Markus Grasmair

Continuous Optimisation

Lecture Notes
Summer 2012

Computational Science Center
University of Vienna
A-1090 Vienna, Austria

Preface

These notes are mostly based on the following book and lecture notes:

- J. Frédéric Bonnans, J. Charles Gilbert, Claude Lemaréchal, and Claudia A. Sagastizábal, *Numerical Optimization*, Springer, Berlin, 2nd edition, 2006.
- Otmar Scherzer and Frank Lenzen, *Optimierung*, Vorlesungsskriptum WS 2008/09, University of Innsbruck, Austria, 2009.

The chapter on the conjugate gradient method in addition uses:

- Otmar Scherzer, *Numerische Mathematik*, Vorlesungsskriptum SS 2010, University of Vienna, Austria 2010.
- Jonathan Richard Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, Edition $1\frac{1}{4}$, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

The section on the Levenberg–Marquardt method uses some of the results in:

- Barbara Kaltenbacher, Andreas Neubauer, and Otmar Scherzer, *Iterative Regularization Methods for Nonlinear Ill-posed Problems*, de Gruyter, Berlin, 2008.

The chapter on interior point methods is based on:

- James Renegar, *A Mathematical View of Interior-Point Methods in Convex Optimization*, MPS/SIAM Series on Optimization, SIAM, Philadelphia (PA), 2001.

Contents

List of Algorithms

List of Figures

Chapter 1

Basic Ideas and Concepts

1.1 General Situation

The main goal of this lecture is the presentation of different methods for the minimisation of a given function $f\colon \mathbb{R}^n \to \mathbb{R}$, possibly subject to additional constraints.

Free Optimisation

Here we assume that we are given $f\colon \mathbb{R}^n \to \mathbb{R}$. The task is to find

$$x^* \in \mathbb{R}^n \qquad \text{such that } f(x^*) \text{ is minimal.}$$

This is the problem we will be interested in in the first half of the lecture before turning to the vastly more complicated situation of constrained optimisation problems. In addition, we assume for the most part of the lecture that the function f we want to minimise is differentiable or even two times differentiable. Then, instead of minimising f, we may try to find stationary points of f, that is, points where the derivative of f vanishes. Thus, we replace the minimisation problem by the problem of solving an equation, which in many cases is a task that is easier to approach.

Constrained Optimisation

The minimisation problem becomes more difficult, if the minimum is required to lie in a specific subset of $\mathbb{R}^n$ described by some equality and inequality constraints. That is, find

$$x^* \in \arg\min\bigl\{f(x) : x \in C\bigr\} \qquad \text{where } x \in C :\Longleftrightarrow \begin{cases} c_j(x) \le 0\,, & j \in I\,, \\ c_j(x) = 0\,, & j \in E\,, \end{cases}$$

for some functions $c_j\colon \mathbb{R}^n \to \mathbb{R}$, $j \in I$ (inequality constraints), and $c_j\colon \mathbb{R}^n \to \mathbb{R}$, $j \in E$ (equality constraints).

Additional Complications

In practise, there arise several complications we have to keep in mind:

For one, we often do not have direct access to the function f. Instead, we only have a black-box that takes the argument x and returns the value $f(x)$, hopefully also the derivative of f at x, and, if we are very lucky, the second order derivative. In addition, the evaluation of f and its derivatives can be very time consuming. Then it is necessary to use sophisticated methods that can find the minimum (or a good approximation) with as few as possible calls of the function f.

A second problem is that the number of unknowns can be huge. In a moderately sized problem we may easily have a few thousand unknowns, but also problems with several million unknowns appear in practical applications. Then already the storage of all the necessary data may pose severe problems—for instance the Hessian, which is required in the (standard) Newton method would be a $10^6 \times 10^6$ matrix.

Finally, in many practical applications one not really wants to obtain an optimum, but rather a near optimum that is stable with respect to small perturbations. This can be due to the fact that the function f to be optimised arises out of some modelling of the real world, which cannot be done exactly; it is almost certain that some modelling errors have occurred. In addition, all the numerical calculations (at least in *continuous* optimisation) are not performed exactly but only within a certain accuracy. If the minimum one obtains is very sensitive with respect to these errors, then this numerical optimum might be quite different from the true optimum one would want to obtain. As a remedy, one can either try to adapt the optimisation method in such a way that the iterations will be comparably stable and then end the iteration prematurely, or modify (regularise) the function f in order to obtain more stable (but still useful) results. This last problem and approaches to its solution will not be dealt with during this lecture.

1.2 Optimality Conditions

We now introduce some (standard) mathematical definitions that will be required throughout these notes.

We denote by $C^k(\mathbb{R}^n)$ the space of k-times continuously differentiable functions on $\mathbb{R}^n$. If $f \in C^1(\mathbb{R}^n)$, we denote by $\nabla f\colon \mathbb{R}^n \to \mathbb{R}^n$ the *gradient* of f, defined by

$$\nabla f(x) := \big(\partial_i f(x)\big)_{1\le i\le n} = \begin{pmatrix} \dfrac{\partial f}{\partial_{x_1}}(x) \\ \vdots \\ \dfrac{\partial f}{\partial_{x_n}}(x) \end{pmatrix}.$$

If $f \in C^2(\mathbb{R}^n)$, we denote by $H_f\colon \mathbb{R}^n \to \mathbb{R}^{n\times n}$ the *Hessian* of f, defined by

$$H_f(x) := \big(\partial^2_{ij} f(x)\big)_{1\le i,j\le n} = \begin{pmatrix} \dfrac{\partial^2 f}{\partial_{x_1}\partial_{x_1}}(x) & \dots & \dfrac{\partial^2 f}{\partial_{x_1}\partial_{x_n}}(x) \\ \vdots & & \vdots \\ \dfrac{\partial^2 f}{\partial_{x_n}\partial_{x_1}}(x) & \dots & \dfrac{\partial^2 f}{\partial_{x_n}\partial_{x_n}}(x) \end{pmatrix}.$$

Whenever $f \in C^2(\mathbb{R}^n)$, the Hessian is a symmetric matrix, that is, $H_f(x)^T = H_f(x)$.

Recall that, given $\hat{x} \in \mathbb{R}^n$, the gradient $\nabla f(\hat{x})$ provides the best approximation of the function f near $\hat{x}$ by a linear function. That is, the mapping

$$x \mapsto f(\hat{x}) + \nabla f(\hat{x})\,(x - \hat{x})$$

is the linear (more accurately: affine) function that yields the best description of f locally around the point $\hat{x}$. Similarly, the gradient and the Hessian together provide the best quadratic approximation of f via the mapping

$$x \mapsto f(\hat{x}) + \nabla f(\hat{x})\,(x - \hat{x}) + \frac{1}{2}(x - \hat{x})^T H_f(\hat{x})\,(x - \hat{x})\,.$$

If $A \in \mathbb{R}^{n\times n}$ is a symmetric matrix, then we say that $A \geq 0$ or that A is *positive semi-definite*, if $x^T Ax \geq 0$ for every $x \in \mathbb{R}^n$. One can show that $A \geq 0$, if and only if all eigenvalues of A are non-negative. In addition, we say that $A \geq B$, if $(A - B) \geq 0$, or, equivalently, $x^T Ax \geq x^T Bx$ for every $x \in \mathbb{R}^n$.

Similarly, we say that $A > 0$ or that A is *positive definite*, if $x^T Ax > 0$ for every $x \in \mathbb{R}^n \setminus \{0\}$, and $A > B$, if $(A-B) > 0$. Again, one can show that $A > 0$, if and only if all eigenvalues of A are positive (that is, *strictly* larger than zero).

Let $f \in C^1(\mathbb{R}^n)$ and $x^* \in \mathbb{R}^n$. We say that x^* is a *stationary point* of f (or *critical point*), if $\nabla f(x^*) = 0$. We say that x^* is a *local minimum* of f, if there exists some $\varepsilon > 0$ such that $f(x^*) \leq f(x)$ for every $x \in \mathbb{R}^n$ with $\|x^* - x\| < \varepsilon$. In case $f(x^*) < f(x)$ for every $x \in \mathbb{R}^n \setminus \{x^*\}$ with $\|x^* - x\| < \varepsilon$ we say that x^* is a *strict local minimum*.

Proposition 1.2.1 (Necessary Conditions). *Assume that x^* is a local minimum of the function $f\colon \mathbb{R}^n \to \mathbb{R}$.*

1. *If $f \in C^1(\mathbb{R}^n)$, then x^* is a stationary point of f.*
2. *If $f \in C^2(\mathbb{R}^n)$, then additionally $H_f(x^*) \geq 0$.*

Proposition 1.2.2 (Sufficient Conditions). *Assume that $f \in C^2(\mathbb{R}^n)$ and that x^* is a stationary point of f. If $H_f(x^*) > 0$, then x^* is a local minimum of f.*

Most algorithms in continuous optimisation do not aim for the actual minimisation of the cost function f, but instead only search for stationary points x^*, possibly with the additional restriction that $H_f(x^*) \geq 0$ (note, however, that the latter is only possible, if we have access to the Hessian). More precisely, the result of most algorithms to be presented in this lecture is a point x^*_ε for which $\|\nabla f(x^*_\varepsilon)\| < \varepsilon$, where $\varepsilon > 0$ is some threshold either specified by the user or the algorithm.

