_1 = \varepsilon_2 = 10^{-15}$, $\varepsilon_3 = 10^{-20}$, $k_{\max} = 100$.

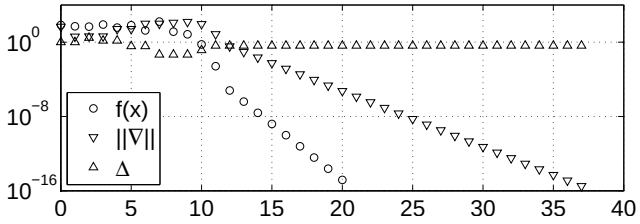



***Figure 6.6.*** *The Dog Leg method applied to Powell's problem.*

The iteration stopped after 37 steps because of a small gradient, and returned $\boldsymbol{x} = \begin{pmatrix} 3.72{\cdot}10^{-34} & 1.26{\cdot}10^{-9} \end{pmatrix}^T$, which is quite a good approximation to $\hat{\boldsymbol{x}} = \boldsymbol{0}$. As in Figure 6.4 we see that the ultimate convergence is linear (caused by the singular $\boldsymbol{J}(\hat{\boldsymbol{x}})$), but considerably faster than with the Levenberg-Marquardt method. ∎

**Example 6.14.** We have used Algorithm 6.21 on the data fitting problem presented in Example 5.2. As in Example 6.10 we use the starting point $\boldsymbol{x}_0 = \begin{pmatrix} -1, & -2, & 1, & -1 \end{pmatrix}^T$, and take $\Delta_0 = 1$ and the stopping criteria given by $\varepsilon_1 = 10^{-8}$, $\varepsilon_2 = \varepsilon_3 = 10^{-12}$, $k_{\max} = 100$. The algorithm stopped after 30 iterations with $\boldsymbol{x} \simeq \begin{pmatrix} -4, & -5, & 4, & -4 \end{pmatrix}^T$. The performance is illustrated below. As in Figure 6.2 we note a very fast ultimate rate of convergence.

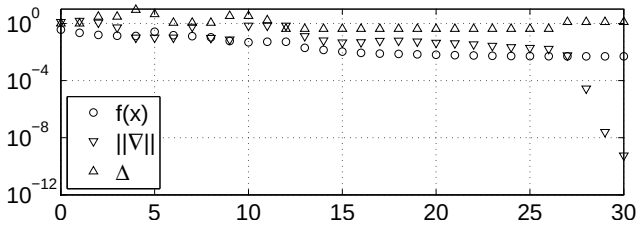



***Figure 6.7.*** *Dog Leg method applied to the problem from Example 5.2.*

∎

The last two examples seem to indicate that the Dog Leg method is considerably better than the Levenberg-Marquardt method. This is true when the least squares problem arises from a system of nonlinear equations. The Dog Leg method is presently considered as the best method for solving systems of nonlinear equations.

For general least squares problems the Dog Leg method has the same disadvantages as the L-M method: the final convergence can be expected to be linear (and slow) if $f(\hat{\boldsymbol{x}}) \neq 0$. For a given problem and given starting guess $\boldsymbol{x}_0$ it is not possible to say beforehand which of the two methods will be the faster.

## 6.4. A secant version of the L–M method

The methods discussed in this chapter assume that the vector function $\boldsymbol{r}$ is differentiable, ie the Jacobian

$$\boldsymbol{J}(\boldsymbol{x}) = \left( \frac{\partial r_i}{\partial x_j} \right)$$

exists. In many practical optimization problems it happens that we cannot give formulae for the elements in $\boldsymbol{J}$, for instance because $\boldsymbol{r}$ is given by a "black box". The secant version of the L–M method is intended for problems of this type.

The simplest remedy is to replace $\boldsymbol{J}(\boldsymbol{x})$ by a matrix $\boldsymbol{G}$ obtained by *numerical differentiation*: The $(i,j)^{\text{th}}$ element is approximated by the finite difference approximation, cf Appendix A.3,

$$\frac{\partial r_i}{\partial x_j}(\boldsymbol{x}) \;\simeq\; g_{ij} \;\equiv\; \frac{r_i(\boldsymbol{x}+\delta \boldsymbol{e}_{(j)}) - r_i(\boldsymbol{x})}{\delta} \quad , \tag{6.22}$$

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

**Example 6.15.** Let $m=n=1$ and consider one nonlinear equation

$$f : \mathbb{R} \mapsto \mathbb{R}. \qquad \text{Find } \hat{x} \text{ such that } f(\hat{x}) = 0 \ .$$

For this problem we can write the Newton–Raphson algorithm (6.4) in

the form

$$\begin{aligned} & f(x{+}h) \;\simeq\; \ell(h) \;\equiv\; f(x) + f'(x)h \\ & \text{Solve the linear problem } \;\ell(h) = 0 \\ & x_{\text{new}} := x + h \end{aligned} \tag{6.23}$$

If we cannot implement $f'(x)$, then we can approximate it by

$$(f(x{+}\delta) - f(x))/\delta$$

with $\delta$ chosen appropriately small. More generally, we can replace (6.23) by

$$\begin{aligned} & f(x{+}h) \;\simeq\; \lambda(h) \;\equiv\; f(x) + g\,h \quad \text{with} \;\; g \simeq f'(x) \\ & \text{Solve the linear problem } \;\lambda(h) = 0 \\ & x_{\text{new}} := x + h \end{aligned} \tag{6.24}$$

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

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

This gives

$$g \;=\; (f(x) - f(x_{\text{prev}})) \,/\, (x - x_{\text{prev}}) \;,$$

and with this choice of $g$ we recognize (6.24) as the *secant method*, see for instance [18, pp 70f]. 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. ∎

Now, consider the linear model (6.8) for $\boldsymbol{r} : \mathbb{R}^n \mapsto \mathbb{R}^m$,

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

We will replace it by

$$\boldsymbol{r}(\boldsymbol{x}{+}\boldsymbol{h}) \;\simeq\; \boldsymbol{\lambda}(\boldsymbol{h}) \;\equiv\; \boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{G}\boldsymbol{h} \;,$$

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

$$\boldsymbol{r}(\boldsymbol{x}_{\text{new}}{+}\boldsymbol{h}) \;\simeq\; \boldsymbol{r}(\boldsymbol{x}_{\text{new}}) + \boldsymbol{G}_{\text{new}}\boldsymbol{h} \;.$$

Especially, we want this model to hold with equality for $\boldsymbol{h} = \boldsymbol{x}{-}\boldsymbol{x}_{\text{new}}$, ie

$$\boldsymbol{r}(\boldsymbol{x}) \;=\; \boldsymbol{r}(\boldsymbol{x}_{\text{new}}) + \boldsymbol{G}_{\text{new}}(\boldsymbol{x}{-}\boldsymbol{x}_{\text{new}}) \;. \tag{6.25}$$

This gives us $m$ equations in the $m{\cdot}n$ unknown elements of $\boldsymbol{G}_{\text{new}}$, so we need more conditions. Broyden, [6], suggested to supplement (6.25) with

$$\boldsymbol{G}_{\text{new}}\boldsymbol{v} \;=\; \boldsymbol{G}\,\boldsymbol{v} \quad \text{for all} \;\; \boldsymbol{v} \perp (\boldsymbol{x}{-}\boldsymbol{x}_{\text{new}}) \;. \tag{6.26}$$

It is easy to verify that conditions (6.25)–(6.26) are satisfied by the *updating formula* discussed in Example 3.6:

**Definition 6.27. Broyden's rank one update**

$$\boldsymbol{G}_{\text{new}} \;=\; \boldsymbol{G} + \boldsymbol{u}\,\boldsymbol{h}^T \;,$$

where

$$\boldsymbol{h} = \boldsymbol{x}_{\text{new}} - \boldsymbol{x} \;, \quad \boldsymbol{u} = \frac{1}{\boldsymbol{h}^T\boldsymbol{h}}\left(\boldsymbol{r}(\boldsymbol{x}_{\text{new}}) - \boldsymbol{r}(\boldsymbol{x}) - \boldsymbol{G}\boldsymbol{h}\right) \;.$$

Comparing with Example 6.15 we see that condition (6.25) corresponds to the secant condition in the case $n=1$. We say that this approach is a *generalized secant method*.

A brief sketch of the central part of Algorithm 6.18 with this modification has the form

Solve $(\boldsymbol{G}^T\boldsymbol{G} + \mu\boldsymbol{I})\boldsymbol{h}_{\text{slm}} = -\boldsymbol{G}^T\boldsymbol{r}(\boldsymbol{x})$
$\boldsymbol{x}_{\text{new}} := \boldsymbol{x} + \boldsymbol{h}_{\text{slm}}$
Update $\boldsymbol{G}$ by Definition 6.27
Update $\mu$ and $\boldsymbol{x}$ as in Algorithm 6.18

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

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 iterations the current $\boldsymbol{G}$ is such a poor approximation to the true Jacobian, that $-\boldsymbol{G}^T\boldsymbol{r}(\boldsymbol{x})$ is not even a downhill direction. In that case $\boldsymbol{x}$ will stay unchanged and $\mu$ is increased. The approximation $\boldsymbol{G}$ is changed, but may still be a poor approximation, leading to a further increase in $\mu$, etc. Eventually the process is stopped by $\boldsymbol{h}_{\text{slm}}$ being so small that the second criterion in (6.17) is satisfied, although $\boldsymbol{x}$ may be far from $\hat{\boldsymbol{x}}$.

A number of strategies have been proposed to overcome this problem, for instance to make occasional recomputations of $\boldsymbol{G}$ by finite differences. In Algorithm 6.29 below we supplement the updatings determined by Definition 6.27 with a cyclic, coordinate-wise series of updatings: Let $\boldsymbol{h}$ denote the current step, and let $j$ be the current coordinate number. If the pseudo angle $\theta$ between $\boldsymbol{h}$ and $\boldsymbol{e}_{(j)}$ is "large", then we compute a

finite difference approximation to the $j^{\text{th}}$ column of $\boldsymbol{J}$. More specific, this is done if

$$\cos\theta \;=\; \frac{|\boldsymbol{h}^T\boldsymbol{e}_{(j)}|}{\|\boldsymbol{h}\|\cdot\|\boldsymbol{e}_{(j)}\|} \;<\; \gamma \quad\Leftrightarrow\quad |h_j| \;<\; \gamma\|\boldsymbol{h}\| \;. \tag{6.28}$$

Experiments indicated that the (rather pessimistic) choice $\gamma = 0.8$ gave good performance. With this choice we can expect that each iteration step needs (almost) two evaluations of the vector function $\boldsymbol{r}$.

Now we are ready to present the algorithm:

**Algorithm 6.29. Secant version of the L–M method**

**begin**
  $k := 0$; $\boldsymbol{x} := \boldsymbol{x}_0$; $\boldsymbol{G} := \boldsymbol{G}_0$; $j := 0$; $\mu := \mu_0$ {1°}
  $\boldsymbol{g} := \boldsymbol{G}^T\boldsymbol{r}(\boldsymbol{x})$; *found* = ( $\|\boldsymbol{g}\|_\infty \le \varepsilon_1$ )
  **while** (**not** *found*) **and** ($k < k_{\max}$)
    $k := k{+}1$; Solve $(\boldsymbol{G}^T\boldsymbol{G} + \mu\boldsymbol{I})\boldsymbol{h} = -\boldsymbol{g}$
    **if** $\|\boldsymbol{h}\| \le \varepsilon_2(\|x\| + \varepsilon_2)$
      *found* := **true**
    **else**
      $j := \text{mod}(j, n){+}1$; **if** $|h_j| < 0.8\|\boldsymbol{h}\|$ {2°}
        Update $\boldsymbol{G}$ by (6.22), using $\boldsymbol{x}_{\text{new}} = \boldsymbol{x} + \eta\,\boldsymbol{e}_{(j)}$ {3°}
      $\boldsymbol{x}_{\text{new}} := \boldsymbol{x} + \boldsymbol{h}$; Update $\boldsymbol{G}$ by 6.27
      Update $\mu$ as in Algorithm 6.18
      **if** $f(\boldsymbol{x}_{\text{new}}) < f(\boldsymbol{x})$
        $\boldsymbol{x} := \boldsymbol{x}_{\text{new}}$
      $\boldsymbol{g} := \boldsymbol{G}^T\boldsymbol{r}(\boldsymbol{x})$; *found* := ( $\|\boldsymbol{g}\|_\infty \le \varepsilon_1$ ) {4°}
**end**

Comments:

1° Initialization. $\mu_0$ is computed by (6.13). $\boldsymbol{x}_0$, $\tau$, parameters in the stopping criteria (6.17) and the relative step $\delta$ (see 3° below) to use in (6.22) are input, and $\boldsymbol{G}_0$ is either input or it is computed by (6.22).

2° Cf (6.28). $\text{mod}(j, n)$ is the remainder after division by $n$.

3° The step $\eta$ is given by
  **if** $x_j = 0$ **then** $\eta := \delta^2$ **else** $\eta := \delta|x_j|$ .

4° Whereas the iterate $\boldsymbol{x}$ is updated only if the descending condition is satisfied, the approximation $\boldsymbol{G}$ is updated in every step. Therefore the approximate gradient $\boldsymbol{g}$ may change also when $\boldsymbol{r}(\boldsymbol{x})$ is unchanged.

**Example 6.16.** We have applied Algorithm 6.29 to the Rosenbrock problem from Example 6.9. If we use the same starting point and stopping criteria as in that example, and take $\delta = 10^{-7}$ in the difference approximation (6.22), we find the solution after 29 iteration steps, involving a total of 53 evaluations of $\boldsymbol{r}(\boldsymbol{x})$. For comparison, the "true" L–M algorithm needs only 17 steps, implying a total of 18 evaluations of $\boldsymbol{r}(\boldsymbol{x})$ and $\boldsymbol{J}(\boldsymbol{x})$.

We have also used the secant algorithm on the data fitting problem from 6.10. With $\delta = 10^{-7}$ and the same starting point and stopping criteria as in Example 6.10 the secant version needed 145 iterations, involving a total of 295 evaluations of $\boldsymbol{r}(\boldsymbol{x})$. For comparison, Algorithm 6.18 needs 62 iterations.

These two problems indicate that Algorithm 6.29 is robust, but they also illustrate a general rule of thumb: If gradient information is available, it normally pays to use it. ∎

In many applications the numbers $m$ and $n$ are large, but each of the functions $f_i(\boldsymbol{x})$ depends only on a few of the elements in $\boldsymbol{x}$. In such cases most of the $\frac{\partial f_i}{\partial x_j}(\boldsymbol{x})$ are zero, and we say that $\boldsymbol{J}(\boldsymbol{x})$ is a *sparse matrix*. There are efficient methods exploiting sparsity in the solution of the equation $(\boldsymbol{G}^T\boldsymbol{G} + \mu\boldsymbol{I})\boldsymbol{h} = -\boldsymbol{g}$, see for instance [17]. In the updating formula in Definition 6.27, however, normally all elements in the vectors $\boldsymbol{h}$ and $\boldsymbol{u}$ are nonzero, so that $\boldsymbol{G}_{\text{new}}$ will be a *dense matrix*. It is outside the scope of this book to discuss how to cope with this; we refer to [21], [51] and [44, Chapter 9].