Remark 1.2.3. There is a slight difference between the necessary condition and the sufficient condition for local minima: The former states that $H_f(x^*) \geq 0$, while in the latter we require strict positivity of $H_f(x^*)$. Consider for instance the function $f(x) := 2x^6 - 3x^4$, which has a local maximum at $x^* = 0$ although $f'(0) = 0$ and $f''(0) = 0$ (cf. Figure 1.1, left). Similarly, the function $f(x, y) := x^2 - y^4$ has a stationary point at $(x^*, y^*) = (0, 0)$ and $H_f(x^*, y^*) = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix} \geq 0$. Still, the point (x^*, y^*) is no local minimum (cf. Figure 1.1, right). In practical applications, however, this will hardly every make any difference. ∎

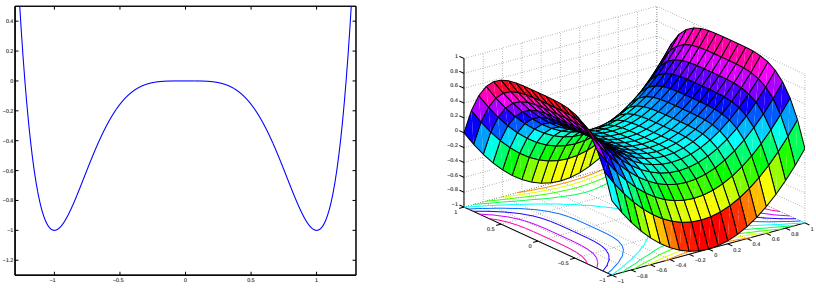

Figure 1.1: Stationary points with positive semi-definite Hessian that are no local minima.

Minima of Convex Functions

There is one important class of functions, where the necessary and sufficient conditions for a minimiser are equivalent: convex functions. We will now briefly recall the most important properties of convex functions that are relevant for optimisation.

A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *convex*, if

$$f\big(\lambda x + (1-\lambda)y\big) \leq \lambda f(x) + (1-\lambda) f(y) \qquad \text{for all } x,\, y \in \mathbb{R}^n \text{ and } 0 < \lambda < 1\,.$$

Proposition 1.2.4. *Assume that $f \in C^1(\mathbb{R}^n)$ is convex. Then x^* is a* global *minimum of f, if and only if x^* is a stationary point of f.*

A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *strictly convex*, if

$$f\big(\lambda x + (1-\lambda)y\big) < \lambda f(x) + (1-\lambda) f(y) \qquad \text{for all } x \neq y \in \mathbb{R}^n \text{ and } 0 < \lambda < 1\,.$$

A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *uniformly convex*, if there exists $c > 0$ such that

$$f\big(\lambda x + (1-\lambda)y\big) + c\|x-y\|^2 \lambda(1-\lambda) \leq \lambda f(x) + (1-\lambda) f(y)$$

for all x, $y \in \mathbb{R}^n$, and $0 < \lambda < 1$.

Let $f \in C^2(\mathbb{R}^n)$ and $x \in \mathbb{R}^n$. The function f is *locally elliptic*, if $H_f(x) > 0$ for every $x \in \mathbb{R}^n$.

Proposition 1.2.5. *Let $f \in C^2(\mathbb{R}^n)$.*

- *The function f is convex, if and only if $H_f(x) \geq 0$ for every $x \in \mathbb{R}^n$.*
- *If f is locally elliptic, then f is strictly convex.*
- *The function f is uniformly convex, if and only if there exists $c > 0$ such that $H_f(x) \geq c\,\mathrm{Id}$ for every $x \in \mathbb{R}^n$.*

Proposition 1.2.6. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be convex.*

- *Every local minimum of f is already a global minimum.*
- *If f is strictly convex, then it has at most one minimum.*
- *If f is uniformly convex, then it has precisely one minimum.*

1.3 Convergence

In this lecture we will always face the following situation: The cost function f to be minimised is fixed and we possibly have also access to its gradient and Hessian. We will discuss different algorithms $\mathcal{A}$ that receive as input some initial guess of the solution x_{init} and a set of parameters $\mathcal{P}$, and return a sequence

$$x^{(k)} := \mathcal{A}(k, x_{\text{init}}, \mathcal{P}) \in \mathbb{R}^n \,,$$

which should hopefully approximate the solution (or critical point!) $x^* \in \mathbb{R}^n$ as the iteration number k tends to infinity.

Definition 1.3.1. We say that the algorithm $\mathcal{A}$ is *globally convergent* (to x^*), if, for some suitably chosen set of parameters $\mathcal{P}$, the sequence $x^{(k)}$ converges to x^* *independent of the initial guess* x_{init}.

We say that the algorithm is *locally convergent* (to x^*) or, simply, *convergent*, if there exists some $\varepsilon > 0$ such that, for some suitably chosen set of parameters $\mathcal{P}$, the sequence $x^{(k)}$ converges to x^* *whenever the initial guess satisfies* $\|x_{\text{init}} - x^*\| < \varepsilon$. ■

Obviously, globally convergent algorithms are preferable, because usually we cannot guarantee that the initial guess is already close to the true solution. In addition, in case of a locally convergent algorithm, the set of initial guesses that lead to a convergent sequence may be arbitrarily small.

After having shown that some algorithm is convergent, one may ask the question, how fast the iterates $x^{(k)}$ converge to x^*. There are two related approaches to the estimation of convergence rates: Q-convergence and R-convergence.

Definition 1.3.2. Assume that $(x^{(k)})_k$ converges to x^* and $x^* \neq x^{(k)}$ for every k. We define for $1 \leq p < \infty$ the *p-quotient convergence factor* or *Q-factor* of the sequence $(x^{(k)})_k$ as

$$Q_p := Q_p\big((x^{(k)})_k\big) := \limsup_{k\to\infty} \frac{\|x^{(k+1)} - x^*\|}{\|x^{(k)} - x^*\|^p} \,.$$

The number

$$\mathcal{O}_Q := \mathcal{O}_Q\big((x^{(k)})_k\big) := \inf\big\{p \geq 1 : Q_p\big((x^{(k)})_k\big) = +\infty\big\}$$

is called the *Q-convergence order* of the sequence $(x^{(k)})_k$. ■

Remark 1.3.3. The Q-factor depends on the actual choice of the norm $\|\cdot\|$ on $\mathbb{R}^n$, while the Q-convergence order does not. Note that in some cases, it may make sense to replace the Euclidean norm on $\mathbb{R}^n$ by a different one, for instance the 1-norm $\|x\|_1 := \sum_i |x_i|$ or the ∞-norm $\|x\|_\infty := \max_i |x_i|$. Also, in some cases the analysis can become easier (and more natural), if one applies a weighted Euclidean norm of the form $\|x\|_A := \sqrt{x^T A x}$, where $A \in \mathbb{R}^{n\times n}$ is symmetric and positive definite. This happens for instance in the study of conjugate gradient methods. ■

Remark 1.3.4. There exists some $p_0 \geq 1$ such that $Q_p\big((x^{(k)})_k\big) = 0$ for all $1 \leq p < p_0$ and $Q_p\big((x^{(k)})_k\big) = +\infty$ for all $p_0 < p < +\infty$. ■

Of particular importance are the convergence orders 1 and 2:

1. The convergence of an algorithm $\mathcal{A}$ is
   - *super-linear* if $Q_1 = 0$,
   - *linear* if $0 < Q_1 < 1$,
   - *sub-linear* if $Q_1 \geq 1$.
2. The convergence of an algorithm $\mathcal{A}$ is
   - *super-quadratic* if $Q_2 = 0$,
   - *quadratic* if $0 < Q_2 < +\infty$,
   - *sub-quadratic* if $Q_2 = +\infty$ and $\mathcal{O}_Q = 2$.

Lemma 1.3.5. *Assume that the sequence $(x^{(k)})_k$ converges Q-linearly to x^*. Then, for every $Q_1 < q < 1$, there exists a constant $C > 0$ and an index k_0 such that*

$$\|x^{(k)} - x^*\| \leq Cq^k \qquad \text{for every } k \geq k_0 \, .$$

This means that the error of a linear convergent algorithm decreases by a constant factor $q < 1$ in each step. Put differently, the number of correct digits of the solution increases by a constant number in each step. Similarly, in case of quadratic convergence, we can say that the number of correct digits roughly doubles in each iteration.

A different approach for the measurement of the accuracy of an algorithm is that of R-convergence:

Definition 1.3.6. Assume that the sequence $(x^{(k)})_k$ converges to x^*.

We define for $1 \leq p < \infty$ the *p-root factor* or R-factor of the sequence $(x^{(k)})_k$ as

$$R_p := R_p\big((x^{(k)})_k\big) := \begin{cases} \limsup\limits_{k\to\infty} \|x^{(k)} - x^*\|^{1/k} & \text{if } p = 1 \, , \\ \limsup\limits_{k\to\infty} \|x^{(k)} - x^*\|^{1/p^k} & \text{if } p > 1 \, . \end{cases}$$

The number

$$\mathcal{O}_R := \mathcal{O}_R\big((x^{(k)})_k\big) := \inf\big\{p \geq 1 : R_p\big((x^{(k)})_k\big) = 1\big\}$$

is called the *R-convergence order* of the sequence $(x^{(k)})_k$. ∎

Here for $\log R_p < C < 0$ one has

$$\|x^{(k)} - x^*\| \leq \exp(Cp^k)$$

for $p > 1$ and k sufficiently large, or

$$\|x^{(k)} - x^*\| \leq \exp(Ck)$$

for k sufficiently large and $p = 1$.

Note the conceptual difference between Q-convergence and R-convergence. The former measures the improvement of the solution in each step, while the latter measures the quality of the approximation directly. It is in principle possible that a sequence has a very good convergence behaviour with respect to R-convergence, while its Q-convergence is only sublinear.

1.4 Steepest Descent

As indicated in Section 1.2, the goal of most optimisation algorithms is the search for stationary points, that is, the solution of the equation

$$\nabla f(x) = 0 \,. \tag{1.1}$$

Among the simplest methods for the solution of the (usually non-linear!) equation (1.1) are fixed point iterations, which, given an initial guess $x^{(1)}$ and a step size $t \in \mathbb{R}$ compute

$$x^{(k+1)} := x^{(k)} - t\nabla f(x^{(k)}) \,. \tag{1.2}$$

Lemma 1.4.1. *Assume that the function $f \in C^2(\mathbb{R}^n)$ is locally elliptic and the equation* (1.1) *has a solution. Then the iteration defined in* (1.2) *converges Q-linearly, if $t > 0$ is sufficiently close to zero.*

Example 1.4.2. Consider the (trivial) example, where $f\colon \mathbb{R}^n \to \mathbb{R}$ is defined as $f(x) := \frac{1}{2}x^T Ax$ for some positive definite, symmetric matrix $A \in \mathbb{R}^{n\times n}$. Then $\nabla f(x) = Ax$ for all $x \in \mathbb{R}^n$. Thus, the iteration (1.2) reads as

$$x^{(k+1)} = x^{(k)} - tAx^{(k)} = \ldots = \big(I - tA\big)^{(k)} x^{(1)} \,.$$

Thus the sequence $x^{(k+1)}$ converges to zero, the unique minimiser of f, provided that $\|I - tA\| < 1$, where $\|\cdot\|$ denotes any matrix norm on $\mathbb{R}^{n\times n}$. In addition, one has the estimate

$$\|x^{(k+1)}\| \le \|I - tA\|^k \|x^{(1)}\| \tag{1.3}$$

for all k.

Consider for instance the choice $\|\cdot\| := \|\cdot\|_2$, the *spectral norm*, which is defined as

$$\|B\|_2 := \max\big\{\sigma : \sigma \text{ is a singular value of } B\big\} \,.$$

Then the estimate for the norm of x^{k+1} is sharp in the sense that there exists an initial vector $x^{(1)}$ for which the estimate (1.3) holds with equality. Thus, in the worst case, the fastest (though still linear) convergence is obtained if the step length $t > 0$ is chosen such that the norm $\|I - tA\|_2$ is minimal.

Now recall that the singular values of a matrix B are the eigenvalues of the matrix B^*B and, therefore, non-negative real numbers. In the particular case of a symmetric matrix B, which we consider here, the singular values of the matrix coincide with the absolute values of its eigenvalues. Moreover, if $\lambda_1 \ge \ldots \ge \lambda_n$ are the eigenvalues of A, then $1 - t\lambda_1, \ldots, 1 - t\lambda_n$ are the eigenvalues of $I - tA$. This shows that

$$\|I - tA\|_2 = \max\big\{|1 - t\lambda_1|, |1 - t\lambda_n|\big\} \,.$$

Minimising this expression for t, we see that the minimal value is obtained for $t^* := 2/(\lambda_1 + \lambda_n)$, in which case

$$\|I - t^*A\|_2 = \frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n} = \frac{1 - \frac{\lambda_n}{\lambda_1}}{1 + \frac{\lambda_n}{\lambda_1}} < 1 \,.$$

This value is small, if all the eigenvalues of A are of approximately the same size, while it becomes close to 1, as the ratio between smallest and largest eigenvalue of A is close to zero, that is, if the matrix A is ill-conditioned.

For a concrete example showing the behaviour of the iterates $x^{(k)}$ see Figure 1.2. ∎

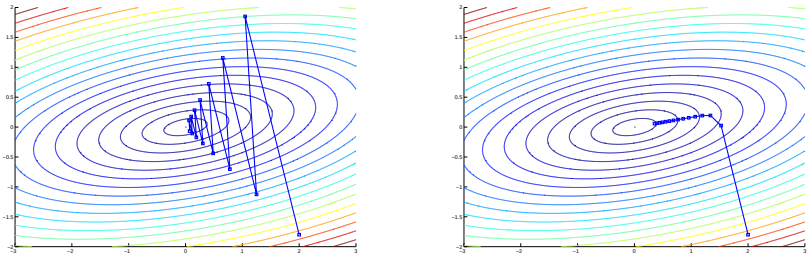

Figure 1.2: Contour plot of the function f defined in Example 1.4.2 and the result of the first 15 iterations of the steepest descent method (1.2) with $A = \left(\begin{smallmatrix} 1 & -1 \\ -1 & 7 \end{smallmatrix}\right)$, $x^{(1)} = (2, -1.8)$, and $t = 0.25$ (left hand side) and $t = 0.125$ (right hand side).

One major reason, why the fixed point iteration (1.2) performs that badly, is that it completely neglects the fact that our main goal is the minimisation of the function f. An obvious point, where this additional information could be used, is the choice of the step size. First, instead of having a fixed step size, we may decide to choose different step sizes $t^{(k)}$ in each iteration. Then it makes sense to choose the step size $t^{(k)}$ in such a way that the function value at the next iterate becomes minimal, that is, $f\big(x^{(k)} - t\nabla f(x^{(k)})\big) \to \min$. If we do so, then we end up with the steepest descent method described in Algorithm 1. Note, however, that, usually, we cannot compute the minimum of f along the line in direction $\nabla f(x^{(k)})$ exactly. Instead, we have to content ourselves with finding sufficiently good approximations of the optimal step size. Methods for finding such approximations will be discussed in Chapter 2.

Data: a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$ and its gradient ∇f;
an initial guess x_{init};
Result: $x^* \in \mathbb{R}^n$;

Initialisation: set $x^{(1)} := x_{\text{init}}$, $k = 1$;

while *convergence criterion not yet satisfied* **do**
  compute $d^{(k)} := -\nabla f(x^{(k)})$;
  compute $t^{(k)} := \arg\min\big\{ f(x^{(k)} + td^{(k)}) : t \geq 0 \big\}$;
  define $x^{(k+1)} := x^{(k)} + t^{(k)} d^{(k)}$;
  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$;

Algorithm 1: Steepest descent with exact line search.

At this point, one might ask the question, why we should insist on choosing the updates precisely in direction of the (negative) gradient. Indeed, although there is some motivation behind this choice apart from the discussion above, if we have better search directions available, we should make use of them. If we do so, we end up with Algorithm 2, which is the basic structure of many optimisation methods.

Data: a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$;
an initial guess x_{init};
Result: $x^* \in \mathbb{R}^n$;
Initialisation: set $x^{(1)} := x_{\text{init}}$, $k = 1$;
while *convergence criterion not yet satisfied;*
do
  find a search direction $d^{(k)} \in \mathbb{R}^n$;
  compute a reasonable step size $t^{(k)} \in \mathbb{R}$;
  define $x^{(k+1)} := x^{(k)} + t^{(k)} d^{(k)}$;
  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$;

Algorithm 2: Basic setup of a minimisation algorithm.

As indicated above, there is one reason apart from the motivation by the fixed point iteration why the choice of the gradient as a descent direction makes sense. By definition of the gradient, we have, for $t \in \mathbb{R}$ and $d \in \mathbb{R}^n$ small enough,

$$f(x + td) \approx f(x) + t\langle \nabla f(x), d\rangle \ .$$

That is, for d sufficiently small, say $\|d\| \leq \delta$, the decrease of the function f in direction d is approximated by the inner product $\langle \nabla f(x), d\rangle$. Because we aim for the minimisation of f, one idea is therefore to find $d \in \mathbb{R}$ such that

$$\langle \nabla f(x), d\rangle \to \min \qquad \text{subject to } \|d\| \leq \delta \ . \tag{1.4}$$

If the norm $\|\cdot\|$ is the Euclidean norm, then the solution of this minimisation problem is the negative gradient of f (or, to be precise, $d = -\delta \nabla f(x)/\|\nabla f(x)\|$, but at that point we are only interested in finding a direction). Note, however, that different choices of the norm in (1.4) lead to other optimal descent directions. For instance, if we choose $\|d\| := \|d\|_1 = \sum_i |d_i|$, then

$$d = \pm e_i \qquad \text{with} \qquad i \in \arg\max\big\{|\partial_j f(x)| : 1 \leq j \leq n\big\} \ ,$$

and the sign of d depends on the sign of $\partial_i f(x)$.

Chapter 2

Line Search

2.1 General Scheme

In this chapter, we assume that we are already given a search direction $d \in \mathbb{R}^n$ and our task is to find a good step size t. We can therefore say that we want to minimise the function

$$t \mapsto g(t) := f(x + td) \ .$$

Here it is necessary to stress again the additional complications involved in this task. Most of all, we should keep in mind that we do not have any analytical formula for the function g. All that we usually have is a black box that takes the value $x + td$ and returns the value $g(t) = f(x + td)$ and, possibly, also the derivative $g'(t) = \langle \nabla f(x + td), d \rangle$. Thus, trial and error based on these values is the only viable strategy.

Simply speaking, all line search algorithms consist of two sub-algorithms: First, an algorithm that, given the values $t > 0$, $g(t)$, and, possibly, $g'(t)$, decides whether the step size t is too large, too small, or acceptable. Second, an algorithm that computes a new candidate for the step size if the former candidate has been rejected. Thus, on a very basic level, line search algorithms read as the one summarised in Algorithm 3. The question is now, how to construct the two sub-algorithms.