## 6.5. A secant version of the Dog Leg method

The idea of using a secant approximation to the Jacobian can, of course, also be used in connection with the Dog Leg method from Section 6.3. In this section we shall consider the special case of $m = n$, ie in the solution of nonlinear systems of equations. Broyden, [6], not only gave the formula from Definition 6.27 for updating the approximate Jacobian. He also gave a formula for updating an approximate inverse of the Jacobian,

$\boldsymbol{K} \simeq \boldsymbol{J}(\boldsymbol{x})^{-1}$. The two formulas are

$$\begin{aligned} \boldsymbol{G}_{\text{new}} &= \boldsymbol{G} + \frac{1}{\boldsymbol{h}^T\boldsymbol{h}}(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{h})\boldsymbol{h}^T \ , \\ \boldsymbol{K}_{\text{new}} &= \boldsymbol{K} + \frac{1}{\boldsymbol{v}^T\boldsymbol{y}}(\boldsymbol{h} - \boldsymbol{K}\,\boldsymbol{y})\boldsymbol{v}^T \ , \end{aligned} \tag{6.30}$$

where

$$\boldsymbol{h} = \boldsymbol{x}_{\text{new}} - \boldsymbol{x} \ , \quad \boldsymbol{y} = \boldsymbol{r}(\boldsymbol{x}_{\text{new}}) - \boldsymbol{r}(\boldsymbol{x}) \ , \quad \boldsymbol{v} = \boldsymbol{K}\,\boldsymbol{h} \ .$$

With these matrices the steepest descent direction $\boldsymbol{h}_{\text{sd}}$ and the Gauss–Newton step $\boldsymbol{h}_{\text{gn}}$ (which is identical to the Newton step in this case, cf Example 6.7) are approximated by

$$\boldsymbol{h}_{\text{ssd}} = -\boldsymbol{G}^T\boldsymbol{r}(\boldsymbol{x}) \quad \text{and} \quad \boldsymbol{h}_{\text{sgn}} = -\boldsymbol{K}\boldsymbol{r}(\boldsymbol{x}) \ . \tag{6.31}$$

Algorithm 6.21 is easily modified to use these approximations. The initial $\boldsymbol{G} = \boldsymbol{G}_0$ can be found by the difference approximation (6.22), and $\boldsymbol{K}_0$ can be computed as $\boldsymbol{G}_0^{-1}$. It is easy to verify that then the current $\boldsymbol{G}$ and $\boldsymbol{K}$ satisfy $\boldsymbol{G}\boldsymbol{K} = \boldsymbol{I}$.

Like the secant version of the L–M method, this method needs extra updates to keep $\boldsymbol{G}$ and $\boldsymbol{K}$ as good approximations to the current Jacobian and its inverse. We have found that the strategy discussed around (6.28) also works well in this case. It should also be mentioned that the denominator in (3.22) may be zero or very small. If

$$|\boldsymbol{h}^T\boldsymbol{D}\boldsymbol{y}| < \sqrt{\varepsilon_{\text{M}}}\,\|\boldsymbol{h}\| \ ,$$

then $\boldsymbol{K}$ is not updated, but computed as $\boldsymbol{K} = \boldsymbol{G}^{-1}$.

Each update with (3.22) "costs" $10n^2$ flops[3)] and the computation of the two step vectors by (6.31) plus the computation of $\alpha$ by (6.19) costs $6n^2$ flops. Thus, each iteration step with the gradient–free version of the Dog Leg method costs about $16n^2$ flops plus evaluation of $\boldsymbol{r}(\boldsymbol{x}_{\text{new}})$. For comparison, each step with Algorithm 6.21 costs about $\frac{2}{3}n^3+6n^2$ flops plus evaluation of $\boldsymbol{r}(\boldsymbol{x}_{\text{new}})$ and $\boldsymbol{J}(\boldsymbol{x}_{\text{new}})$. Thus, for large values of $n$ the gradient-free version is cheaper per step. However, the number of iteration steps is often considerably larger, and if the Jacobian matrix is available, then the gradient version is normally faster.

**Example 6.17.** We have used Algorithm 6.21 and the gradient–free Dog Leg method on *Rosenbrock's function* $\boldsymbol{r} : \mathbb{R}^2 \mapsto \mathbb{R}^2$, given by

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

$$\boldsymbol{r}(\boldsymbol{x}) = \begin{pmatrix} 10(x_2 - x_1^2) \\ 1 - x_1 \end{pmatrix} ,$$

cf Example 6.9. The function has one root, $\hat{\boldsymbol{x}} = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$, and with both methods we used the starting point $\boldsymbol{x}_0 = \begin{pmatrix} -1.2 & 1 \end{pmatrix}^T$ and $\varepsilon_1 = 10^{-10}$, $\varepsilon_2 = 10^{-14}$, $k_{\max} = 100$ in the stopping criteria (6.17), and $\delta = 10^{-7}$ in (6.22). Algorithm 6.21 stopped at the solution after 17 iterations, ie after 18 evaluations of $\boldsymbol{r}$ and its Jacobian. The secant version also stopped at the solution; this needed 28 iterations and a total of 49 evaluations of $\boldsymbol{r}$. ∎

## 6.6. Final remarks

We have discussed a number of algorithms for solving nonlinear least squares problems. It should also 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 6.18. Weighted least squares.** First, we want to point out that we have presented methods for unweighted least squares problems. As discussed in Sections 5.1 and 5.4, it may be appropriate to look for $\hat{\boldsymbol{x}}_{\mathbf{w}}$, a local minimizer for

$$f(\boldsymbol{x}) \;=\; \tfrac{1}{2} \left\| \boldsymbol{W}\boldsymbol{r}(\boldsymbol{x}) \right\|^2 \;=\; \tfrac{1}{2} \sum_{i=1}^{m} \left( w_i r_i(\boldsymbol{x}) \right)^2 , \tag{6.32}$$

for given weights $\{w_i\}$. Many implementations of the algorithms, for instance the MATLAB functions in immoptibox, do not explicitly allow for weights. This, however, is no restriction: The program calls a user-supplied function, which for a given $\boldsymbol{x}$ should return $\boldsymbol{r}(\boldsymbol{x})$ and maybe also the Jacobian $\boldsymbol{J}(\boldsymbol{x})$. For a weighted problem you simply let your function return $\boldsymbol{W}\boldsymbol{r}(\boldsymbol{x})$ and $\boldsymbol{W}\boldsymbol{J}(\boldsymbol{x})$. ∎

**Example 6.19. Change of variables.** In Powell's problem from Examples 6.2, 6.7, 6.11 and 6.13 the variable $x_2$ occurs only as $x_2^2$. We can change the variables to $\boldsymbol{z} = \begin{pmatrix} x_1 & x_2^2 \end{pmatrix}^T$, and the problem takes the form: Find $\boldsymbol{z}^* \in \mathbb{R}^2$ such that $\boldsymbol{r}(\boldsymbol{z}^*) = \boldsymbol{0}$, where

$$\boldsymbol{r}(\boldsymbol{z}) = \begin{pmatrix} z_1 \\ \frac{10 z_1}{z_1 + 0.1} + 2 z_2 \end{pmatrix} \qquad \text{with} \qquad \boldsymbol{J}(\boldsymbol{z}) = \begin{pmatrix} 1 & 0 \\ (z_1 + 0.1)^{-2} & 2 \end{pmatrix} .$$

This Jacobian is nonsingular for all $\boldsymbol{z}$. The Levenberg–Marquardt algo-

rithm 6.18 with starting point $\boldsymbol{z}_0 = \begin{pmatrix} 3 & 1 \end{pmatrix}^T$, $\tau = 10^{-16}$ and $\varepsilon_1 = 10^{-12}$, $\varepsilon_2 = 10^{-16}$ in the stopping criteria (6.17) stops after 4 iterationss with $\boldsymbol{z} \simeq \begin{pmatrix} \texttt{2.84e-21} & \texttt{-1.42e-19} \end{pmatrix}^T$. This is a good approximation to $\boldsymbol{z}^* = \boldsymbol{0}$. ∎

**Example 6.20. Exponential fit.** In Examples 6.3, 6.6, 6.10 and 6.14 we demonstrated the performance of some algorithms when applied to the data fitting problem from Example 5.2, with the fitting model (slightly reformulated from the previous examples)

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

The parameters $x_3$ and $x_4$ appear linearly, and we can reformulate the model to have only two parameters,

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

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

$$\boldsymbol{F}\,\boldsymbol{c} \;\simeq\; \boldsymbol{y} \;,$$

with $\boldsymbol{F} = \boldsymbol{F}(\boldsymbol{x}) \in \mathbb{R}^{m \times 2}$ given by the rows $(\boldsymbol{F})_{i,:} = (e^{x_1 t_i} \;\; e^{x_2 t_i})$. The associated residual vector is

$$\boldsymbol{r}(\boldsymbol{x}) \;=\; \boldsymbol{y} - \boldsymbol{F}(\boldsymbol{x})\,\boldsymbol{c}(\boldsymbol{x}) \;.$$

It can be shown that the Jacobian is

$$\boldsymbol{J} \;=\; -\boldsymbol{F}\boldsymbol{G} - \boldsymbol{H}[\boldsymbol{c}] \;,$$

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

$$\boldsymbol{H} \;=\; [\boldsymbol{t}]\,\boldsymbol{F}, \qquad \boldsymbol{G} = \left(\boldsymbol{F}^T\boldsymbol{F}\right)^{-1}\left([\boldsymbol{H}^T\boldsymbol{r}] - \boldsymbol{H}^T\boldsymbol{F}\,[\boldsymbol{c}]\right) \;.$$

Algorithm 6.18 with the same poor starting guess as in Example 6.10, $\boldsymbol{x}_0 = \begin{pmatrix} -1 & -2 \end{pmatrix}^T$, $\tau = 10^{-3}$ and $\varepsilon_1 = \varepsilon_2 = 10^{-8}$ finds the solution $\boldsymbol{x} \simeq \begin{pmatrix} -4 & -5 \end{pmatrix}^T$ after 13 iteration steps; about $\frac{1}{5}$ of the number of steps needed with the 4-parameter model.

This approach can be generalized to any model, where some of the parameters occur linearly. It has the name *separable least squares*, and is discussed for instance in [22] and [41]. ∎

**Example 6.21. Scaling.** This example illustrates a frequent difficulty with least squares problems: Normally the algorithms work best when the problem is scaled so that all the (nonzero) components of $\hat{\boldsymbol{x}}$ are of the same order of magnitude.

Consider the so-called *Meyer's problem*

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

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 $\hat{\boldsymbol{x}} \simeq \begin{pmatrix} 5.61{\cdot}10^{-3} & 6.18{\cdot}10^{3} & 3.45{\cdot}10^{2} \end{pmatrix}^T$ with $f(\hat{\boldsymbol{x}}) \simeq 43.97$.

An alternative formulation is

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

with $u_i = 0.45{+}0.05i$. The reformulation corresponds to

$$\boldsymbol{z} = \begin{pmatrix} 10^{-3} e^{13} x_1 & 10^{-3} x_2 & 10^{-2} x_3 \end{pmatrix}^T \;,$$

and the minimizer is

$$\boldsymbol{z}^* \simeq \begin{pmatrix} 2.48 & 6.18 & 3.45 \end{pmatrix}^T$$

with $\frac{1}{2} \, \|\boldsymbol{\rho}(\boldsymbol{z}^*)\|_2^2 \simeq 4.397{\cdot}10^{-5}$.

If we use Algorithm 6.18 with $\tau = 1$, $\varepsilon_1 = 10^{-6}$, $\varepsilon_2 = 10^{-10}$ and the equivalent starting vectors

$$\boldsymbol{x}_0 = \begin{pmatrix} 2{\cdot}10^{-2} & 4{\cdot}10^{3} & 2.5{\cdot}10^{2} \end{pmatrix}^T , \qquad \boldsymbol{z}_0 = \begin{pmatrix} 8.85 & 4 & 2.5 \end{pmatrix}^T \;,$$

then we need 175 iterations with the first formulation, and 88 iterations with the well-scaled reformulation. ∎

**Example 6.22. Multiexponential fit.** We wish to fit the data in Figure 5.9 (repeated in Figure 6.9) with the model

$$M(\boldsymbol{c}, \boldsymbol{z}, t) \;=\; \sum_{j=1}^{p} c_j \, e^{z_j t} \;. \tag{6.33}$$

This means that the parameter vector

$$\boldsymbol{x} \;=\; \begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{c} \end{pmatrix}$$

has $n = 2p$ elements. This is a *data representation* problem, and similar to the discussion with polynomials and splines, Sections 5.6 – 5.8, we have no prior knowledge of the number of terms that we can use without dominating effect of "noise".

In Example 5.14 we saw that it was relevant to use a weighted fit, and that the appropriate weights were given by

$$w_i \;=\; \left(19.7 + 1490e^{-1.89t_i}\right)^{-1} \;.$$

Figure 6.8 shows the weighted residuals for the weighted least squares fits with one and two terms in (6.33).

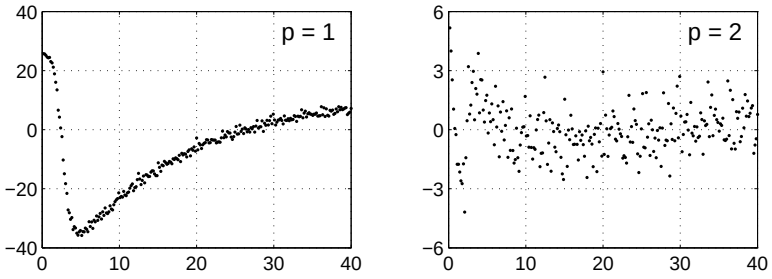



***Figure 6.8.*** *Weighted residuals for $p=1$ and $p=2$.*

We see a very clear trend for $p=1$, and a less pronounced trend for $p=1$. For $p=3$ (not shown) we cannot see a trend, so $p=3 \sim n=6$ seems a good choice.

As in Section 5.6 we can quantify the trend. Now, however, we should replace $r_i$ by $w_i r_i$, so that (5.48) is modified to

$$T_n \;=\; \sqrt{2(m-1)}\, R_n \;,$$

where

$$v_n \;=\; \frac{1}{m-n}\sum_{i=1}^{m} \left(w_i \hat{r}_i\right)^2 \;, \quad R_n \;=\; \frac{1}{(m-1)v_n}\sum_{i=1}^{m-1} w_i \hat{r}_i w_{i+1} \hat{r}_{i+1} \;.$$

We got the following results

| $p$ | $n$ | $v_n$ | $T_n$ |
|---|---|---|---|
| 1 | 2 | 249 | 22.0 |
| 2 | 4 | 1.70 | 6.62 |
| 3 | 6 | 1.04 | −1.54 |
| 4 | 8 | 1.03 | −2.09 |

We see significant decreases from $v_2$ to $v_4$ and from $v_4$ to $v_6$, while the decrease from $v_6$ to $v_8$ is insignificant. So the variance estimates indicate that we should use $p=3 \sim n=6$. This is corroborated by the trend measure: $T_6$ is the first $T_n$ smaller than one.