Initialisation: set $t_L = 0$ and $t_R = +\infty$;
choose some initial $t > 0$;
while t *not satisfactory* **do**
  if t *is too small* **then**
    $t_L \leftarrow t$;
  else
    $t_R \leftarrow t$;
  end
  compute new $t \in (t_L, t_R)$;
end
define $t^* := t$;

Algorithm 3: Sketch of a line search algorithm.

First we note that the classification sub-algorithm has to satisfy at least the following properties.

1. For each $t > 0$, either t is classified as too small, or t is classified as too large, or t is accepted.
2. There is some upper bound $t_{\max}$ such that every $t > t_{\max}$ is classified as too large. Thus it cannot happen that the step size increases to $+\infty$ and the line search fails to terminate (note that the upper bound need not be given explicitly).
3. Whenever t_L is classified as too small and t_R is classified as too large, there exists a non-empty open interval $I \subset [t_L, t_R]$ such that *every* element in I is classified as satisfactory (thus the algorithm has a chance to terminate for suitable updates of t).

Note that these three properties do not imply that the result will be a good choice for a step size; they are merely required for obtaining any result at all.

Example 2.1.1. An easy example of a classification sub-algorithm that satisfies these three properties and is not completely unreasonable is the following:
Let $\varepsilon > 0$ be fixed. We say that

- t is too small, if $g'(t) < -\varepsilon$,
- t is too large, if $g'(t) > +\varepsilon$,
- t is acceptable, if $|g'(t)| \leq \varepsilon$.

This method works, provided that $g'(t) \to \infty$ as $t \to \infty$ (which is an assumption often justified in applications). Also, at first glance, it looks at least reasonable: if $g'(t) < 0$, then we know that the value of g will decrease if we slightly increase g. Thus, indeed, t may be deemed as too small. Similarly, if $g'(t) > 0$, then we can decrease the value of g by decreasing t; thus t is probably too large. In addition, we ensure the termination of the algorithm, because we stop the iteration already when g' is very close to zero, but not necessarily precisely equal to zero.

At second glance, however, one might notice one major flaw of the method: We never check, whether we have actually decreased the function value at all! Indeed, with this method it is quite likely that the result will be a local minimum whose value is larger then the value at the point we have started with. Thus, it is (slightly) better to use a refined method, which also checks whether we have decreased g. We might therefore say that

- t is too small, if $g(t) < g(0)$ and $g'(t) < -\varepsilon$,
- t is too large, if $g(t) \geq g(0)$ or $g'(t) > +\varepsilon$,
- t is acceptable, if $g(t) < g(0)$ and $|g(t)| \leq \varepsilon$. ∎

Although the method described in Example 2.1.1 indeed yields a result that is smaller than the current function value and probably close to a local minimum of g, there are several good reasons, why it should not be used in practise. The most important reason is: We do not actually want to minimise the function

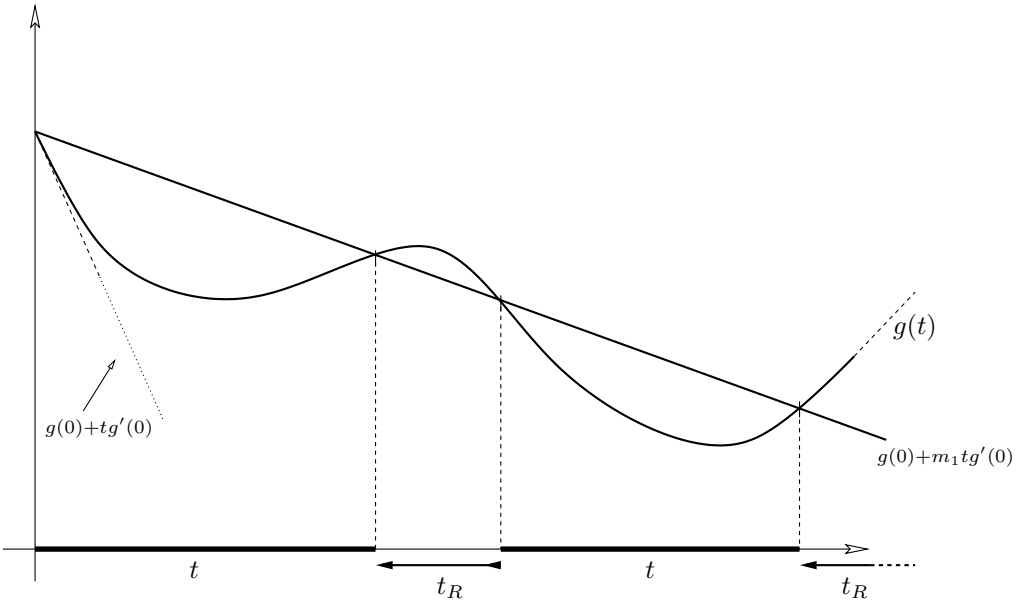

Figure 2.1: Armijo's rule. A step size is declared too large, if the actual decrease of the function value is much smaller than the predicted decrease.

g. Instead, our goal is the minimisation of f, and the reason why we consider the minimisation of g at all is that we want to find a good step size in order to accelerate the convergence of the algorithm as a whole. Thus we should not spend too much time within the line search.

2.2 Choice of the Step Size

In the following, we will always assume that $g'(0) < 0$. This assumption is reasonable in all applications, because it simply means that d is a descent direction.

2.2.1 Armijo

One main criterion in all modern line search algorithms is that the actual decrease of the function g is (at least) of the same order as the expected decrease. The expected decrease for a step size $t > 0$ is given by the derivative of g at zero, multiplied by t. Thus we should consider a step size too large, if the difference $g(t) - g(0)$ is much larger than $tg'(0)$ (note that $g'(0)$ is assumed to be negative!).

In practise, this means that we choose some $0 < m_1 < 1$ and say that a step size t is too large, if

$$g(t) > g(0) + m_1 t g'(0) \; . \tag{2.1}$$

The condition (2.1) is called *Armijo's rule*. For an illustration see Figure 2.1. In principle, this condition alone can already be used for the classification step in a line search algorithm. Then we end up with the simple Algorithm 4.

Initialisation: choose some $t > 0$ and $0 < m_1 < 1$;
while $g(t) > g(0) + m_1 t g'(0)$ **do**
| decrease t;
end
define $t^* := t$;

Algorithm 4: Line search with Armijo's rule.

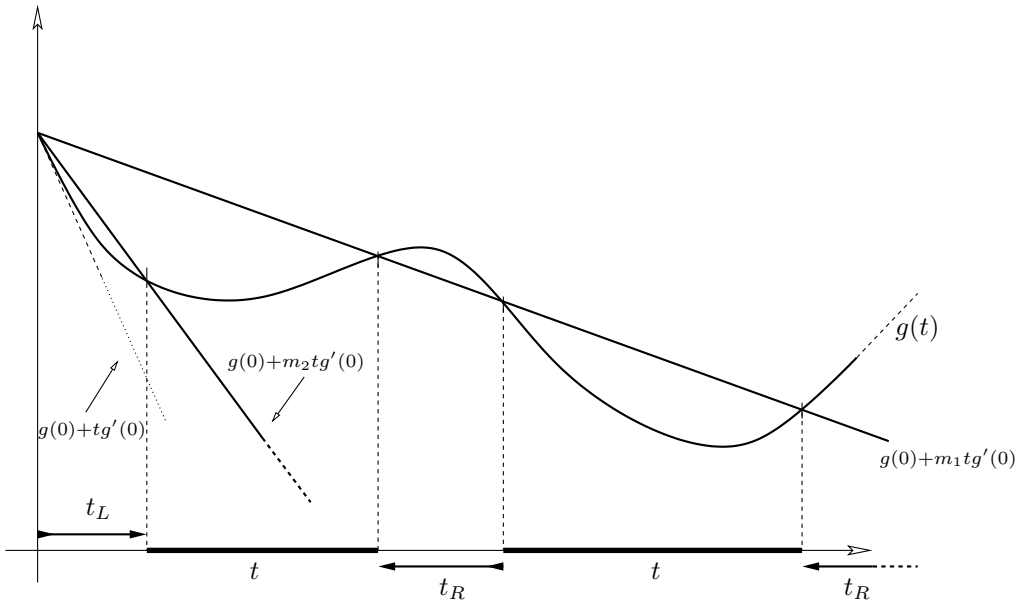

Figure 2.2: Goldstein and Price line search. In addition to Armijo's rule, a step size is declared too small, if the actual decrease of the function value is not much smaller (or even larger) than the predicted decrease.

The usage of Armijo's rule alone can be dangerous, because it never declares a step size to be too small. Thus a good (meaning: sufficiently large) initialisation of t at the beginning of the line search is extremely important; else it is possible that the number of iterations of the algorithm is exceedingly large. Typically, a constant initialisation (say, $t = 1$) is chosen. However, this only works, if we either have a good understanding of the function we want to optimise, or the algorithm that determines the search direction at the same time yields a step length. This is for instance the case in the Newton method and its derivatives, where the step length $t = 1$ is asymptotically optimal and the main task of the line search is to increase the region of convergence of the method.

2.2.2 Goldstein and Price

The second classification sub-algorithm we discuss is based on Armijo's rule, but, in addition, introduces a criterion that decides whether a step size is too small. Again this criterion compares the actual decrease with the expected decrease. The difference is now that we declare the step size too small if the actual decrease is not much smaller than the expected one. The idea is that, in

this case, it should be possible to decrease the value of g further by increasing t.