The resulting fitting model is

$$M(\hat{\boldsymbol{c}}_{\mathbf{w}}, \hat{\boldsymbol{z}}_{\mathbf{w}}, t)) \;=\; 99.3e^{-0.00611t} + 7310e^{-0.983t} + 32700e^{-2.05t} \;.$$

Its is shown in Figure 6.9.

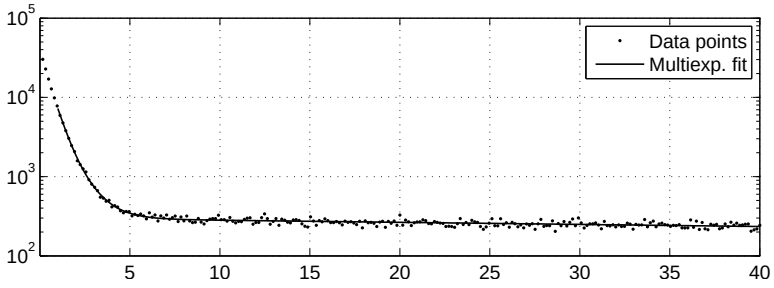



***Figure 6.9.*** *Fit to OSL data with 3 terms in* (6.33).

■

# Chapter 7

# Fitting in other Norms

In this chapter we discuss methods for parameter estimation in cases, where the definition of "best" is different from the (weighted) least squares, cf Section 5.1. More precisely, we are given a vector function $\boldsymbol{r} : \mathbb{R}^n \mapsto \mathbb{R}^m$ and we want to find $\hat{\boldsymbol{x}}$ that minimizes some measure of $\boldsymbol{r}(\boldsymbol{x})$, for instance $\|\boldsymbol{r}(\boldsymbol{x})\|_1$ or $\|\boldsymbol{r}(\boldsymbol{x})\|_\infty$. These two cases are treated in the first two sections, and in the rest of the chapter we discuss a method which can be thought of as a hybrid between the least squares and the least absolute deviation definitions of "best".

The function $\boldsymbol{r}$ may depend nonlinearly on $\boldsymbol{x}$, and we present algorithms of Gauss–Newton type, cf Section 6.1, based on successive approximations by first order Taylor expansions,

$$\begin{aligned} & r_i(\boldsymbol{x}+\boldsymbol{h}) \;\simeq\; \ell_i(\boldsymbol{h}) \;\equiv\; r_i(\boldsymbol{x}) + \boldsymbol{\nabla} r_i(\boldsymbol{x})^T \boldsymbol{h} \;, \quad i = 1, \ldots, m \;, \\ \Updownarrow & \\ & \boldsymbol{r}(\boldsymbol{x}+\boldsymbol{h}) \;\simeq\; \boldsymbol{\ell}(\boldsymbol{h}) \;\equiv\; \boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} \;, \end{aligned} \tag{7.1}$$

where $\boldsymbol{\nabla} r_i \in \mathbb{R}^n$ is the gradient of $r_i$ and $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ is the Jacobian of $\boldsymbol{r}$. It has the rows $\boldsymbol{J}_{i:} = (\boldsymbol{\nabla} r_i)^T$. In the first two sections we combine the Gauss–Newton method with a trust region approach, cf Section 2.4. The generic algorithm is

**Algorithm 7.2. Generic algorithm**

Given starting point $\boldsymbol{x}$ and trust region radius $\Delta$
**while not** STOP
  $\hat{\boldsymbol{h}} = \operatorname{argmin}_{\|\boldsymbol{h}\|_\infty \le \Delta} L(\boldsymbol{h})$
  $\varrho = \big(f(\boldsymbol{x}) - f(\boldsymbol{x}+\hat{\boldsymbol{h}})\big) / \big(L(\boldsymbol{0}) - L(\hat{\boldsymbol{h}})\big)$
  **if** $\varrho \ge 0.75$ **then** $\Delta := \Delta * 2$
  **if** $\varrho \le 0.25$ **then** $\Delta := \Delta / 3$
  **if** $\varrho > 0$ **then** $\boldsymbol{x} := \boldsymbol{x} + \hat{\boldsymbol{h}}$
**end**

The functions $f$ and $L$ are defined by $f(\boldsymbol{x}) = \|\boldsymbol{r}(\boldsymbol{x})\|_p$ and $L(\boldsymbol{h}) = \|\boldsymbol{\ell}(\boldsymbol{h})\|_p$, respectively. Combining these definitions with (7.1) we see that $L(\boldsymbol{0}) = f(\boldsymbol{x})$.

## 7.1. Fitting in the $L_\infty$ norm

We seek a minimizer $\hat{\boldsymbol{x}}$ of the function

$$f(\boldsymbol{x}) \;=\; \|\boldsymbol{r}(\boldsymbol{x})\|_\infty \;=\; \max_i |r_i(\boldsymbol{x})| \;.$$

Hald and Madsen [25] proposed to use Algorithm 7.2. The linearized problem

$$\hat{\boldsymbol{h}} \;=\; \underset{\|\boldsymbol{h}\|_\infty \le \Delta}{\operatorname{argmin}} \; \left\{ \, L(\boldsymbol{h}) \equiv \|\boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\,\boldsymbol{h}\|_\infty \, \right\}$$

is solved by an LP algorithm, cf Appendix A.9. To get an LP formulation of the problem, we introduce the variable $h_{n+1} = L(\boldsymbol{h})$ and the extended vector

$$\tilde{\boldsymbol{h}} \;=\; \begin{pmatrix} \boldsymbol{h} \\ h_{n+1} \end{pmatrix} \in \mathbb{R}^{n+1} \;.$$

Now $\hat{\boldsymbol{h}}$ can be found by solving the problem

$$\begin{array}{ll} \text{minimize} & h_{n+1} \\ \text{subject to} & \left.\begin{array}{c} -\Delta \le h_i \le \Delta \\ -h_{n+1} \le r_i(\boldsymbol{x}) + \boldsymbol{\nabla} r_i(\boldsymbol{x})^T \boldsymbol{h} \le h_{n+1} \end{array}\right\} \quad i = 1, \ldots, n \end{array}$$

See [25] for further information. This simple linear programming formulation is enabled by the change of the trust region definition from the 2–norm discussed in Section 2.4 to the $\infty$–norm used here.

## 7.2. Fitting in the $\mathbf{L}_1$ norm

We seek a minimizer $\hat{\boldsymbol{x}}$ of the function

$$f(\boldsymbol{x}) \;=\; \|\boldsymbol{r}(\boldsymbol{x})\|_1 \;=\; \sum_{i=1}^{m} |r_i(\boldsymbol{x})| \;.$$

In some applications the solution is referred to as the *least absolute deviation* fit.

Hald and Madsen [26] proposed to use Algorithm 7.2. The linearized problem

$$\hat{\boldsymbol{h}} \;=\; \underset{\|\boldsymbol{h}\|_\infty \le \Delta}{\operatorname{argmin}} \; \left\{ L(\boldsymbol{h}) \equiv \|\boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\,\boldsymbol{h}\|_1 \right\}$$

is solved by an LP algorithm: We introduce auxiliary variables $h_{n+i}$, $i = 1, \ldots, m$, and the extended vector

$$\tilde{\boldsymbol{h}} = \begin{pmatrix} \boldsymbol{h} \\ h_{n+1} \\ \vdots \\ h_{n+m} \end{pmatrix} \in \mathbb{R}^{n+m} \; .$$

Now $\hat{\boldsymbol{h}}$ can be found by solving the problem

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{m} h_{n+i} \\ &\text{subject to} \quad \left.\begin{aligned} -\Delta \le h_i &\le \Delta \\ -h_{n+i} \le r_i(\boldsymbol{x}) + \boldsymbol{\nabla} r_i(\boldsymbol{x})^T \boldsymbol{h} &\le h_{n+i} \end{aligned}\right\} \quad i = 1, \ldots, m \end{aligned}$$

See [26] for further information.

**Example 7.1.** When $\boldsymbol{r}$ is an affine function, $\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{y} - \boldsymbol{F}\boldsymbol{x}$, the gradients are constant, and the LP formulation is

$$\begin{aligned} &\text{minimize} \quad \textstyle\sum_{i=1}^{m} x_{n+i} \\ &\text{subject to} \quad -x_{n+i} \le y_i - \boldsymbol{F}_{i,:}\,\boldsymbol{x} \le x_{n+i} \; , \quad i = 1, \ldots, m \; . \end{aligned}$$

This formulation is the background for the *Barrodale–Roberts* algorithm [2], which is often used for robust parameter estimation with linear fitting models, cf Examples 5.5 and 7.3. ∎

## 7.3. Huber estimation

This approach combines the smoothness of the least squares estimator with the robustness of the $L_1$-estimator, cf Example 5.5.

For a function $\boldsymbol{r} : \mathbb{R}^n \mapsto \mathbb{R}^m$ the *Huber estimator* $\boldsymbol{x}_\gamma$ is defined by[1)]

$$\boldsymbol{x}_\gamma \;=\; \underset{\boldsymbol{x}}{\operatorname{argmin}}\big\{ f_\gamma(\boldsymbol{x}) \;\equiv\; \sum_{i=1}^{m} \phi_\gamma(r_i(\boldsymbol{x}))\big\} \;, \tag{7.3}$$

where $\phi_\gamma$ is the *Huber function*,

$$\phi_\gamma(u) \;=\; \begin{cases} \dfrac{1}{2\gamma} u^2 & \text{if } |u| \le \gamma \;, \\ |u| - \frac{1}{2}\,\gamma & \text{if } |u| > \gamma \;. \end{cases} \tag{7.4}$$

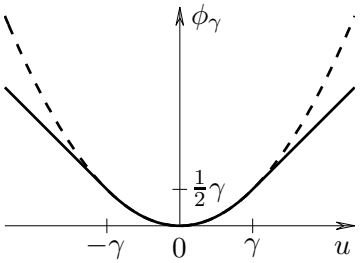



***Figure 7.1.*** *Huber function (full line) and the scaled* $\mathrm{L}_2$ *function* $\frac{1}{2\gamma}u^2$ *(dotted line).*

The threshold $\gamma$ is used to distinguish between "small" and "large" function values (residuals). Based on the values of the $r_i(\boldsymbol{x})$ we define a generalized sign vector $\boldsymbol{s} = \boldsymbol{s}_\gamma(\boldsymbol{x})$ and an "activity matrix" $\boldsymbol{W} = \boldsymbol{W}_\gamma(\boldsymbol{x}) = \operatorname{diag}(w_1, \ldots, w_m)$. Note that $w_i = 1 - s_i^2$.

| | $r_i(\boldsymbol{x}) < -\gamma$ | $\lvert r_i(\boldsymbol{x})\rvert \le \gamma$ | $r_i(\boldsymbol{x}) > \gamma$ |
|---|---|---|---|
| $s_i(\boldsymbol{x})$ | $-1$ | 0 | 1 |
| $w_i(\boldsymbol{x})$ | 0 | 1 | 0 |

(7.5)

Now the objective function in (7.3) can be expressed as

$$f_\gamma(\boldsymbol{x}) \;=\; \frac{1}{2\gamma}\, \boldsymbol{r}^T \boldsymbol{W}\, \boldsymbol{r} + \boldsymbol{r}^T \boldsymbol{s} - \tfrac{1}{2}\,\gamma\, \boldsymbol{s}^T \boldsymbol{s} \;, \tag{7.6}$$

where we have omitted the argument $(\boldsymbol{x})$ and index $\gamma$ on the right-hand side. The gradient is

$$\boldsymbol{\nabla} f_\gamma \;=\; \frac{1}{\gamma}\, \boldsymbol{J}^T (\boldsymbol{W}\, \boldsymbol{r} + \boldsymbol{\gamma}\, \boldsymbol{s}) \;. \tag{7.7}$$

[1)] Strictly speaking, it is misleading to discuss this approach under the heading "other norms": $f_\gamma$ in (7.3) is <u>not</u> a norm: the triangle inequality is not satisfied.

At a minimizer the gradient must be zero. Thus, a necessary condition for $\boldsymbol{x}_\gamma$ being a minimizer of $f_\gamma$ is that it satisfies the equation

$$\boldsymbol{J}(\boldsymbol{x})^T \left(\boldsymbol{W}_\gamma(\boldsymbol{x})\boldsymbol{r}(\boldsymbol{x}) + \gamma\, \boldsymbol{s}_\gamma(\boldsymbol{x})\right) \;=\; \boldsymbol{0}\;. \tag{7.8}$$

### 7.3.1. Linear Huber estimation

First consider an affine function

$$\boldsymbol{r}(\boldsymbol{x}) \;=\; \boldsymbol{y} - \boldsymbol{F}\,\boldsymbol{x}\;, \tag{7.9}$$

where $\boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{F} \in \mathbb{R}^{m\times n}$ are constant. It follows from (7.5) that $s_i(\boldsymbol{x}) = 0$ for all $\boldsymbol{x}$ between the two hyperplanes $r_i(\boldsymbol{x}) = -\gamma$ and $r_i(\boldsymbol{x}) = \gamma$. These $2m$ hyperplanes divide $R^n$ into subregions $\{D_k\}$. Inside each subregion $\boldsymbol{s}_\gamma(\boldsymbol{x})$ and $\boldsymbol{W}_\gamma(\boldsymbol{x})$ are constant, so $f_\gamma$ is a *piecewise quadratic*. The gradient is

$$\boldsymbol{\nabla} f_\gamma(\boldsymbol{x}) \;=\; -\frac{1}{\gamma}\,\boldsymbol{F}^T\big(\boldsymbol{W}(\boldsymbol{x})(\boldsymbol{y} - \boldsymbol{F}\,\boldsymbol{x}) + \gamma\,\boldsymbol{s}(\boldsymbol{x})\big)\;. \tag{7.10}$$

It varies continuously across the hyperplanes, while the Hessian

$$\boldsymbol{\nabla}^2 f_\gamma(\boldsymbol{x}) \;=\; \frac{1}{\gamma}\,\boldsymbol{F}^T\boldsymbol{W}(\boldsymbol{x})\,\boldsymbol{F} \;\equiv\; \boldsymbol{H}(\boldsymbol{x}) \tag{7.11}$$

is constant in the interior of each $D_k$ and jumps as $\boldsymbol{x}$ crosses one of the dividing hyperplanes.

**Example 7.2.** Consider the affine function $\boldsymbol{r} : \mathbb{R}^2 \mapsto \mathbb{R}^3$ given by

$$\boldsymbol{r}(\boldsymbol{x}) \;=\; \begin{pmatrix} 1.5 \\ 2.0 \\ -3.5 \end{pmatrix} - \begin{pmatrix} -0.5 & 2.0 \\ 3.0 & -1.0 \\ -1.0 & 0.5 \end{pmatrix} \boldsymbol{x}\;,$$

and let the threshold $\gamma = 0.5$.