In practise, this means that we choose two numbers $0 < m_1 < m_2 < 1$ and say that:

- t is too large if $g(t) > g(0) + m_1 t g'(0)$,
- t is too small if $g(t) < g(0) + m_2 t g'(0)$,
- t is acceptable, if

$$m_2 g'(0) \leq \frac{g(t) - g(0)}{t} \leq m_1 g'(0) \ .$$

These three conditions are called the rule of *Goldstein and Price*. An interpretation of these condition is shown in Figure 2.2. The method is summarised in Algorithm 5.

Initialisation: set $t_L = 0$ and $t_R = +\infty$;
choose some initial $t > 0$;
declare t inacceptable;
fix $0 < m_1 < m_2 < 1$;
while t *is inacceptable* **do**
  if $g(t) > g(0) + m_1 t g'(0)$ **then**
    set $t_R \leftarrow t$;
    choose new $t \in (t_L, t_R)$;
  else if $g(t) < g(0) + m_2 t g'(0)$ **then**
    set $t_L \leftarrow t$;
    choose new $t \in (t_L, t_R)$;
  else
    declare t acceptable;
  end
end
define $t^* := t$;

Algorithm 5: Line search according to Goldstein and Price.

2.2.3 Wolfe

The two methods discussed above only use the function values $g(0)$ and $g(t)$, as well as the derivative $g'(0)$ for determining the step length, but not the derivative of g at other points. It is reasonable to assume that the additional usage of gradient information may lead to better results of the line search provided that the cost of computing derivatives is not too large. We will, however, still base the decision whether a step size is too large on Armijo's rule and only use the gradient information for declaring step sizes too small.

Now, the seemingly obvious possibility of declaring a step size t too small, if the derivative $g'(t)$ is below some negative threshold close to zero, cannot be used, as it may lead in some situations to an impossibility of finding any acceptable step size at all. We cannot guarantee that there is a value t^* satisfying Armijo's rule such that $g'(t^*) \approx 0$ (see Figure 2.3). Because of Armijo's rule

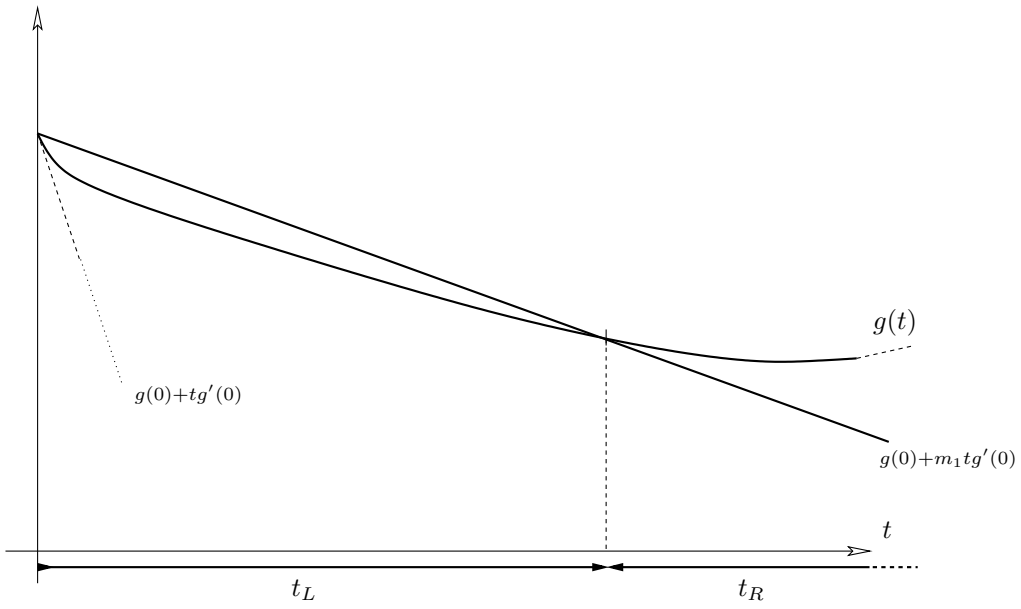

Figure 2.3: Impossibility of basing a line search on Armijo's rule and smallness of the derivative.

$g(t) \leq g(0) + m_1 t g'(0)$, we only obtain that all the values between $g'(0)$ and $m_1 g'(0)$ are attained. Indeed, if $\hat{t}$ denotes the smallest positive value for which Armijo's rule is satisfied exactly, that is, $g(\hat{t}) = g(0)+m_1 t g'(0)$, then, necessarily, $g'(\hat{t}) \geq m_1 g'(0) > g'(0)$. Moreover, the mean value theorem implies that g' attains, in the interval $(0, \hat{t})$, all values in the range $(g'(0), g'(\hat{t}))$.

The argumentation above implies that we can only obtain a convergent line search algorithm with Armijo's rule and gradient information, if we declare a step length t acceptable, if $g'(t) \geq m_1 g'(0)$. One possibility is to call a step size t too small, if the derivative $g'(t)$ is smaller than $m_2 g'(0)$, where m_2 is larger than the constant in Armijo's rule but still smaller than 1.

That is, we choose two numbers $0 < m_1 < m_2 < 1$ and say that:

- t is too large if $g(t) > g(0) + m_1 t g'(0)$,
- t is too small if $g(t) \leq g(0) + m_1 t g'(0)$ and $g'(t) < m_2 g'(0)$,
- t is acceptable, if $g(t) \leq g(0) + m_1 t g'(0)$ and $g'(t) \geq m_2 g'(0)$.

This leads to *Wolfe's line search*. An interpretation of these conditions is provided in Figure 2.4. The method is summarised in Algorithm 6.

While in general Wolfe's line search should be preferred over the other methods, in situations where the evaluation of g' takes considerably more time than the evaluation of g alone the method of Goldstein and Price should take precedence. Note moreover that Wolfe's line search is very well suited for the Quasi-Newton methods to be discussed in the next chapter, as its usage ensures super-linear convergence of the algorithm.

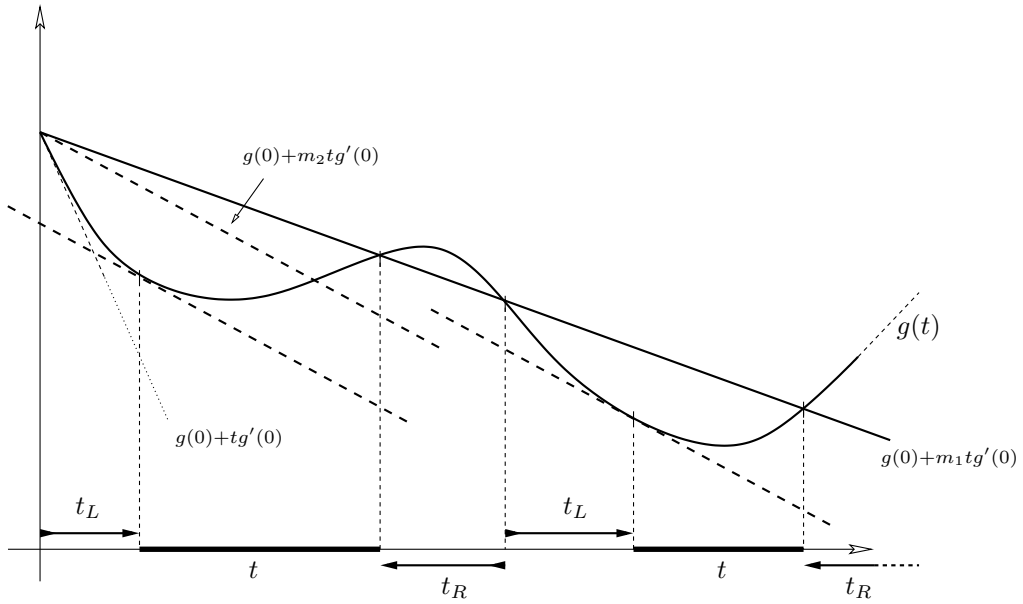

Figure 2.4: Wolfe's line search. In addition to Armijo's rule, one compares the derivative of g at t with the derivative of g at the origin.

Initialisation: set $t_L = 0$ and $t_R = +\infty$;
choose some initial $t > 0$;
declare t inacceptable;
fix $0 < m_1 \leq m_2 < 1$;
while *t is inacceptable* **do**
  if $g(t) > g(0) + m_1 t g'(0)$ **then**
    set $t_R \leftarrow t$;
    choose new $t \in (t_L, t_R)$;
  else if $g'(t) < m_2 g'(0)$ **then**
    set $t_L \leftarrow t$;
    choose new $t \in (t_L, t_R)$;
  else
    declare t acceptable;
  end
end
define $t^* := t$;

Algorithm 6: Wolfe's line search.

2.2.4 Choice of the Constants and Comments

In all three methods described above, it is necessary to fix some constants (either only m_1 or both m_1 and m_2). While in theory arbitrary choices $0 < m_1 < m_2 < 1$ are possible, in practise it makes sense to obey the restrictions

$$0 < m_1 < \frac{1}{2}$$

and, for Goldstein and Price,

$$\frac{1}{2} < m_2 < 1 \, .$$

One major reason is that this guarantees that for quadratic functions g the optimal step size will not be rejected. Indeed, assume that $g(t) = -at + bt^2$ for some $a, b > 0$. This function attains its minimum at the point $t^* = a/(2b)$, and the minimal value is $g(t^*) = -a^2/(4b) = -at^*/2$. This point satisfies Armijo's rule, if and only if

$$-\frac{at^*}{2} = g(t^*) \leq g(0) + m_1 t^* g'(0) = -m_1 a t^* \, ,$$

that is, if and only if $m_1 \leq 1/2$. Similarly, the second condition in the Goldstein–Price rule is satisfied, if and only if $m_2 \geq 1/2$.