In $\mathbb{R}^2$ a hyperplane is a straight line, and Figure 7.2 below shows the lines, along which $r_i(\boldsymbol{x}) = 0$ (full lines); the dividing hyperplanes, $r_i(\boldsymbol{x}) = \pm\gamma$ (dotted lines) and two of the subregions are indicated by hatching. For $\boldsymbol{x} \in D_k$ we have $\boldsymbol{s} = \begin{pmatrix} 0 & -1 & -1 \end{pmatrix}^T$, while $\boldsymbol{s} = \begin{pmatrix} -1 & 1 & -1 \end{pmatrix}^T$ for $\boldsymbol{x} \in D_j$.

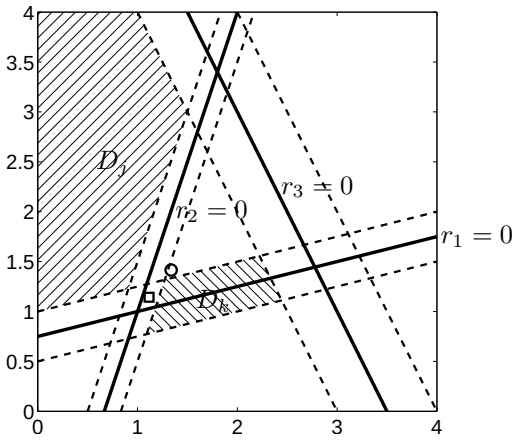



***Figure 7.2.*** *Dividing hyperplanes in* $\mathbb{R}^2$.

The square indicates the Huber solution for $\gamma = 0.5$, $\boldsymbol{x}_\gamma = \begin{pmatrix} 1.116 & 1.143 \end{pmatrix}^T$ with $\boldsymbol{s}_\gamma = \begin{pmatrix} 0 & 0 & -1 \end{pmatrix}^T$. For comparison, the circle marks the least squares solution to $\boldsymbol{r}(\boldsymbol{x}) \simeq \boldsymbol{0}$: $\boldsymbol{x}_{(2)} = \begin{pmatrix} 1.337 & 1.415 \end{pmatrix}^T$. ∎

With a slight modification of (5.13) we can show that the Hessian (7.11) is positive semidefinite. This implies that if we find a stationary point $\boldsymbol{x}_\mathrm{s}$, ie $\boldsymbol{\nabla} f_\gamma(\boldsymbol{x}_\mathrm{s}) = \boldsymbol{0}$, then it is a minimizer for $f_\gamma$: $\boldsymbol{x}_\gamma = \boldsymbol{x}_\mathrm{s}$. We use Newton's method with line search to find a stationary point.

**Algorithm 7.12. Linear Huber**
Given starting point $\boldsymbol{x}$ and threshold $\gamma$
**repeat**
  Find $\boldsymbol{v}$ by solving $\boldsymbol{H}(\boldsymbol{x})\boldsymbol{v} = -\boldsymbol{\nabla} f_\gamma(\boldsymbol{x})$
  $\boldsymbol{x} :=$ line_search$(\boldsymbol{x}, \boldsymbol{v})$
**until** STOP

The line search is very simple: $\varphi(\alpha) = f_\gamma(\boldsymbol{x} + \alpha\boldsymbol{v})$ is a piecewise quadratic in the scalar variable $\alpha$, so $\alpha^*$ is a root for $\varphi'$, which is piecewise linear; the coefficients change when $\boldsymbol{x} + \alpha\boldsymbol{v}$ passes a dividing hyperplane for increasing $\alpha$. The stopping criterion is that $\boldsymbol{s}_\gamma(\boldsymbol{x}+\alpha^*\boldsymbol{v}) = \boldsymbol{s}_\gamma(\boldsymbol{x})$. See [33] and [38] about details and efficient implementation.

**Example 7.3.** The function (7.9) may be the residual from data fitting with a linear model, cf Chapter 5. *Peter Huber* [29] introduced his M-estimator as a method for reducing the effect of wild points, cf Example 5.5, and from (7.8) we see that this is achieved by replacing a large $r_i$ by just the threshold $\gamma$ times the *sign* of this wild point residual. ∎

**Example 7.4.** Suppose that $\gamma$ is changed to $\delta$ and that $\boldsymbol{s}_\delta(\boldsymbol{x}_\delta) = \boldsymbol{s}_\gamma(\boldsymbol{x}_\gamma) = \boldsymbol{s}$, and therefore $\boldsymbol{W}_\delta(\boldsymbol{x}_\delta) = \boldsymbol{W}_\gamma(\boldsymbol{x}_\gamma) = \boldsymbol{W}$. Then it follows from (7.8) and (7.10) that the Huber solution is a linear function of the threshold,

$$\boldsymbol{x}_\delta \;=\; \boldsymbol{x}_\gamma + (\gamma - \delta)\left(\boldsymbol{F}^T\boldsymbol{W}\,\boldsymbol{F}\right)^{-1}\boldsymbol{F}^T\boldsymbol{s}\;. \tag{7.13}$$

Let $\boldsymbol{x}_{(2)}$ denote the least squares solution to $\boldsymbol{r}(\boldsymbol{x}) \simeq \boldsymbol{0}$. This is also the Huber solution (with $\boldsymbol{s} = \boldsymbol{0}$) for all $\gamma \geq \gamma_\infty \equiv \|\boldsymbol{r}(\boldsymbol{x}_{(2)})\|_\infty$. For $\gamma < \gamma_\infty$ one or more $r_i$ will be "large", so $\boldsymbol{s}$ changes.

Madsen and Nielsen [34] showed that there exists a $\gamma_0 > 0$ such that $\boldsymbol{s}_\gamma(\boldsymbol{x}^*_\gamma)$ is constant for $\gamma \leq \gamma_0$ and used this together with (7.13) to develop an efficient method for linear $L_1$ estimation. This might replace the LP algorithm used to solve the linearized problems in Section 7.2. ∎

### 7.3.2. Nonlinear Huber Estimation

Now consider a function $\boldsymbol{r}$ that depends nonlinearly on $\boldsymbol{x}$. We must use iteration to find the minimizer $\boldsymbol{x}_\gamma$ of $f_\gamma$. At the current iterate $\boldsymbol{x}$ we use the approximation (7.1) and the corresponding approximation to the objective function

$$\begin{aligned} f_\gamma(\boldsymbol{x}+\boldsymbol{h}) \;&\simeq\; L(\boldsymbol{h}) \\ &=\; \frac{1}{2\gamma}\,\boldsymbol{\ell}^T\boldsymbol{W}\,\boldsymbol{\ell} + \boldsymbol{\ell}^T\boldsymbol{s} - \tfrac{1}{2}\,\gamma\,\boldsymbol{s}^T\boldsymbol{s}\;, \end{aligned}$$

where $\boldsymbol{\ell} = \boldsymbol{\ell}(\boldsymbol{h}) = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\boldsymbol{h}$ and $\boldsymbol{s}$ and $\boldsymbol{W}$ are given by (7.5) with $\boldsymbol{r}(\boldsymbol{x})$ replaced by $\boldsymbol{\ell}(\boldsymbol{h})$.

As in Sections 7.1 and 7.2 we can combine this Gauss–Newton model with a trust region approach, cf [27]. Instead we shall describe a Levenberg–Marquardt like algorithm, cf 6.18, see Algorithm 7.14.

The linearized problem at 1° is solved by a slight modification of Algorithm 7.12 with starting point $\boldsymbol{v} = \boldsymbol{0}$. Note that $L(\boldsymbol{0}) = f_\gamma(\boldsymbol{x})$.

**Algorithm 7.14. Nonlinear Huber**
Given starting point $\boldsymbol{x}$, threshold $\gamma$ and $\mu > 0$. $\nu = 2$
**while not** STOP
  $\hat{\boldsymbol{h}} = \operatorname{argmin}_{\boldsymbol{h}} \{L(\boldsymbol{h}) + \frac{1}{2}\mu h^T h\}$ {1°}
  $\varrho := (f_\gamma(\boldsymbol{x}) - f_\gamma(\boldsymbol{x}+\hat{\boldsymbol{h}}))/(L(\boldsymbol{0}) - L(\hat{\boldsymbol{h}}))$
  **if** $\varrho > 0$ {step acceptable}
    $\boldsymbol{x} := \boldsymbol{x} + \hat{\boldsymbol{h}}$
    $\mu := \mu * \max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}$; $\nu := 2$
  **else**
    $\mu := \mu * \nu$; $\nu := 2 * \nu$
  **end**
 **end**
**end**

Finally, we give two examples of modified Huber functions with reduced influence from positive components of $\boldsymbol{r}$.

$$\widehat{\phi}_\gamma(u) = \begin{cases} |u| - \frac{1}{2}\gamma & \text{for } u < -\gamma \ , \\ \dfrac{1}{2\gamma}\, u^2 & \text{for } -\gamma \le u \le 0 \ , \\ 0 & \text{for } u > 0 \ . \end{cases}$$

$$\widetilde{\phi}_\gamma(u) = \begin{cases} \dfrac{1}{2\gamma}\, u^2 & \text{for } u \le \gamma \ , \\ u - \frac{1}{2}\gamma & \text{for } u > \gamma \ . \end{cases}$$

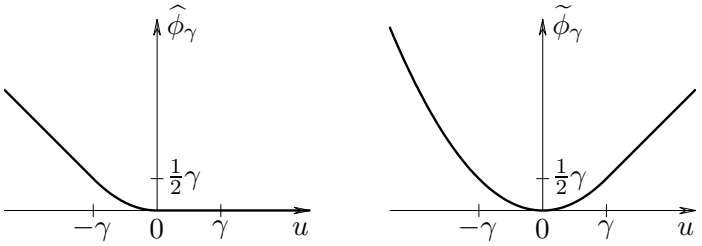



***Figure 7.3.*** *One-sided Huber functions.one-sided Huber function*

See [39] about implementation and some applications of these functions.

# Appendix A. Some Mathematical Background

## A.1. Norms and condition number

For a vector $\boldsymbol{x} \in \mathbb{R}^n$ the Hölder $p$-norm is defined by

$$\|\boldsymbol{x}\|_p \;=\; (|x_1|^p + \cdots + |x_n|^p)^{1/p} \;\;, \quad \text{for} \;\; p \geq 1 \;.$$

In the book we use three different $p$–norms:

$$\begin{aligned} \|\boldsymbol{x}\|_1 &= |x_1| + \cdots + |x_n| \;, \\ \|\boldsymbol{x}\|_2 &= \sqrt{x_1^2 + \cdots + x_n^2} \;, \\ \|\boldsymbol{x}\|_\infty &= \max_i\{|x_i|\} \;. \end{aligned} \tag{A.1}$$

If we do not give the $p$–value, we mean the 2–norm, and we sometimes make use of the identity

$$\|\boldsymbol{x}\|^2 \;=\; \|\boldsymbol{x}\|_2^2 \;=\; \boldsymbol{x}^T\boldsymbol{x} \;, \tag{A.2}$$

which is easily verified.

For a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ the *induced $p$–norm*is defined by

$$\|\boldsymbol{A}\|_p \;=\; \max_{\boldsymbol{x}\neq\boldsymbol{0}} \left\{ \frac{\|\boldsymbol{A}\,\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p} \right\} \;=\; \max_{\|\boldsymbol{v}\|_p=1} \{\|\boldsymbol{A}\,\boldsymbol{v}\|_p\} \;. \tag{A.3}$$

Corresponding to the three vector norms in (A.1) it can be shown that

$$\begin{aligned} \|\boldsymbol{A}\|_1 &= \max_j \{\|a_{1j}| + \cdots + |a_{mj}|\} \;, \\ \|\boldsymbol{A}\|_2 &= \sqrt{\max \lambda_j(\boldsymbol{A}^T\boldsymbol{A})} \;, \\ \|\boldsymbol{A}\|_\infty &= \max_i \{|a_{i1}| + \cdots + |a_{in}|\} \;. \end{aligned} \tag{A.4}$$

In the expression for the 2–norm $\lambda_j$ is the $j$th eigenvalue of the matrix $\boldsymbol{M} = \boldsymbol{A}^T\boldsymbol{A}$. In Appendix A.2 we show that this matrix has real, non-negative eigenvalues. For a symmetric matrix we have

$$\|\boldsymbol{A}\|_2 \;=\; \max\{|\lambda_j(\boldsymbol{A})|\} \quad \text{if} \;\; \boldsymbol{A}^T = \boldsymbol{A} \;. \tag{A.5}$$

The symmetry of $\boldsymbol{A}$ implies that it is square, $m = n$.

The *Frobenius norm* of matrix $\boldsymbol{A}$ is defined by

$$\|\boldsymbol{A}\|_F \;=\; \big(\,\textstyle\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2\,\big)^{1/2}\;.$$

This is recognized as the 2–norm of the vector in $\mathbb{R}^{mn}$ obtained by stacking the columns of $\boldsymbol{A}$.

The *condition number* $\kappa_p(\boldsymbol{A})$ of a full rank matrix $\boldsymbol{A}\in\mathbb{R}^{m\times n}$ is defined by

$$\kappa_p(\boldsymbol{A}) \;=\; \|\boldsymbol{A}\|_p \Big/ \min_{\boldsymbol{x}\neq\boldsymbol{0}} \left\{ \frac{\|\boldsymbol{A}\,\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p} \right\} \;=\; \|\boldsymbol{A}\|_p \max_{\boldsymbol{A}\boldsymbol{x}\neq\boldsymbol{0}} \left\{ \frac{\|\boldsymbol{x}\|_p}{\|\boldsymbol{A}\,\boldsymbol{x}\|_p} \right\}\;.$$

If $\boldsymbol{A}$ is square and nonsingular, then we get the equivalent formulation

$$\kappa_p(\boldsymbol{A}) \;=\; \|\boldsymbol{A}\|_p\,\|\boldsymbol{A}^{-1}\|_p\;,$$

and if $\boldsymbol{A}$ is symmetric, it follows from (A.5) and well-knows facts about the eigenvalues of the inverse matrix, that

$$\kappa_2(\boldsymbol{A}) \;=\; \max\{|\lambda_j(\boldsymbol{A})|\} \,/\, \min\{|\lambda_j(\boldsymbol{A})|\}\;. \tag{A.6}$$

## A.2. Symmetric, Positive Definite Matrices

The matrix $\boldsymbol{A}\in\mathbb{R}^{n\times n}$ is symmetric if $\boldsymbol{A}=\boldsymbol{A}^T$, ie if $a_{ij}=a_{ji}$ for all $i,j$.

**Definition A.7.** The symmetric matrix $\boldsymbol{A}\in\mathbb{R}^{n\times n}$ is

*positive definite* if $\boldsymbol{x}^T\boldsymbol{A}\,\boldsymbol{x}>0$ for all $\boldsymbol{x}\in\mathbb{R}^n,\ \boldsymbol{x}\neq\boldsymbol{0}$ ,

*positive semidefinite* if $\boldsymbol{x}^T\boldsymbol{A}\,\boldsymbol{x}\geq 0$ for all $\boldsymbol{x}\in\mathbb{R}^n,\ \boldsymbol{x}\neq 0$ .

Some useful properties of such matrices are listed in Theorem A.8 below. The proof can be found by combining theorems in almost any textbook on linear algebra and on numerical linear algebra.