This choice of the constant is particularly important, if the direction d is obtained by a Newton type method (see Chapter 3), where the function f is locally approximated by a quadratic function. In this situation all the good properties of the method are destroyed, if these restrictions are not satisfied.

In addition, in all the discussed methods it is advisable to add two further stopping criteria: Stop the iteration if t_L becomes too large, and also if the difference $t_R - t_L$ becomes too small.

2.3 Interpolation and Extrapolation

While the above methods provide criteria for when to stop the search for a good step size, they do not tell how one should proceed when a step size is rejected. The selection of the next candidate for the step size in the line search is the topic of this section. Here we always assume that we are already given some first candidate $0 < t^{(1)} < +\infty$.

The computation of the step size can be subdivided in two different phases. The first phase is that of *extrapolation*, which occurs as long as the upper bound t_R is $+\infty$. The second phase is that of *interpolation*, when both t_L and t_R are finite numbers.

The simplest methods for extrapolation and interpolation are the following:

- *Extrapolation:* Fix $a > 1$ and define $t^{(k+1)} := at_L$.
- *Interpolation:* Set $t^{(k+1)} := (t_L + t_R)/2$.

While these methods do work, they are typically not very efficient. At least if we do not only have access to g but also to g', it is most of the time better to base the computation of $t^{(k+1)}$ on an approximation of the function g obtained from the values of g and g' at the previous iterates $t^{(k)}$ and $t^{(k-1)}$ (with $t^{(0)} := 0$).

Possibly the best method for computing $t^{(k+1)}$ is to fit a cubic function ψ in such a way that

$$\psi(t^{(k)}) = g(t^{(k)})\,, \qquad \psi(t^{(k-1)}) = g(t^{(k-1)})\,,$$
$$\psi'(t^{(k)}) = g'(t^{(k)})\,, \quad \psi'(t^{(k-1)}) = g'(t^{(k-1)})\,.$$

Then one can define $t^{(k+1)}$ as the local minimum of ψ provided it exists and satisfies the desired restriction $t_L < t^{(k+1)} < t_R$ (note that a cubic function has at most one local minimum and never a global minimum).

For the computation of the local minimum of the fitting polynomial ψ define

$$p := g'(t^{(k)}) + g'(t^{(k-1)}) - 3\frac{g(t^{(k)}) - g(t^{(k-1)})}{t^{(k)} - t^{(k-1)}}\,,$$

set

$$D := p^2 - g'(t^{(k)})g'(t^{(k-1)}) \qquad \text{and} \qquad d := -\sqrt{D}\,\mathrm{sgn}(t^{(k)} - t^{(k-1)})\,.$$

If the *discriminant* D is smaller then zero, then ψ has no local minimum. Else the local minimum is

$$t_{\min} = t^{(k)} + r(t^{(k-1)} - t^{(k)})\,, \qquad \text{where} \qquad r := \frac{d + p - g'(t^{(k)})}{2d + g'(t^{(k-1)}) - g'(t^{(k)})}\,. \tag{2.2}$$

- *Extrapolation:* Fix $a > 1$. If $t_{\min}$ defined in (2.2) exists define

  $$t^{(k+1)} := \max\{t_{\min}, at_L\}\,,$$

  else set $t^{(k+1)} := at_L$.

- *Interpolation:* Fix $0 < \theta < 1/2$. If $t_{\min}$ defined in (2.2) exists define

  $$t^{(k+1)} := \max\big\{\min\{t_{\min}, t_R - \theta(t_R - t_L)\}, t_L + \theta(t_R - t_L)\big\}\,,$$

  else set $t^{(k+1)} := (t_L + t_R)/2$.

Thus it is guaranteed that the lower bound in the extrapolation phase always increases by a constant factor $a > 1$. Similarly, in the interpolation phase, the next step size will always be in the interval (t_L, t_R), but never too close to the boundaries t_L and t_R. Note that the constants $a > 1$ and $0 < \theta < 1/2$ should never be chosen too small, as this could lead to a large number of required iterations. Again it has to be stressed that the line search is only a very small part of the optimisation algorithm.

Chapter 3

Higher Order Methods

3.1 Newton's Method

The idea behind Newton's method is to approximate the cost function f first locally by a quadratic function and then to minimise this quadratic function exactly. An iteration of this procedure then leads to a theoretically extremely efficient algorithm.

Assume that $x \in \mathbb{R}^n$ is given and that $f \in C^2(\mathbb{R}^n)$. A Taylor expansion of f around x yields

$$f(x+d) = f(x) + \langle \nabla f(x), d \rangle + \frac{1}{2} d^T H_f(x) d + o(|d|^2) \; .$$

Thus the mapping

$$d \mapsto f(x) + \langle \nabla f(x), d \rangle + \frac{1}{2} d^T H_f(x) d$$

provides a good approximation of f around x. In order to minimise this approximation (if possible), we compute the zeros of its gradient, which is given by

$$d \mapsto \nabla f(x) + H_f(x) d \; .$$

Thus the stationary points d of the approximation satisfy the equation

$$H_f(x)\, d = -\nabla f(x) \; .$$

The basic Newton method now simply uses the vector d for defining the next iteration (see Algorithm 7).

Theorem 3.1.1. *Assume that $f \in C^2(\mathbb{R}^n)$ and that $H_f(x^*)$ is invertible. Then Newton's method converges locally Q-superlinear. If in addition $f \in C^3(\mathbb{R}^n)$, then the convergence is locally Q-quadratic.*

Note that this result does not claim that we have global convergence . Indeed, for arbitrary initialisation x_{init}, we can expect Newton's method to diverge. One possibility for obtaining a higher probability of convergence is the combination of Newton's method with one of the line search algorithms of the previous chapter. Typically, the rules of Wolfe or Goldstein and Price are used, but it is

Data: a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$;
an initial guess x_{init};
Result: $x^* \in \mathbb{R}^n$;
Initialisation: set $x^{(1)} := x_{\text{init}}$, $k = 1$;
while *convergence criterion not yet satisfied* **do**
  define $d^{(k)}$ by solving the equation

$$H_f(x^{(k)})\, d^{(k)} = -\nabla f(x^{(k)})$$

  define $x^{(k+1)} := x^{(k)} + d^{(k)}$;
  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$;

Algorithm 7: Newton's method.

also possible to rely solely on Armijo's rule, as Newton's method not only yields a direction, but also a theoretically optimal guess for the step size, namely $t = 1$. Also, the divergence of Newton's method is due to too large step sizes and not too small ones (which are the main problem of Armijo's rule).

Recall here the discussion in Section 2.2.4. If one combines Newton's method with a line search, then it is absolutely necessary to guarantee that the initial guess $t = 1$ will be classified as acceptable step size when one is already close to the actual minimum. Else the quadratic, but possibly also the superlinear convergence, will be lost, which is the one major reason why Newton's method is used at all. For the line searches discussed before, the acceptance of the initial guess $t = 1$ can be guaranteed, if the constants m_1 and m_2 are chosen such that $0 < m_1 < 1/2$ and, in the case of Goldstein and Price, $1/2 < m_2 < 1$.

The resulting method is summarised in Algorithm 8. In the case of a strictly convex target function f, one can also derive a global convergence result. We will state the theorem only for the case of Wolfe's line search, but analogous results also hold for the other two methods.

Theorem 3.1.2. *Assume that $f \in C^2(\mathbb{R}^n)$ is strictly convex with (unique) minimiser x^* and $H_f(x^*)$ is invertible. Then Newton's method with line search based on Wolfe's rule converges globally Q-superlinear to x^*, if the line search is initialised with $t = 1$ and the parameters satisfy $0 < m_1 < 1/2$ and $m_1 < m_2 < 1$. If $f \in C^3(\mathbb{R}^n)$, the convergence is Q-quadratic.*

3.1.1 Drawbacks of Newton's Method

Apart from the divergence of the method unless combined with a line search, there are at least two additional problems that make the usage of Newton's method problematic in many applications. First, the motivation of the method itself—first approximation of f by a quadratic function, then minimisation of the approximation—is reasonable, only if the quadratic approximation of f at x has a minimum. This is the case, only if the Hessian $H_f(x)$ is positive semi-definite. Now note that the Hessian $H_f(x)$ is positive semi-definite for *every*

Data: a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$;
an initial guess x_{init};
Result: $x^* \in \mathbb{R}^n$;

Initialisation: set $x^{(1)} := x_{\text{init}}$, $k = 1$;

while *convergence criterion not yet satisfied* **do**
  define $d^{(k)}$ by solving the equation

$$H_f(x^{(k)})\, d^{(k)} = -\nabla f(x^{(k)})$$

  find a good step size $t^{(k)}$ by applying a line search with direction $d^{(k)}$ and initialisation $t = 1$;
  define $x^{(k+1)} := x^{(k)} + t^{(k)} d^{(k)}$;
  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$;

Algorithm 8: Newton's method with line search.

$x \in \mathbb{R}^n$, if and only if f is convex. Thus, one can say that Newton's method makes only sense for the minimisation of convex functions.