A *unit lower triangular matrix* $\boldsymbol{L}$ is characterized by $\ell_{ii}=1$ and $\ell_{ij}=0$ for $j>i$. Note, that the LU factorization $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{U}$ is made **without** pivoting. Also note that points 4°–5° give the following relation between the LU and the *Cholesky factorization*

$$\boldsymbol{A} \;=\; \boldsymbol{L}\,\boldsymbol{U} \;=\; \boldsymbol{L}\,\boldsymbol{D}\,\boldsymbol{L}^T \;=\; \boldsymbol{C}^T\boldsymbol{C}\;,$$

showing that

$$\boldsymbol{C}=\boldsymbol{D}^{1/2}\boldsymbol{L}^T\;,\quad \text{with}\quad \boldsymbol{D}^{1/2}=\operatorname{diag}(\sqrt{u_{ii}})\;.$$

The Cholesky factorization can be computed directly (ie without the intermediate results $\boldsymbol{L}$ and $\boldsymbol{U}$) by Algorithm A.10 below, which includes a test for positive definiteness.

**Theorem A.8.** Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be symmetric and let $\boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{U}$, where $\boldsymbol{L}$ is a unit lower triangular matrix and $\boldsymbol{U}$ is an upper triangular matrix. Further, let $\{(\lambda_j, \boldsymbol{v}_j)\}_{j=1}^n$ denote the eigensolutions of $\boldsymbol{A}$, ie

$$\boldsymbol{A}\,\boldsymbol{v}_j \;=\; \lambda_j \boldsymbol{v}_j \;, \quad j = 1, \ldots, n \;. \tag{A.9}$$

Then

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

2° The following statements are equivalent
   a) $\boldsymbol{A}$ is positive definite (positive semidefinite)
   b) All $\lambda_j > 0$ ( $\lambda_j \geq 0$ )
   c) All $u_{ii} > 0$ ( $u_{ii} \geq 0$ ) .

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

3° The LU factorization is numerically stable.

4° $\boldsymbol{U} = \boldsymbol{D}\,\boldsymbol{L}^T$ with $\boldsymbol{D} = \mathrm{diag}(u_{ii})$.

5° $\boldsymbol{A} = \boldsymbol{C}^T\boldsymbol{C}$, the *Cholesky factorization.* $\boldsymbol{C} \in \mathbb{R}^{n\times n}$ is upper triangular.

**Algorithm A.10. Cholesky factorization**

**begin**
  $k := 0$; *posdef* := **true** {Initialisation}
  **while** *posdef* **and** $k < n$
    $k := k{+}1$; $d := a_{kk} - \sum_{i=1}^{k-1} c_{ik}^2$
    **if** $d > 0$ {test for pos. def.}
      $c_{kk} := \sqrt{d}$ {diagonal element}
      **for** $j := k{+}1, \ldots, n$
        $c_{kj} := \left(a_{kj} - \sum_{i=1}^{k-1} c_{ij}c_{ik}\right) / c_{kk}$ {superdiagonal elements}
    **else**
      *posdef* := **false**
**end**

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

$$\boldsymbol{C}^T\boldsymbol{z} = \boldsymbol{b} \qquad \text{and} \qquad \boldsymbol{C}\,\boldsymbol{x} = \boldsymbol{z}\ ,$$

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

From (A.9) it follows immediately that

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

for any $\mu \in \mathbb{R}$. Combining this with 2° in Theorem A.8 we see that if $\boldsymbol{A}$ is symmetric and positive semidefinite and $\mu > 0$, then the matrix $\boldsymbol{A}+\mu\boldsymbol{I}$ is also symmetric and it is guaranteed to be positive definite. Combining this with (A.6) we see that the condition number can be expressed as

$$\kappa(\boldsymbol{A}+\mu\boldsymbol{I}) \;=\; \frac{\max\{\lambda_j\} + \mu}{\min\{\lambda_j\} + \mu} \;\le\; \frac{\max\{\lambda_j\} + \mu}{\mu}\quad .$$

This is a decreasing function of $\mu$.

The eigenvalues of a matrix $\boldsymbol{A}$ can be bounded by means of *Gershgorin's theorem*:

$$|\lambda_j| \;\le\; \|\boldsymbol{A}\|_\infty \;=\; \max_i \sum_j |a_{ij}|\ . \tag{A.11}$$

## A.3. Difference Approximations

We wish to approximate the derivative of a function $f : \mathbb{R} \mapsto \mathbb{R}$ that is three times continuously differentiable. The Taylor expansion from $x$ is

$$f(x+h) \;=\; f(x) + hf'(x) + \tfrac{1}{2}\,h^2\,f''(x) + \tfrac{1}{6}\,h^3\,f'''(\xi)\ ,$$

where $\xi$ is between $x$ and $x+h$. By means of this we see that

$$\begin{aligned}
D_+(h) \;&\equiv\; \frac{f(x+h) - f(x)}{h} \;&=\; f'(x) + \tfrac{1}{2}\,h\,f''(x) + \tfrac{1}{6}\,h^2\,f'''(\xi_+)\ ,\\
D_-(h) \;&\equiv\; \frac{f(x) - f(x-h)}{h} \;&=\; f'(x) - \tfrac{1}{2}\,h\,f''(x) + \tfrac{1}{6}\,h^2\,f'''(\xi_-)\ ,\\
D_0(h) \;&\equiv\; \frac{f(x+h) - f(x-h)}{2h} \;&=\; f'(x) + \tfrac{1}{6}\,h^2\,f'''(\xi_0)\ .
\end{aligned} \tag{A.12}$$

Assuming that $h > 0$, these three expressions are respectively the *forward*, the *backward* and the *central difference approximation* to $f'(x)$. It follows that the first two approximations have error $O(h)$, while the central approximation has error $O(h^2)$. This means that by reducing $h$ we can reduce the error. On a computer, however, there is a lower limit on how small we can take $h$ before rounding errors spoil the results.

First, $x$ and $h$ are floating point numbers, and the argument $x+h$ is represented by the floating point number $\mathsf{fl}\big[x+h\big]$, which satisfies

$$\mathsf{fl}\big[x + h\big] \;=\; (x + h)(1 + \epsilon)\ , \quad |\epsilon| \le \varepsilon_{\mathsf{M}}\ ,$$

where $\varepsilon_{\mathsf{M}}$ is the *machine precision*. In general

$$\overline{h} \;=\; \mathsf{fl}\big[x + h\big] - x \;\neq\; h\ . \tag{A.13}$$

For instance $\overline{h} = 0$ if $x > 0$ and $0 < h \leq \varepsilon_{\mathsf{M}}\, x$. In order to simplify the further analysis we shall assume that $h$ is larger than this, and that the $h$ in (A.12) is the true step as defined by (A.13):

$$\texttt{xph} := \texttt{x + h;} \quad \texttt{h} := \texttt{xph - x}$$

In general $f(z)$ is not a floating point number, and the best that we can hope for is that

$$\overline{f}(z) \;=\; \mathsf{fl}\big[f(z)\big] \;=\; f(z)(1+\delta_z)\;, \quad |\delta_z| \leq K\,\varepsilon_{\mathsf{M}}\;,$$

where $K$ is a small positive number, $1 \leq K \leq 10$, say. The computed forward difference approximation is

$$\mathsf{fl}\big[D_+(h)\big] \;=\; \frac{f(x{+}h)(1+\delta_{x+h}) - f(x)(1+\delta_x)}{h}\,(1+\epsilon)\;,$$

where the error factor $(1+\epsilon)$ accounts for the rounding errors associated with the subtraction in the nominator and the division by $h$. If $h$ is sufficiently small the two terms in the nominator will have the same sign and the same order of magnitude, $\frac{1}{2} \leq \overline{f}(x{+}h)/\overline{f}(x) \leq 2$, and then there is no rounding error in the subtraction. The division error remains, corresponding to $\epsilon \leq \varepsilon_{\mathsf{M}}$, and ignoring this we get

$$\mathsf{fl}\big[D_+(h)\big] \;\simeq\; D_+(h) + \frac{f(x{+}h)\,\delta_{x+h} - f(x)\,\delta_x}{h} \;\simeq\; \frac{f(x)(\delta_{x+h} - \delta_x)}{h}\;.$$

We combine this with (A.12), and if $|f''(\xi) \leq A$ in the neighbourhood of $x$, then we get the error bound

$$|\mathsf{fl}\big[D_+(h)\big] - f'(x)| \;\leq\; \tfrac{1}{2}\,A\,h + 2K\,|f(x)|\,\varepsilon_{\mathsf{M}}\,h^{-1}\;.$$

## A.4. Orthogonal transformation

Let $\boldsymbol{x} \in \mathbb{R}^m$. The vector $\widetilde{\boldsymbol{x}} \in \mathbb{R}^m$ is obtained by an *orthogonal transformation* of $\boldsymbol{x}$, if

$$\widetilde{\boldsymbol{x}} \;=\; \boldsymbol{Q}\,\boldsymbol{x}\;,$$

where $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$ is an *orthogonal matrix*, ie $\boldsymbol{Q}^{-1} = \boldsymbol{Q}^T$, or

$$\boldsymbol{Q}^T\boldsymbol{Q} \;=\; \boldsymbol{Q}\,\boldsymbol{Q}^T \;=\; \boldsymbol{I}\;. \tag{A.14}$$

An orthogonal matrix $\boldsymbol{Q}$ has a number of important properties, for instance the following,

**Theorem A.15.** If $\boldsymbol{Q}$ is an orthogonal matrix, then

1° $\boldsymbol{Q}^T$ is also orthogonal.

2° Both the columns and the rows of $\boldsymbol{Q}$ are orthonormal.

3° $\|\boldsymbol{Q}\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2 \quad$ and $\quad (\boldsymbol{Q}\boldsymbol{x})^T(\boldsymbol{Q}\boldsymbol{y}) = \boldsymbol{x}^T\boldsymbol{y}$ .

4° $\|\boldsymbol{Q}\|_2 = 1 \quad$ and $\quad \kappa(\boldsymbol{Q}) = 1$ .

5° If $\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_N$ are orthogonal, then the product $\boldsymbol{Q} = \boldsymbol{Q}_1\boldsymbol{Q}_2\cdots\boldsymbol{Q}_N$ is orthogonal.

**Proof.**

1° Follows from (A.14) by interchanging the roles of $\boldsymbol{Q}$ and $\boldsymbol{Q}^T$.

2° The $(i,k)$th element in the matrix equation $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$ has the form

$$\boldsymbol{Q}_{:,i}^T\boldsymbol{Q}_{:,k} = \delta_{ik} = \begin{cases} 0 & \text{for } i \neq k \\ 1 & \text{for } i = k \end{cases} ,$$

showing that the columns $\{\boldsymbol{Q}_{:,j}\}$ are orthonormal. Similarly, $\boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{I}$ implies that the rows $\{\boldsymbol{Q}_{i,:}\}$ are orthonormal.

3° $\|\boldsymbol{Q}\boldsymbol{x}\|_2^2 = (\boldsymbol{Q}\boldsymbol{x})^T(\boldsymbol{Q}\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{x} = \|\boldsymbol{x}\|_2^2$ .
The proof of $\widetilde{\boldsymbol{x}}^T\widetilde{\boldsymbol{y}} = \boldsymbol{x}^T\boldsymbol{y}$ is similar.

4° $\|\boldsymbol{Q}\|_2 \equiv \max_{\boldsymbol{x}\neq\boldsymbol{0}} \{\|\boldsymbol{Q}\boldsymbol{x}\|_2/\|\boldsymbol{x}\|_2\} = \max_{\boldsymbol{x}\neq\boldsymbol{0}} \{\|\boldsymbol{x}\|_2/\|\boldsymbol{x}\|_2\} = 1$ .
Here, we used 3°. Similarly we see that $\min_{\boldsymbol{x}\neq\boldsymbol{0}} \{\|\boldsymbol{Q}\boldsymbol{x}\|_2/\|\boldsymbol{x}\|_2\} = 1$ , and $\kappa(\boldsymbol{Q}) = 1$ follows from Definition A.6 of the condition number.

5° We only need to show that the product of two orthogonal matrices is orthogonal: Let $\widetilde{\boldsymbol{Q}} = \boldsymbol{Q}_1\boldsymbol{Q}_2$, then

$$\widetilde{\boldsymbol{Q}}^T\widetilde{\boldsymbol{Q}} = \boldsymbol{Q}_2^T\boldsymbol{Q}_1^T\boldsymbol{Q}_1\,\boldsymbol{Q}_2 = \boldsymbol{Q}_2^T\,\boldsymbol{Q}_2 = \boldsymbol{I} ,$$

ie $\widetilde{\boldsymbol{Q}}$ is orthogonal.

□

The two properties in 3° can be expressed in words: An orthogonal transformation preserves the length of a vector and the pseudo angle[1)] between two vectors. Property 4° implies that orthogonal transformations do not enlarge effects of rounding errors. Finally, a transformation is often performed as a series of orthogonal subtransformations, and 5° shows that the complete transformation is orthogonal.

[1)] The pseudo angle $\theta$ between the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined by $\cos\theta = \boldsymbol{x}^T\boldsymbol{y}/(\|\boldsymbol{x}\|_2\|\boldsymbol{y}\|_2)$.

Orthogonal transformations can be constructed in a number of ways. We shall only present the so-called *Householder transformation:* Given a vector $w \in \mathbb{R}^m,\ w \neq 0$; the matrix[2)]

$$\boldsymbol{Q} \;=\; \boldsymbol{I} - \beta\, \boldsymbol{w}\, \boldsymbol{w}^T \quad \text{with} \quad \beta \;=\; \frac{2}{\boldsymbol{w}^T \boldsymbol{w}} \tag{A.16}$$

is easily shown to satisfy $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$, ie, it is orthogonal. A transformation with such a matrix is simple:

$$\widetilde{\boldsymbol{x}} \;=\; (\boldsymbol{I} - \beta\, \boldsymbol{w}\, \boldsymbol{w}^T)\boldsymbol{x} \;=\; \boldsymbol{x} + \gamma\, \boldsymbol{w}\ , \quad \text{where} \quad \gamma \;=\; -\beta\, \boldsymbol{w}^T \boldsymbol{x}\ . \tag{A.17}$$

The "cost" is $4m$ flops instead of the $2m^2$ flops required by a general matrix–vector multiplication.
The following lemma gives the background for the next section.

**Lemma A.18.** Given $\boldsymbol{x}$ and $\widetilde{\boldsymbol{x}}$ with $\boldsymbol{x} \neq \widetilde{\boldsymbol{x}}$ but $\|\boldsymbol{x}\|_2 = \|\widetilde{\boldsymbol{x}}\|_2$.
Let $\boldsymbol{w} = \boldsymbol{x} - \widetilde{\boldsymbol{x}}$, then (A.17) transforms $\boldsymbol{x}$ to $\widetilde{\boldsymbol{x}}$.