Even for convex functions, problems arise if they are not strictly convex. Then, at some points x, the Hessian $H_f(x)$ will only be positive semi-definite, but not positive definite. That is, at least one eigenvalue is zero, which implies that the matrix $H_f(x)$ is not invertible. Then the solution of the equation $H_f(x)d = -\nabla f(x)$ becomes problematic, in particular so, if one accounts for unavoidable rounding errors in the numerical solution. Thus, in this case Newton's method is not even well-defined.

Different problems arise, if one applies Newton's method to non-convex problems. Then, at some points x, the Hessian $H_f(x)$ will be indefinite. The problem at these points is that, although $H_f(x)$ might be invertible and thus the Newton direction $d = -H_f(x)^{-1}\nabla f(x)$ well-defined, the direction d need not be a descent direction. Then, instead of yielding a better direction than simple steepest descent, Newton's method provides in a sense the worst possible choice of a direction. In particular, this direction should not be used as the input of a line search algorithm. Thus, before starting the line search, it is at least necessary to check whether d is a descent direction, which is equivalent to testing whether $\langle \nabla f(x), d\rangle < 0$. If it is not, then one has to choose a different direction, for instance the negative gradient of f at x.

The second large problem in Newton's method is that, even if everything works in theory, that is, the function f is strictly convex, we still have to solve a linear equation $H_f(x)d = -\nabla f(x)$ for obtaining the descent direction d in each step. If the number of unknowns n is large, then it is not advisable to find d by inverting the matrix $H_f(x)$ and multiplying the inverse with $-\nabla f(x)$. Instead one should use any one of the many existing algorithms for the solution of systems of linear equations.

If f is strictly convex and, consequently, $H_f(x)$ positive semi-definite, classically the Cholesky decomposition of $H_f(x)$ is used. That is, one computes a lower triangular matrix L such that $H_f(x) = LL^T$, and then solves the equation

$H_f(x)d = -\nabla f(x)$ by first solving $Le = -\nabla f(x)$ for e by forward substitution, and then $L^T d = e$ by backward substitution.

Even better, if slightly less known, is the LDL-decomposition: We decompose the Hessian into

$$H_f(x) = LDL^T ,$$

where L is a lower triangular matrix with all diagonal entries equal to 1, and D a diagonal matrix with positive entries (provided $H_f(x) > 0$). The point is that this decomposition avoids the computation of square roots that appear in the Cholesky decomposition. Because of numerical errors, it cannot be guaranteed that the matrix stays positive definite during the Cholesky algorithm; thus it is in principle possible that the numerical realisation has to compute squares of negative numbers, at which point the algorithm will break down. Also, an LDL-decomposition can be possible even in cases where the matrix $H_f(x)$ itself is not positive definite—then, however, the entries of D need not be positive.

Whatever method one uses for computing the descent direction d in Newton's method, the computation time has to be taken into account when one studies the efficiency of the method. For decomposition methods, the time is typically of order n^3; iterative methods like conjugate gradients may often be faster. In addition, problems will arise when the matrix $H_f(x)$ is ill-conditioned, because then an accurate solution of the equation $H_f(x)d = -\nabla f(x)$ is impossible.

In the following two sections we discuss two methods that have been developed specifically to counter these two problems of Newton's method: the necessity of solving a large linear system in each step and the possibility that the obtained direction need not be a descent direction.

3.2 Quasi-Newton Methods

3.2.1 One-dimensional Motivation

Newton's method is often encountered not in the context of non-linear optimisation, but rather in that of the solution of non-linear equations. There one is given a non-linear mapping $G\colon \mathbb{R} \to \mathbb{R}$ and one is interested in finding some $x \in \mathbb{R}$ satisfying $G(x) = 0$. The idea is to employ an iterative algorithm, defining the next iterate $x^{(k+1)}$ by approximating the function G around $x^{(k)}$ linearly by $G(x) \approx G(x^{(k)}) + G'(x^{(k)})(x - x^{(k)})$, and then solving the linear equation $G(x^{(k)}) + G'(x^{(k)})(x - x^{(k)}) = 0$ for x. That is, one defines the next iterate as

$$x^{(k+1)} = x^{(k)} - \frac{1}{G'(x^{(k)})}\, G(x^{(k)}) .$$

Now, the computation of the derivative $G'(x^{(k)})$ can be avoided by noting that

$$G'(x^{(k)}) \approx \frac{G(x^{(k)}) - G(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} .$$

Using this additional approximation, we then obtain the next iterate as

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{G(x^{(k)}) - G(x^{(k-1)})}\, G(x^{(k)}) .$$

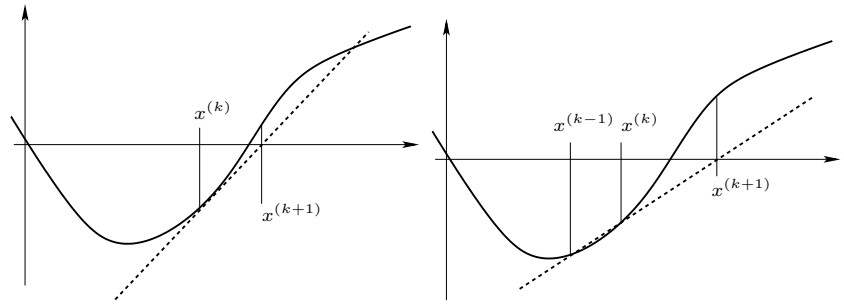

Figure 3.1: Comparison of the one-dimensional Newton method (left) and the secant method (right).

More intuitively, the idea behind this approach is to approximate the function G (or its graph) near $x^{(k)}$ by the *secant* through the points $\big(x^{(k)}, G(x^{(k)})\big)$ and $\big(x^{(k-1)}, G(x^{(k-1)})\big)$ and then to find the zero of this line. Hence the name: *secant method*. See also Figure 3.1

Now note that Newton's method for an optimisation problem $f(x) \to \min$ is precisely the same as Newton's method for the equation $f'(x) = 0$. Thus, in the one-dimensional case, we may use the iteration

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{f'(x^{(k)}) - f'(x^{(k-1)})}\, f'(x^{(k)})$$

for minimising f, thereby avoiding any computation of second order derivatives. This simplification, however, comes at a price: instead of quadratic convergence, one only has super-linear convergence (more precisely, the Q-convergence order equals the golden section $(\sqrt{5}+1)/2$). Note, moreover, that the secant method for one-dimensional optimisation problems also has an interpretation that is very similar to that of Newton's method. In Newton's method, we have approximated the function f by its best quadratic approximation near $x^{(k)}$, and then minimised this approximation. Here, we also use a quadratic approximation $\tilde{f}$ of f which is then minimised, but now we construct the function $\tilde{f}$ in such a way that it satisfies the equations

$$\begin{aligned} \tilde{f}(x^{(k)}) &= f(x^{(k)})\,, \\ \tilde{f}'(x^{(k)}) &= f'(x^{(k)})\,, \\ \tilde{f}''(x^{(k)})(x^{(k)} - x^{(k-1)}) &= f'(x^{(k)}) - f'(x^{(k-1)})\,. \end{aligned} \tag{3.1}$$

3.2.2 Higher-dimensional Generalisation

For generalising the one-dimensional secant method to higher dimensions we basically use the same ideas: We approximate the function f near $x^{(k)}$ by a quadratic function of the form

$$\tilde{f}(x^{(k)} + d) = f(x^{(k)}) + \big\langle \nabla f(x^{(k)}), d \big\rangle + \frac{1}{2} d^T M^{(k)} d$$

and define the search direction $d^{(k)}$ as the minimiser of the function $\tilde{f}$, which can be computed by solving the equation

$$M^{(k)}d = -\nabla f(x^{(k)})$$

for d. This approach makes sense, if the following conditions are satisfied:

- The matrix $M^{(k)}$ is symmetric and positive definite for all k.
- The equation $M^{(k)}d = -\nabla f(x^{(k)})$ can be solved without too much effort.
- The matrix $M^{(k)}$ in some sense approximates the Hessian $H_f(x^{(k)})$; else we cannot expect good convergence rates.

Note that the positive definiteness guarantees that

$$\langle \nabla f(x^{(k)}), d^{(k)} \rangle = -\big\langle \nabla f(x^{(k)}), (M^{(k)})^{-1} \nabla f(x^{(k)}) \big\rangle < 0\,,$$

that is, $d^{(k)}$ is a descent direction.

An even better approach is to directly approximate the *inverse* of the Hessian instead of the Hessian itself. Then, instead of solving a linear system in each step, we only have to perform a matrix–vector multiplication. In addition, usually, the approximation $W^{(k+1)}$ of $H_f(x^{(k+1)})^{-1}$ is built up inductively from $W^{(k)}$ by adding a correction term $B^{(k)}$, which only uses information about the iterations k and $k-1$. Moreover, one usually requires that the matrix $W^{(k+1)}$ satisfies the secant equation

$$W^{(k+1)}\big(\nabla f(x^{(k+1)}) - \nabla f(x^{(k)})\big) = x^{(k+1)} - x^{(k)}\,,$$

which gives the matrix $W^{(k+1)}$ at least some chance of approximating the inverse of the Hessian (cf. (3.1)). Thus we obtain the method described in Algorithm 9.