**Proof.** We first note that

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{w} \;&=\; (\boldsymbol{x} - \widetilde{\boldsymbol{x}})^T(\boldsymbol{x} - \widetilde{\boldsymbol{x}}) \\ &=\; \boldsymbol{x}^T\boldsymbol{x} + \widetilde{\boldsymbol{x}}^T\widetilde{\boldsymbol{x}} - 2\,\widetilde{\boldsymbol{x}}^T\boldsymbol{x} \\ &=\; 2(\boldsymbol{x}^T\boldsymbol{x} - \widetilde{\boldsymbol{x}}^T\boldsymbol{x}) \;=\; 2\,\boldsymbol{w}^T\boldsymbol{x}\ . \end{aligned}$$

Here we exploited that $\|\boldsymbol{x}\|_2 \;=\; \|\widetilde{\boldsymbol{x}}\|_2 \;\Leftrightarrow\; \boldsymbol{x}^T\boldsymbol{x} \;=\; \widetilde{\boldsymbol{x}}^T\widetilde{\boldsymbol{x}}$ . Next, the definitions in (A.16) give

$$\gamma \;=\; -\beta\, \boldsymbol{w}^T\boldsymbol{x} \;=\; \frac{-2}{\boldsymbol{w}^T\boldsymbol{w}}\, \boldsymbol{w}^T\boldsymbol{x} \;=\; -1\ ,$$

so that $\boldsymbol{Q}\boldsymbol{x} = \boldsymbol{x} - (\boldsymbol{x} - \widetilde{\boldsymbol{x}}) = \widetilde{\boldsymbol{x}}$ . □

## A.5. QR factorization

Given $\boldsymbol{x} \in \mathbb{R}^m$. We want to transform it into $\widetilde{\boldsymbol{x}} \in \mathbb{R}^m$ satisfying

$$\widetilde{x}_i \;=\; \begin{cases} x_i & i = 1, \ldots, k-1 \\ 0 & i = k+1, \ldots, m \end{cases}\ .$$

If we ensure that $\widetilde{x}_k^2 = \|\boldsymbol{x}_{k:m}\|_2^2 = x_k^2 + \cdots + x_m^2$, then the two vectors satisfy the conditions in Lemma A.18. For the sake of accuracy we choose the sign of

[2)] Remember that the outer product $\boldsymbol{w}\boldsymbol{w}^T$ is a *matrix* and the inner product $\boldsymbol{w}^T\boldsymbol{w}$ is a *scalar*.

$\widetilde{x}_k$ so that $|w_k|$ is as large as possible. The expression for $\beta$ can be found as in the proof of Lemma A.18.

$$\widetilde{\boldsymbol{x}} = \begin{pmatrix} x_1 \\ \vdots \\ x_{k-1} \\ \widetilde{x}_k \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \qquad \boldsymbol{w} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ x_k - \widetilde{x}_k \\ x_{k+1} \\ \vdots \\ x_m \end{pmatrix}, \tag{A.19}$$

where

$$\widetilde{x}_k \;=\; -\mathrm{sign}(x_k)\|\boldsymbol{x}_{k:m}\|_2\;, \qquad \beta \;=\; \frac{2}{\boldsymbol{w}^T\boldsymbol{w}} \;=\; \frac{-1}{\widetilde{x}_k w_k}\;. \tag{A.20}$$

Now, let $\boldsymbol{x} = \boldsymbol{A}_{:,1}$, the first column of a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$. By means of (A.19) with $k = 1$ we get

$$\boldsymbol{A}^{(1)} \;=\; \boldsymbol{Q}_1\,\boldsymbol{A} \;=\; \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{m2}^{(1)} & \cdots & a_{mn}^{(1)} \end{pmatrix},$$

where $a_{11}^{(1)} = \widetilde{x}_1$ and (A.17) is used to transform columns $2, \ldots, n$. Next, we compute $\boldsymbol{A}^{(2)} = \boldsymbol{Q}_2\boldsymbol{A}^{(1)}$, where $\boldsymbol{Q}_2$ corresponds to $\boldsymbol{w}^{(2)}$ found by (A.20) with $\boldsymbol{x} = \boldsymbol{A}^{(1)}_{:,2},\ k = 2$. The first element in $\boldsymbol{w}^{(2)}$ is zero, and therefore the first row is not changed, $\boldsymbol{A}^{(2)}_{1,:} = \boldsymbol{A}^{(1)}_{1,:}$. Also, the zeros in the first column of $\boldsymbol{A}^{(1)}$ imply that $(\boldsymbol{w}^{(2)})^T\boldsymbol{A}^{(1)}_{:,1} = 0$ and (A.17) shows that $\boldsymbol{A}^{(2)}_{:,1} = \boldsymbol{A}^{(1)}_{:,1}$. This means that the zeros obtained in the first transformation are not changed.

The process continues with the $\boldsymbol{Q}_3$ determined by $\boldsymbol{x} = \boldsymbol{A}^{(2)}_{:,3},\ k = 3$, etc. If, for some $k < n$ it happens that all $a_{ik}^{(k-1)} = 0,\ i = k, \ldots, m$, then the process breaks down; the rank of $\boldsymbol{A}$ is less than $n$ and it is outside the scope of this book to discuss that. Otherwise $\boldsymbol{A}$ has full rank and we finish after $n$ subtransformations with a "generalized upper triangular" matrix

$$\boldsymbol{A}^{(n)} \;=\; \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix} \;=\; \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \\ & & \boldsymbol{0} & \end{pmatrix},$$

where $r_{ij} = a_{ij}^{(i)}$ and all the diagonal elements $r_{ii} \neq 0$. Summarizing, we get $\boldsymbol{A}^{(n)} = \boldsymbol{Q}_n \cdots \boldsymbol{Q}_2 \boldsymbol{Q}_1 A$, or

$$\boldsymbol{A} \;=\; \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix} \;, \quad \text{with} \quad \boldsymbol{Q} = \boldsymbol{Q}_1^T \boldsymbol{Q}_2^T \cdots \boldsymbol{Q}_n^T \;. \tag{A.21}$$

This is a so-called *QR factorization* of $\boldsymbol{A}$. The last $m-n$ columns in $\boldsymbol{Q}$ do not contribute to the product, and another formulation is the so-called *"economy size"* (or *"thin"*) QR factorization:

$$\boldsymbol{A} \;=\; \hat{\boldsymbol{Q}}\, \boldsymbol{R} \;, \tag{A.22}$$

where the $m \times n$ matrix $\hat{\boldsymbol{Q}}$ consists of the first $n$ columns in $\boldsymbol{Q}$.

We derived (A.21) via Householder transformations, in which case the $\{\boldsymbol{Q}_k\}$ are symmetric, and we can write $\boldsymbol{Q} = \boldsymbol{Q}_1 \boldsymbol{Q}_2 \cdots \boldsymbol{Q}_n$. Note, however, that $\boldsymbol{Q}$ is not symmetric, $\boldsymbol{Q}^T = \boldsymbol{Q}_n \cdots \boldsymbol{Q}_2 \boldsymbol{Q}_1$.

There are other ways to compute a QR-factorization, and the result is not unique. However, if $\boldsymbol{A} = \hat{\boldsymbol{Q}}' \boldsymbol{R}'$ and $\boldsymbol{A} = \hat{\boldsymbol{Q}}'' \boldsymbol{R}''$ are two factorizations, then it can be shown that $\boldsymbol{R}'_{i,:} = s_i \boldsymbol{R}''_{i,:}$, where $s_i = 1$ or $s_i = -1$ and $s_i$ can be different for different values of $i$. This means that except for "row-wise sign" the R-factor is unique.

## A.6. Minimum norm least squares solution

Consider the least squares problem: Given $\boldsymbol{F} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{y} \in \mathbb{R}^m$ with $m \geq n$, find $\boldsymbol{x} \in \mathbb{R}^n$ such that $\|\boldsymbol{y} - \boldsymbol{F}\boldsymbol{x}\|$ is minimized. To analyze this, we shall use the *singular value decomposition* (SVD) of $\boldsymbol{F}$,

$$\boldsymbol{F} \;=\; \boldsymbol{U}\, \boldsymbol{\Sigma}\, \boldsymbol{V}^T \;=\; \begin{pmatrix} \hat{\boldsymbol{U}} & \breve{\boldsymbol{U}} \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{\Sigma}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{V}} & \breve{\boldsymbol{V}} \end{pmatrix}^T \;=\; \hat{\boldsymbol{U}}\, \tilde{\boldsymbol{\Sigma}}\, \hat{\boldsymbol{V}}^T \;, \tag{A.23}$$

where the matrices $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ are orthogonal, and $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$ is "diagonal":

$$\hat{\boldsymbol{\Sigma}} \;=\; \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_p \end{pmatrix} \quad \text{with} \quad \sigma_1 \geq \cdots \geq \sigma_p > 0 \;,$$

and if $p < n$ we extend with $\sigma_{p+1} = \cdots = \sigma_n = 0$. The $\sigma_j$ are the *singular values* and the number $p$ is the *rank* of $\boldsymbol{F}$, equal to the dimension of the *range* of $\boldsymbol{F}$, $\mathcal{R}(\boldsymbol{F}) \subseteq \mathbb{R}^m$, cf Definition 5.26, page 87.

The columns in $\boldsymbol{U}$ and $\boldsymbol{V}$ can be used as orthonormal bases in $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively, and it follows from (A.23) that

$$\boldsymbol{F}\, \boldsymbol{V}_{:,j} \;=\; \sigma_j\, \boldsymbol{U}_{:,j} \;, \quad j = 1, \ldots, n \;.$$

Further, letting $\eta_j = \boldsymbol{U}_{:,j}^T \boldsymbol{y}$ and $\xi_j = \boldsymbol{V}_{:,j}^T \boldsymbol{x}$, we can write

$$\boldsymbol{y} \;=\; \sum_{j=1}^{m} \eta_j\, \boldsymbol{U}_{:,j} \;, \qquad \boldsymbol{x} \;=\; \sum_{j=1}^{n} \xi_j\, \boldsymbol{V}_{:,j} \;,$$

and we get

$$\boldsymbol{r} \;=\; \boldsymbol{y} - \boldsymbol{F}\,\boldsymbol{x} \;=\; \sum_{j=1}^{p}(\eta_j - \sigma_j\xi_j)\boldsymbol{U}_{:,j} + \sum_{j=p+1}^{m}\eta_j\boldsymbol{U}_{:,j}\ .$$

By means of the orthonormality of the columns in $\boldsymbol{U}$ this leads to

$$\|\boldsymbol{r}\|^2 \;=\; \boldsymbol{r}^T\boldsymbol{r} \;=\; \sum_{j=1}^{p}(\eta_j - \sigma_j\xi_j)^2 + \sum_{j=p+1}^{m}\eta_j^2 \quad , \tag{A.24}$$

which is minimized when

$$\eta_j - \sigma_j\xi_j \;=\; 0\ , \quad j = 1,\ldots,p\ .$$

Thus, the least squares solution can be expressed as

$$\hat{\boldsymbol{x}} \;= \sum_{j=1}^{p}\frac{\eta_j}{\sigma_j}\,\boldsymbol{V}_{:,j} + \sum_{j=p+1}^{n}\xi_j\boldsymbol{V}_{:,j}\ .$$

When the matrix $\boldsymbol{F}$ is *rank deficient*, ie when $p<n$, the least squares solution has $n-p$ degrees of freedom: $\xi_{p+1},\ldots,\xi_n$ are arbitrary. Similar to the discussion around (A.24) we see that $\|\hat{\boldsymbol{x}}\|$ is minimized when $\xi_{p+1} = \cdots = \xi_n = 0$. The solution corresponding to this choice of the free parameters is the so-called *minimum norm solution*,

$$\hat{\boldsymbol{x}}_{\min} \;= \sum_{j=1}^{p}\frac{\beta_j}{\sigma_j}\,\boldsymbol{V}_{:,j} \;= \sum_{j=1}^{p}\frac{\boldsymbol{U}_{:,j}^T\boldsymbol{y}}{\sigma_j}\,\boldsymbol{V}_{:,j}\ . \tag{A.25}$$

Put another way: the four fundamental subspaces from Definition 5.26 are spanned by the columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ as follows,

| $\mathcal{R}(\boldsymbol{F})$ | $\mathcal{R}(\boldsymbol{F}^T)$ | $\mathcal{N}(\boldsymbol{F})$ | $\mathcal{N}(\boldsymbol{F}^T)$ |
|---|---|---|---|
| span($\hat{\boldsymbol{U}}$) | span($\hat{\boldsymbol{V}}$) | span($\check{\boldsymbol{V}}$) | span($\check{\boldsymbol{U}}$) |

Thus, the minimum norm solution is obtained by setting the arbitrary component in $\mathcal{N}(\boldsymbol{F})$ equal to zero.

We shall also use the SVD analysis to show some properties of the normal equations as defined in Section 5.2.1 and two of the Gauss–Newton methods from Chapter 6: It follows from (A.23) and the orthogonality of $\boldsymbol{U}$ that

$$\begin{aligned}\boldsymbol{A} \;&=\; \boldsymbol{F}^T\boldsymbol{F} \;=\; \boldsymbol{V}\,\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\,\boldsymbol{V}^T \;=\; \textstyle\sum_{j=1}^{p}\sigma_j^2\,\boldsymbol{V}_{:,j}\,\boldsymbol{V}_{:,j}^T\ ,\\ \boldsymbol{b} \;&=\; \boldsymbol{F}^T\boldsymbol{y} \;=\; \boldsymbol{V}\,\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{y} \;=\; \textstyle\sum_{j=1}^{p}\sigma_j\,\eta_j\,\boldsymbol{V}_{:,j}\ .\end{aligned}$$

This shows that the normal equations system is *consistent*: $\boldsymbol{b} \in \mathcal{R}(\boldsymbol{A}) = \text{span}\,(\boldsymbol{V}_{:,1},\ldots,\boldsymbol{V}_{:,p})$.

Now consider the Gauss–Newton model from Section 6.1,

$$\boldsymbol{r}(\boldsymbol{x}+\boldsymbol{h}) \;\simeq\; \boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\,\boldsymbol{h}\ ,$$

and let $\boldsymbol{J}(\boldsymbol{x}) = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. The step $\boldsymbol{h}_{\text{gn}}$ is a least squares solution to $\boldsymbol{J}(\boldsymbol{x})\boldsymbol{h} \simeq$

$-\boldsymbol{r}(\boldsymbol{h})$, and in case of a rank deficient Jacobian we want the minimum norm step, as given by the right-hand side in (A.25) with $\boldsymbol{y}$ replaced by $-\boldsymbol{r}(\boldsymbol{x})$.

In case of the Levenberg–Marquardt methodL–M method from Section 6.2 the solution to the system $(\boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{I})\boldsymbol{h}_{\mathrm{lm}} = -\boldsymbol{J}^T\boldsymbol{r}$ can be expressed as

$$\boldsymbol{h}_{\mathrm{lm}} \;=\; \sum_{j=1}^{p} \frac{\sigma_j \beta_j}{\sigma_j^2 + \mu}\, \boldsymbol{V}_{:,j} \;, \qquad \beta_j = -\boldsymbol{U}_{:,j}^T \left(\boldsymbol{J}^T\boldsymbol{r}\right) \;.$$

This shows that the step has zero component in $\mathcal{N}(\boldsymbol{J})$, and if $\mu \ll \sigma_p$, then $\boldsymbol{h}_{\mathrm{lm}}$ is almost equal to the minimum norm least squares solution to $\boldsymbol{J}\,\boldsymbol{h} \simeq -\boldsymbol{r}$.

## A.7. Basic statistical concepts

Let $Z$ be a random variable with outcomes (measured values) $z_1, \ldots, z_p$. The *expected value* and *variance* of $Z$ are

$$\overline{z} \;=\; \mathsf{E}\big[Z\big] \;, \qquad \mathsf{V}\big[Z\big] \;=\; \mathsf{E}\big[(Z-\overline{z})^2\big] \;.$$

$\overline{z}$ may be interpreted as the average of $z_1, \ldots, z_p$ for $p \to \infty$. We also use the formulation

$$Z \;=\; \overline{z} + E \quad \text{with} \quad \mathsf{E}\big[E\big] \;=\; 0 \;, \quad \mathsf{V}\big[E\big] \;=\; \mathsf{V}\big[Z\big] \;. \tag{A.26}$$

If $X$ and $Y$ are independent random variables and $c$ is a constant, then

$$\begin{aligned} \mathsf{E}\big[X+Y\big] \;&=\; \mathsf{E}\big[X\big] + \mathsf{E}\big[Y\big] \;, \\ \mathsf{E}\big[c\cdot X\big] \;&=\; c\cdot\mathsf{E}\big[X\big] \;, \\ \mathsf{E}\big[X\cdot Y\big] \;&=\; \mathsf{E}\big[X\big]\cdot\mathsf{E}\big[Y\big] \;. \end{aligned} \tag{A.27}$$

Next, let $\boldsymbol{Z}$ be a random vector with $m$ elements. We can still use (A.26), where now $\overline{z}_i = \mathsf{E}\big[Z_i\big]$, $i = 1, \ldots, m$. The *variance–covariance matrix* is

$$\boldsymbol{V} \;=\; \mathsf{V}\big[\boldsymbol{Z}\big] \;=\; \mathsf{E}\big[\boldsymbol{E}\,\boldsymbol{E}^T\big] \;. \tag{A.28}$$

Note that $v_{ii} = \mathsf{V}\big[Z_i\big]$ and $v_{ij} = 0$ if $Z_i$ and $Z_j$ are independent (also said to be *uncorrelated*).

Now consider a continuously differentiable function $u : \mathbb{R}^n \mapsto \mathbb{R}$,

$$u \;=\; u(\boldsymbol{z}) \;=\; u(z_1, \ldots, z_n) \;,$$

where $\boldsymbol{z}$ is an outcome of a random vector $\boldsymbol{Z}$. We introduce (A.26), and by Taylor's theorem (with the assumption that the $\{e_j\}$ are small) we get

$$u(\overline{\boldsymbol{z}} + \boldsymbol{e}) \;\simeq\; u(\overline{z}) + \boldsymbol{g}^T\boldsymbol{e} \quad \text{with} \quad g_j = \frac{\partial u}{\partial z_j}(\overline{\boldsymbol{z}}) \;,$$

and from (A.27) and (A.28) we see that

$$\begin{aligned} \mathsf{E}\big[u(\boldsymbol{Z})\big] &= u(\overline{\boldsymbol{z}}) \,, \\ \mathsf{V}\big[u(\boldsymbol{Z})\big] &= \mathsf{E}\big[\boldsymbol{g}^T\boldsymbol{e}\;\boldsymbol{e}^T\boldsymbol{g}\big] \;=\; \boldsymbol{g}^T\boldsymbol{V}\,\boldsymbol{g} \,. \end{aligned} \tag{A.29}$$

If the $Z_i$ are independent, then the variance simplifies to

$$\mathsf{V}\big[u(\boldsymbol{Z})\big] \;=\; \sum_{j=1}^{n} \left(\frac{\partial u}{\partial z_j}(\overline{\boldsymbol{z}})\right)^2 \mathsf{V}\big[Z_j\big] \,. \tag{A.30}$$

As an important consequence, let $\boldsymbol{u} = \boldsymbol{A}\,\boldsymbol{z}$, where $\boldsymbol{A}$ is a constant matrix. By applying (A.29) to $u_i(\boldsymbol{z}) = \boldsymbol{A}_{i,:}\,\boldsymbol{z}$ we see that

$$\mathsf{E}\big[\boldsymbol{u}\big] \;=\; \boldsymbol{A}\,\overline{\boldsymbol{z}} \,, \qquad \mathsf{V}\big[\boldsymbol{u}\big] \;=\; \boldsymbol{A}\,\boldsymbol{V}\,\boldsymbol{A}^T \,. \tag{A.31}$$

## A.8. Cubic splines

A cubic spline $s(t)$ with *knots* $\tau_0 < \tau_1 \le \tau_2 \le \cdots \le \tau_{n-1} < \tau_n$ is a piecewise 3rd degree polynomial. In each knot interval $s(t)$ is a cubic, $s(t) = s_j(t)$ for $\tau_{j-1} \le t < \tau_j$, and the $s_j$ satisfy

$$\left.\begin{aligned} s_j(\tau_j-) &= s_{j+1}(\tau_j+) \\ s_j'(\tau_j-) &= s_{j+1}'(\tau_j+) \\ s_j''(\tau_j-) &= s_{j+1}''(\tau_j+) \end{aligned}\right\} \quad j = 1, \ldots, n{-}1 \,.$$

Therefore, if the knots are distinct, then $s \in \mathsf{C}^2[\tau_0, \tau_n]$, the space of twice continuously differentiable functions on the interval $[\tau_0, \tau_n]$. In each open knot interval $s'''(t)$ is constant, and in general $s'''(\tau_j-) \ne s'''(\tau_j+)$. If $\tau_j$ is a knot of multiplicity $d > 1$, then there may be a jump in $s^{(4-d)}(t)$ across this knot. Especially if $d \ge 4$, then $s$ splits into two separate splines.

The spline can be represented in a number of ways. The most straightforward is the so-called `pp` (piecewise polynomial) representation,

$$s_j(t) \;=\; a_j(t-\tau_{j-1})^3 + b_j(t-\tau_{j-1})^2 + c_j(t-\tau_{j-1}) + d_j \,. \tag{A.32}$$

This shows that the spline is given by the knots and $4n$ polynomial coefficients. However, the continuity of $s$, $s'$ and $s''$ across the interior knots gives $3(n{-}1)$ conditions, and this leaves

$$4n - 3(n{-}1) \;=\; n{+}3 \;\; \textit{degrees of freedom} \,.$$

Thus, we need $n{+}3$ linearly independent conditions in order to determine $s$ uniquely. If, for instance, we want to interpolate a function $f$ with known values $f(\tau_j) = y_j$, $j = 0, 1, \ldots, n$, we can use the $n{+}1$ conditions $s(\tau_j) = y_j$, $j = 0, 1, \ldots, n$, supplied with two boundary conditions, for instance given values

for the end point slopes $s'(\tau_0)$ and $s'(\tau_n)$, or given values for $s''(\tau_0)$ and $s''(\tau_n)$. A *"natural cubic spline"* satisfies $s''(\tau_0) = s''(\tau_n) = 0$.

Another representation that is better suited for determining a fitting spline is

$$s(t) \;=\; \sum_{j=1}^{n+3} c_j \, B_j(t) \;, \tag{A.33}$$

where the $B_j$ are so-called *B-splines*. Each $B_j$ is a cubic spline; they are linearly independent, and their number equals the dimension of the space that $s$ belongs to. Therefore they form a basis of that space.

The $B_j$ are only defined on $[\tau_0, \tau_n]$ and satisfy $B_j(t) \geq 0$. They have *local support*: $B_j(t)$ is nonzero only for $\tau_{j-4} < t < \tau_j$, ie in four consecutive knot intervals. Put another way: in the interval $[\tau_{j-1}, \tau_j]$ the only nonzero basis splines are $B_{j:j+3}$. They can be computed by the recurrence

$$\begin{aligned}
&B_{j,0}(t) = 1/(\tau_j - \tau_{j-1}) \;, \qquad B_{i,0}(t) = 0 \;\; \text{for } i \neq j \;, \\
&B_{i,r+1}(t) \;=\; \frac{(t - \tau_{i-2-r}) B_{i-1,r}(t) + (\tau_i - t) B_{i,r}(t)}{\tau_i - \tau_{i-2-r}} \;, \quad r = 0, 1 \;, \\
&B_i(t) \;=\; (t - \tau_{i-4}) B_{i-1,2}(t) + (\tau_i - t) B_{i,2}(t) \;.
\end{aligned}$$

Omitting nonzero values, the computation can be illustrated as shown in Figure A.1. The intermediate values can be used to compute derivatives of the

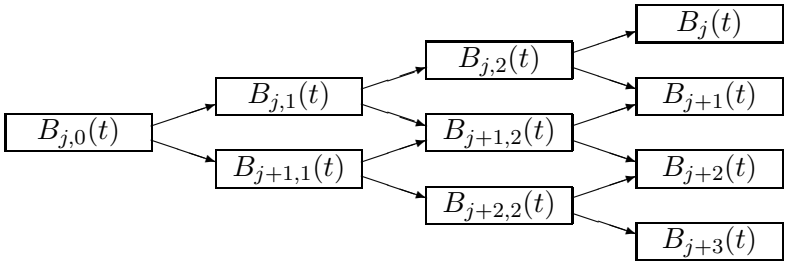



***Figure A.1.*** *Nonzero values in recurrence for B-splines.*

B-splines:

$$\begin{aligned}
B_i'(t) \;&=\; 3\,(B_{i-1,2}(t) - B_{i,2}(t)) \;, \\
B_i''(t) \;&=\; 6\left(\frac{B_{i-2,1}(t) - B_{i-1,1}(t)}{\tau_{i-1} - \tau_{i-4}} - \frac{B_{i-1,1}(t) - B_{i,1}(t)}{\tau_i - \tau_{i-3}}\right) \;.
\end{aligned}$$

## A.9. LP problems

An LP (linear programming) problem (also called a linear optimization problem has the form

$$\begin{aligned} &\text{minimize the linear function} \quad \boldsymbol{c}^T\boldsymbol{x} \\ &\text{subject to the linear constraints} \quad \boldsymbol{A}\,\boldsymbol{x} \;\geq\; \boldsymbol{b}\,, \end{aligned}$$

where $\boldsymbol{c}\in\mathbb{R}^n$, $\boldsymbol{A}\in\mathbb{R}^{m\times n}$ and $\boldsymbol{b}\in\mathbb{R}^m$ are given, and we seek the minimizer $\hat{\boldsymbol{x}}\in\mathbb{R}^n$. The solution of this type of problem plays an important role in for instance finance and operations research. There are a number of efficient methods for solving LP problems. It is outside the scope of the present version of the book, however, to describe them. We refer to [44, Chapters 13 – 14].

# Appendix B. Selected Proofs

## B.1. Proof of Theorem 2.41

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

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

We use the notation $\boldsymbol{g}_i = \boldsymbol{\nabla} f(\boldsymbol{x}_i)$ and define the search directions by $\boldsymbol{h}_i = \boldsymbol{x}_i - \boldsymbol{x}_{i-1}$. Then (2.35) leads to

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

and Algorithm 2.37 and (2.40) combine to

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

and $\alpha_{r+1}$ found by exact line search. Finally, we remind the reader of the following results from the proof of Theorem 2.39

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

Now we are ready for the induction:

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

Next, assume that (B.1) holds for all $j=1,\ldots,k$. Then it follows from the proof of Theorem 2.39 that

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

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

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

Thus, the gradients at the iterates are orthogonal,

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

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

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

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

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

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

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

## B.2. Proof of Theorem 3.4

The Taylor expansion of $\boldsymbol{\nabla} f$ from $\boldsymbol{x}$ is

$$\boldsymbol{\nabla} f(\boldsymbol{x}+\boldsymbol{h}) \;=\; \boldsymbol{\nabla} f(\boldsymbol{x}) + \boldsymbol{\nabla}^2 f(\boldsymbol{x})\, \boldsymbol{h} + \boldsymbol{\eta} \;,$$

where the remainder term $\boldsymbol{\eta}$ depends on $\boldsymbol{h}$. For $\boldsymbol{h} = \hat{\boldsymbol{x}} - \boldsymbol{x}$ we get

$$\boldsymbol{\nabla} f(\boldsymbol{x}) + \boldsymbol{\nabla}^2 f(\boldsymbol{x})(\hat{\boldsymbol{x}} - \boldsymbol{x}) + \boldsymbol{\eta} \;=\; \boldsymbol{\nabla} f(\hat{\boldsymbol{x}}) \;=\; \boldsymbol{0} \;.$$

From the definition of the Newton step we see that

$$\boldsymbol{\nabla} f(\boldsymbol{x}) + \boldsymbol{\nabla}^2 f(\boldsymbol{x})(\boldsymbol{x}_{\text{new}} - \boldsymbol{x}) \;=\; \boldsymbol{0} \;,$$

and by subtracting the last two equations we get

$$\boldsymbol{\nabla}^2 f(\boldsymbol{x})(\boldsymbol{x}_{\text{new}} - \hat{\boldsymbol{x}}) \;=\; -\boldsymbol{\eta} \;.$$

According to Theorem 1.6 and the definition of $O(\cdot)$ on page 5 there exist numbers $\delta_1, c_1 > 0$ such that

$$\|\boldsymbol{\eta}\| \;\leq\; c_1 \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 \quad \text{for} \quad \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \leq \delta_1 \;.$$

Further, $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ is nonsingular in a neighbourhood around $\hat{\boldsymbol{x}}$. This means that there exist numbers $\delta_2, c_2 > 0$ such that if $\|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \leq \delta_2$, then

$$\| \left(\boldsymbol{\nabla}^2 f(\boldsymbol{x})\right)^{-1} \boldsymbol{\eta}\| \;\leq\; c_2 \, \|\boldsymbol{\eta}\| \;.$$

Therefore, if $\|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \leq \delta \equiv \min\{\delta_1, \delta_2\}$, then

$$\|\boldsymbol{x}_{\text{new}} - \hat{\boldsymbol{x}}\| \;\leq\; c_1 c_2 \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 \;.$$

For sufficiently small $\delta$ we have $c_1 c_2 \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \leq c_1 c_2 \delta < 1$, implying that $\boldsymbol{x}_{\text{new}}$ is even closer to $\hat{\boldsymbol{x}}$, and the ensuing iterates stay in this neighbourhood, and the last equation shows that there is quadratic convergence as defined in (1.8). □

## B.3. Proof of (5.25)

Let the symmetric, positive semidefinite matrix $\boldsymbol{A} = \boldsymbol{F}^T\boldsymbol{F}$ have the eigensolutions $(\lambda_j, \boldsymbol{v}_j)$,

$$\boldsymbol{A}\,\boldsymbol{v}_j \;=\; \lambda_j \boldsymbol{v}_j \;, \quad j = 1, \ldots n \;.$$

All the eigenvalues are real and nonnegative, and we assume that they are numbered so that

$$\lambda_1 \;\geq\; \lambda_2 \;\geq\; \cdots \;\geq\; \lambda_n \;\geq\; 0 \;.$$

Further, the $\{\boldsymbol{v}_j\}$ are *orthonormal*, ie $\boldsymbol{v}_k^T\boldsymbol{v}_j = 0$ for $j \neq k$ and $\boldsymbol{v}_k^T\boldsymbol{v}_k = 1$.

Now, for any $\boldsymbol{x} \in \mathbb{R}^n$ we have

$$\begin{aligned} \boldsymbol{x} \;&=\; \xi_1\boldsymbol{v}_1 + \cdots + \xi_n\boldsymbol{v}_n \quad \text{with} \quad \xi_j = \boldsymbol{v}_j^T\boldsymbol{x} \;, \\ \boldsymbol{A}\,\boldsymbol{x} \;&=\; \lambda_1\xi_1\boldsymbol{v}_1 + \cdots + \lambda_n\xi_n\boldsymbol{v}_n \;, \end{aligned}$$

and

$$\frac{\|\boldsymbol{A}\boldsymbol{x}\|_2^2}{\|\boldsymbol{x}\|_2^2} \;=\; \frac{\lambda_1^2\xi_1^2 + \cdots + \lambda_n^2\xi_n^2}{\xi_1^2 + \cdots + \xi_n^2} \;\leq\; \lambda_1^2$$

with the maximum attained for $\boldsymbol{x} = \beta\boldsymbol{v}_1$, ie $\xi_2 = \cdots = \xi_n = 0$. Comparing with the definition (5.18) we have shown that

$$\|\boldsymbol{A}\|_2 \;=\; \lambda_1 \;.$$

Similarly we see that $\max_{\boldsymbol{x}\neq\boldsymbol{0}}\{\|\boldsymbol{x}\|_2/\|\boldsymbol{A}\boldsymbol{x}\|_2\} = 1/\lambda_n$ (attained for $\boldsymbol{x} = \beta\boldsymbol{v}_n$), and by means of definition (5.17) we get

$$\kappa(\boldsymbol{A}) = \lambda_1/\lambda_n \;.$$

Next,

$$\frac{\|\boldsymbol{F}\boldsymbol{x}\|_2^2}{\|\boldsymbol{x}\|_2^2} \;=\; \frac{\boldsymbol{x}^T\boldsymbol{F}^T\boldsymbol{F}\,\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} \;=\; \frac{\boldsymbol{x}^T\boldsymbol{A}\,\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} \;=\; \frac{\lambda_1\xi_1^2 + \cdots + \lambda_n\xi_n^2}{\xi_1^2 + \cdots + \xi_n^2} \;\leq\; \lambda_1 \;,$$

leading to

$$\|\boldsymbol{F}\|_2 \;=\; \sqrt{\lambda_1} \;=\; \sqrt{\|\boldsymbol{A}\|_2} \;.$$

Similarly we see that $\max_{\boldsymbol{x}\neq\boldsymbol{0}}\{\|\boldsymbol{x}\|_2/\|\boldsymbol{F}\boldsymbol{x}\|_2\} = 1/\sqrt{\lambda_n}$ , so that

$$\kappa(\boldsymbol{F}) \;=\; \sqrt{\lambda_1/\lambda_n} \;=\; \sqrt{\kappa(\boldsymbol{A})} \;.$$

□

## B.4. Proof of Theorem 5.28

Let $\boldsymbol{v} = \boldsymbol{R}\boldsymbol{x}$, then

$$\boldsymbol{F}\,\boldsymbol{x} \;=\; \widetilde{\boldsymbol{Q}}\,\boldsymbol{R}\,\boldsymbol{x} \;=\; \widetilde{\boldsymbol{Q}}\,\boldsymbol{v} \;=\; \sum_{j=1}^{n} v_j \boldsymbol{Q}_{:,j} \;.$$

This shows that $\boldsymbol{Q}_{:,j} \in \mathcal{R}(\boldsymbol{F})$, $j = 1, \ldots, n$. Further, because they are orthogonal, these $n$ vectors are linearly independent, and the number of them is equal to the dimension of the subspace. Therefore the columns of $\widetilde{\boldsymbol{Q}}$ is a basis for $\mathcal{R}(\boldsymbol{F})$. This basis is orthonormal.

Next, any vector $\boldsymbol{y} \in \text{span}(\boldsymbol{Q}_{:,n+1:m})$ can be expressed as

$$\boldsymbol{y} \;=\; \sum_{j=1}^{m-n} u_j \boldsymbol{Q}_{:,n+j} \;=\; \widehat{\boldsymbol{Q}}\,\boldsymbol{u}\;,$$

and by means of (5.21) we see that

$$\boldsymbol{F}^T\boldsymbol{y} \;=\; \boldsymbol{R}^T\widetilde{\boldsymbol{Q}}^T\,\widehat{\boldsymbol{Q}}\,\boldsymbol{u} \;=\; \boldsymbol{R}^T\,\boldsymbol{0}\,\boldsymbol{u} \;=\; \boldsymbol{0}\;.$$

This shows that every column in $\widehat{\boldsymbol{Q}}$ belongs to $\mathcal{N}(\boldsymbol{F}^T)$, and proceeding as before we see that they form an orthonormal basis of this nullspace.

Finally, all columns in $\boldsymbol{Q}$ form an orthonormal basis of the whole space $\mathbb{R}^m$. This implies that we can write

$$\boldsymbol{y} \;=\; \sum_{j=1}^{m} u_j \boldsymbol{Q}_{:,j} \;=\; \boldsymbol{Q}\,\boldsymbol{u}\;,$$

and (5.29) is derived by splitting the sum after the $n$th element. Again (5.21) can be used to prove the orthogonality of the two components:

$$\widetilde{\boldsymbol{y}}^T\widehat{\boldsymbol{y}} \;=\; \widetilde{\boldsymbol{u}}^T\,\widetilde{\boldsymbol{Q}}^T\widehat{\boldsymbol{Q}}\,\widehat{\boldsymbol{u}} \;=\; \widetilde{\boldsymbol{u}}^T\,\boldsymbol{0}\,\widehat{\boldsymbol{u}} \;=\; 0\;.$$

□

# Bibliography


[1] M. Al-Baali and R. Fletcher: Variational methods for non-linear least-squares. *J. Opl. Res. Soc.* **36**, No. 5, pp 405–421, 1985.

[2] I. Barrodale and F.D.K. Roberts: Solution of an overdetermined system of equations in the $\ell_1$ norm. *Commun. ACM* **17**, No. 6, pp 319–320, 1974.

[3] Å. Björck. *Numerical methods for least squares problems.* SIAM, Philadelphia, 1996.

[4] A.J. Booker, J.E. Dennis, P.D. Frank, D,B. Serafini, V. Torczon and M.W. Trosset: A rigorous framework for optimization of expensive functions by surrogates. *Structural and Multidisciplinary Optimization* **17**(1), pp 1 – 13, 1999.

[5] M.J. Box, D. Davies and W.H. Swan: *Non-linear optimization techniques.* ICI Monograph No 5, Oliver & Boyd, Edinburgh, 1969.

[6] C.G. Broyden: A class of methods for solving nonlinear simultaneous equations. *Maths. Comp.* **19**, pp 577–593, 1965.

[7] C.G. Broyden: Quasi–Newton methods and their application to function minimization. *Math. Comps.* **21**, pp 368–381, 1967.

[8] O. Caprani, K. Madsen and H.B. Nielsen: *Introduction to interval analysis.* IMM, DTU, (81 pages), 2002. Available as http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/1462/pdf/imm1462.pdf

[9] G. Corliss, C. Faure, A. Griewank, L. Hascoët and U. Naumann (eds): *Automatic differentiation of algorithms.* Springer, New York, 2002.

[10] M.G. Cox: *The incorporation of boundary conditions in spline approximation problems.* NPL Report NAC 80, 1977.

[11] M.G. Cox: *Practical spline approximation.* NPL Report DITC 1/82, 1982.

[12] H. P. Crowder and P. Wolfe: Linear convergence of the conjugate gradient method. *IBM J. Res. Dev.* **16**, pp 431–433, 1972.

[13] C. de Boor: *A practical guide to splines, Revised edition.* Springer Verlag, New York, 2001.

[14] J.E. Dennis, Jr. and R.B. Schnabel: *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice Hall, 1983.

[15] P. Dierckx: *Curve and surface fitting with splines.* Oxford University Press, Oxford, 1995.

[16] I.S. Duff: *The solution of augmented systems.* Rutherford Appleton Laboratory, Report RAL-93-084, 1993.

[17] I.S. Duff, A.M. Erisman and J.K. Reid: *Direct methods for sparse matrices.* Oxford University Press, Oxford, 1986.

[18] L. Eldén, L. Wittmeyer-Koch and H.B. Nielsen: *Introduction to numerical computation – analysis and* MATLAB *illustrations.* Studentlitteratur, Lund, 2004.

[19] R. Fletcher: *Practical methods of optimization, 2nd Edition.* Wiley, 1993.

[20] P.E. Gill and W. Murray: Newton type methods for linearly constrained optimization. In P.E. Gill and W. Murray (eds): *Numerical methods for constrained optimization.* Academic Press, 1974.

[21] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright: Sparse matrix methods in optimization. *SIAM J.S.S.C.* **5**, pp 562–589, 1984.

[22] G. Golub and V. Pereyra: Separable nonlinear least squares: the variable projection method and its applications. *Inverse Problems* **19**, pp R1–R26, 2003.

[23] G. Golub and C.F. Van Loan: *Matrix computations, 3rd edition.* John Hopkins University Press, Baltimore, MD, 1996.

[24] A. Greenbaum: *Iterative methods for solving linear systems.* SIAM, Philadelphia, 1997.

[25] J. Hald and K. Madsen: Combined LP and quasi-Newton methods for minimax optimization. *Mathematical Programming* **20**, pp 49–62, 1981.

[26] J. Hald and K. Madsen: Combined LP and quasi-Newton methods for nonlinear $\ell_1$ optimization. *SIAM J. Numer. Anal.* **22** (1), 1985.

[27] D. Hermey and G.A. Watson: Fitting Data with Errors in all Variables Using the Huber M-Estimator. *SIAM J. Sci. Comput.* **20** (4), pp 1276–1298, 1999.

[28] J.H. Holland: Genetic Algorithms. *Scientific American*, pp 66 – 72, July 1992.

[29] P. Huber: *Robust Statistics.* John Wiley, New York, 1981.

[30] C.L. Lawson and R.J. Hanson: *Solving Least Squares Problems.* Classics in Applied Mathematics, SIAM, Philadelphia, 1995.

[31] K. Levenberg: A Method for the Solution of Certain Problems in Least Squares. *Quart. Appl. Math.* **2**, pp 164–168, 1944.

[32] S.N. Lophaven, H.B. Nielsen and J. Søndergaard: *DACE – a Matlab Kriging toolbox.* Report IMM-REP-2002-12, Informatics and Mathematical Modelling, DTU. (2002), 26 pages. The MATLAB toolbox and report is available from http://www.imm.dtu.dk/∼hbn/dace/

[33] K. Madsen and H.B. Nielsen: Finite Algorrithms for Robust Linear Regression. *BIT* **30**, pp 682–699, 1990.

[34] K. Madsen, H.B. Nielsen: *A Finite Smoothing Algorithm for Linear $\ell_1$ Estimation,* SIAM J. on Optimization **3** (2), 223–235, 1993.

[35] K. Madsen, H.B. Nielsen and J. Søndergaard: *Robust Subroutines for Non-Linear Optimization.* IMM, DTU,
Report IMM-REP-2002-02, 2002. Available as
http://www.imm.dtu.dk/∼km/F_package/robust.pdf

[36] D. Marquardt: An Algorithm for Least Squares Estimation of Nonlinear Parameters. *SIAM J. Appl. Math.* **11**, pp 431–441, 1963.

[37] J.A. Nelder and R. Mead: A Simplex Method for Function Minimization. *Computer J.* **7**, pp 308 – 313, 1965.

[38] H.B. Nielsen: *Implementation of a Finite Algorithm for Linear L1 Estimation.* Institute for Numerical Analysis (now part of IMM), DTU. Report NI 91-01, 49 pages, 1991.

[39] H.B. Nielsen: *Computing a Minimizer of a Piecewise Quadratic – Implementation.* IMM, DTU, Report IMM-REP-1998-14, 31 pages, 1998. Available as http://www.imm.dtu.dk/∼hbn/publ/TR9814.ps

[40] H.B. Nielsen: *Damping Parameter in Marquardt's Method.* IMM, DTU, Report IMM-REP-1999-05, 1999. Available as
http://www.imm.dtu.dk/∼hbn/publ/TR9905.ps

[41] H.B. Nielsen: *Separable Nonlinear Least Squares.* IMM, DTU, Report IMM-REP-2000-01, 2000. Available as
http://www.imm.dtu.dk/∼hbn/publ/TR0001.ps

[42] J. Kowalik and M.R. Osborne: *Methods for Unconstrained Optimization.* Elsevier, 1968.

[43] J. Nocedal: Theory of Algorithms for Unconstrained Optimization. In *Acta Numerica 1992.* Cambridge University Press, 1992

[44] J. Nocedal and S.J. Wright: *Numerical Optimization.* Springer, 1999.

[45] J.A. Payne: An Automatic Curve Fitting Package. Chapter 8 in J.G. Hayes (ed): *Numerical Approximation to Functions and Data.* Athlone Press, London, 1970.

[46] M.J.D. Powell: A Method for Nonlinear Constraints in Minimization Problems. In R. Fletcher (ed): *Optimization.* Academic Press, 1969.

[47] M.J.D. Powell: Curve Fitting by Splines in one Variable. Chapter 6 in J.G. Hayes (ed): *Numerical Approximation to Functions and Data.* Athlone Press, London, 1970.

[48] M.J.D. Powell: A Hybrid Method for Non-Linear Equations. pp 87ff in P. Rabinowitz (ed): *Numerical Methods for Non-Linear Algebraic Equations.* Gordon and Breach, 1970.

[49] J.K. Reid: Implicit Scaling of Linear Least Squares Problems. *BIT* **40** (1), pp 146–157, 2000.

[50] J. Sacks, W.J. Welch, T.J. Mitchell and H.P. Wynn: Design and Analysis of Computer Experiments. *Statistical Science* **4**(4), pp 409–435, 1989.

[51] P.L. Toint: On Large Scale Nonlinear Least Squares Calculations. *SIAM J.S.S.C.* **8**, pp 416–435, 1987.

[52] V. Torczon: *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines.* PhD Thesis, Rice University, Houston, Texas, 1990. Available from http://www.cs.wm.edu/∼va/research/

[53] V. Torczon: Why Pattern Search Works. *OPTIMA* **59**, pp 1 – 7, 1998.

[54] H.A. van der Vorst: *Iterative Krylov Methods for Large Linear Systems.* Cambridge University Press, 2003

[55] M. Zelen: Linear Estimation and Related Topics. pp 558–584 in J. Todd (ed): *Survey of Numerical Analysis.* McGraw-Hill, New York, 1962.

# Index