Data: a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$;
initial guess x_{init};
Result: $x^* \in \mathbb{R}^n$;

Initialisation: set $x^{(1)} := x_{\text{init}}$, $k = 1$, $W^{(1)} = \mathrm{Id}$;

while *convergence criterion not yet satisfied* **do**
  compute
  $$d^{(k)} = -W^{(k)}\nabla f(x^{(k)})$$
  find a good step size $t^{(k)}$ by applying a line search with direction $d^{(k)}$ and initialisation $t = 1$;
  define $x^{(k+1)} := x^{(k)} + t^{(k)}d^{(k)}$;
  compute a correction term $B^{(k)}$;
  define $W^{(k+1)} := W^{(k)} + B^{(k)}$;
  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$;

Algorithm 9: Quasi-Newton method.

The most important methods for computing the update $B^{(k)}$ are those due to Broyden, Fletcher, Goldfarb, Shanno, and Davidon, Fletcher, Powell. Denote in the following

$$s_k := x^{(k+1)} - x^{(k)}\,, \qquad y_k := \nabla f(x^{(k+1)}) - \nabla f(x^{(k)})\,.$$

Davidon–Fletcher–Powell (DFP)

Define

$$B^{(k)} := \frac{s_k s_k^T}{\langle y_k, s_k \rangle} - \frac{W^{(k)} y_k y_k^T W^{(k)}}{\langle y_k, W^{(k)} y_k \rangle} \ .$$

Note here that $s_k \in \mathbb{R}^{n\times 1}$ is a column vector and therefore the product $s_k s_k^T \in \mathbb{R}^{n\times n}$ is a matrix. Similarly, $W^{(k)} y_k \in \mathbb{R}^{n\times 1}$ is a column vector and $y_k^T W \in \mathbb{R}^{1\times n}$ a row vector, and thus, again, the product $W^{(k)} y_k y_k^T W^{(k)}$ is a matrix. Note moreover that the matrix $W^{(k)}$ is symmetric, and therefore $y_k^T W^{(k)} = y_k^T {W^{(k)}}^T = (W^{(k)} y_k)^T$.

Broyden–Fletcher–Goldfarb–Shanno (BFGS)

Here

$$B^{(k)} := -\frac{s_k y_k^T W^{(k)} + W^{(k)} y_k s_k^T}{\langle y_k, s_k \rangle} + \left(1 + \frac{\langle y_k, W^{(k)} y_k \rangle}{\langle y_k, s_k \rangle}\right) \frac{s_k s_k^T}{\langle y_k, s_k \rangle} \ .$$

Of these two methods, (DFP) is the older one, but (BFGS) is used more widely nowadays.

Remark 3.2.1. It is interesting to note that the two methods are in some sense dual to each other: If, starting with the identity matrix, one iteratively computes updates using (BFGS), but exchanges the roles of s_k and y_k, then the resulting matrix is precisely the inverse of the matrix obtained with (DFP). ■

Theorem 3.2.2. *Assume that $W^{(k)}$ is positive definite and $W^{(k+1)} = W^{(k)} + B^{(k)}$ is computed from $W^{(k)}$ using either of the rules (BFGS) or (DFP). Then $W^{(k+1)}$ is positive definite, if and only if $\langle y_k, s_k \rangle > 0$.*

Remark 3.2.3. Using the notation of line search algorithms, the condition $\langle y_k, s_k \rangle > 0$ from Theorem 3.2.2 translates to the condition $t^{(k)}\big(g'(t^{(k)}) - g'(0)\big) > 0$, or, as $t^{(k)} > 0$, the condition $g'(t^{(k)}) > g'(0)$. Because $W^{(k)}$ is positive definite, it follows that $d^{(k)}$ is a descent direction, that is, $g'(0) < 0$. Thus it is easy to check that the condition $g'(t^{(k)}) > g'(0)$ is necessarily satisfied, if a line search is performed using Wolfe's rule (with any parameters satisfying $0 < m_1 < m_2 < 1$). ■

Theorem 3.2.4. *Assume that $f \in C^3(\mathbb{R}^n)$ and $x^* \in \mathbb{R}^n$ satisfies $\nabla f(x^*) = 0$ and $H_f(x^*) > 0$. Consider the Quasi-Newton method with updates of $W^{(k)}$ according to BFGS and a line search with Wolfe's rule using parameters $0 < m_1 < 1/2$ and $m_1 < m_2 < 1$ and an initialisation $t = 1$. If $x^{(k)}$ converges to x^*, then the convergence is Q-super-linear.*

Remark 3.2.5. If the dimension n of the problem becomes too large, then it will be impossible to store the whole matrix $W^{(k)}$. Or, if storage is no problem, still the multiplications $W^{(k)} \nabla f(x^{(k)})$ for the new descent direction and $W^{(k)} y_k$ in DFP might take unreasonably long (storage and time needed are both of

order $O(n^2)$). Note, however, that we can compute the product of $W^{(k)}$ and a vector $z \in \mathbb{R}^n$ also as

$$W^{(k)}z = z + \sum_{j=1}^{k-1} B^{(j)}z = z + \sum_{j=1}^{k-1} \frac{\langle z, s_k^T\rangle}{\langle y_k, s_k\rangle} s_k - \sum_{j=1}^{k-1} \frac{\langle z, W^{(k)}y_k\rangle}{\langle y_k, W^{(k)}y_k\rangle} W^{(k)}y_k \,.$$

Thus, in fact it is only necessary to store the vectors s_k, $W^{(k)}y_k \in \mathbb{R}^n$, and the numbers $\langle y_k, s_k\rangle$, $\langle y_k, W^{(k)}y_k\rangle \in \mathbb{R}$, and one only has to perform $2k$ inner products, scalar multiplications, and additions for the computation of $W^{(k)}z$. If $k \ll n$, this observation greatly enhances the performance of the Quasi-Newton method: Computation time and storage are both of order $O(nk)$ for the k^{th} iteration. The same idea can also be applied to BFGS.

In addition, this representation quite naturally gives rise to an adaptation to the limited memory situation where either the number of iterations becomes large or the available memory is not much larger than n: Instead of storing all the vectors s_k and $W^{(k)}y_k$, one prescribes some number $m \in \mathbb{N}$ *a–priori* and only stores the last m vectors $s_{k-m+1}, \dots, s_k$ and $W^{(k-m+1)}y_{k-m+1}, \dots, W^{(k)}y_k$, thus obtaining a fixed computation time and storage size of order $O(mn)$. ∎

3.3 Levenberg–Marquardt Method

A problem often encountered in optimisation is that of *parameter identification*. Here, the assumption is that one is given a function

$$\Phi\colon \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^q \,, \qquad (x, u) \mapsto v \,,$$

which relates *parameters* $x \in \mathbb{R}^n$ and an *input datum* $u \in \mathbb{R}^p$ with an *output* or *measurement datum* $v \in \mathbb{R}^q$.

Then one collects a number of measurement data $v^{(r)} \in \mathbb{R}^q$, $r = 1, \dots, s$, corresponding to (known) input data $u^{(r)} \in \mathbb{R}^p$, $r = 1, \dots, s$. Now one wants to find the set of parameters $x^* \in \mathbb{R}^n$ that have generated these data. That is, one wants to solve the system of equations

$$\Phi(x, u^{(r)}) = v^{(r)} \qquad \text{for all } 1 \le r \le s \tag{3.2}$$

for the vector x.

The classical example is that of *linear regression*, where the model Φ is linear (or rather affine). Here one wants to find a matrix $A \in \mathbb{R}^{q\times p}$ and a vector $b \in \mathbb{R}^q$ such that

$$Au^{(r)} + b = v^{(r)} \qquad \text{for all } 1 \le r \le s \,.$$

This is a linear equation for the unknowns $(A, b) \in \mathbb{R}^{(q\times p)\times p}$, which is overdetermined if the number of data points is larger than the number of free parameters, that is, if $rs > q(p+1)$. In this case, this equation can only be solved exactly, if the model is consistent with the data. This assumption, however, is hardly ever justified, because, first, a linear model is usually only a first approximation of reality and, second, the data $v^{(r)}$ can usually only be measured with some finite accuracy. Then, instead of trying to solve the system of equations exactly, which is impossible, one only tries to solve it approximately by minimising the

squared data misfit. That is, one finds (A, b) satisfying

$$\frac{1}{2}\sum_{r=1}^{s}\|Au^{(r)} + b - v^{(r)}\|^2 \to \min\ .$$

In the case of non-linear models Φ, the problems are even larger than in the linear setting. Thus one arrives at considering the *non-linear least squares problem*

$$f(x) := \frac{1}{2}\sum_{r=1}^{s}\|\Phi(x, u^{(r)}) - v^{(r)}\|^2 = \frac{1}{2}\sum_{r=1}^{s}\sum_{j=1}^{q}\big(\Phi_j(x, u^{(r)}) - v_j^{(r)}\big)^2 \to \min\ . \quad (3.3)$$

In order to solve the minimisation problem (3.3) in the non-linear case, it is possible to apply a Newton iteration. To that end, we need the gradient and the Hessian of the function f. Note here that, in contrast to the optimisation problems discussed before, we have some knowledge of the function f to be minimised. It is only the model Φ that is assumed to be given only by some black-box.

Computing the gradient of f is quite easy. One has

$$\begin{aligned}\nabla f(x) &= \sum_{r=1}^{s} D_x\Phi(x, u^{(r)})^T\big(\Phi(x, u^{(r)}) - v^{(r)}\big)\\ &= \sum_{r=1}^{s}\sum_{j=1}^{q}\big(\Phi_j(x, u^{(r)}) - v_j^{(r)}\big)\,\nabla_x\Phi_j(x, u^{(r)})\ .\end{aligned}$$

Now the Hessian is also quite straightforward. Here one obtains

$$\begin{aligned}H_f(x) = &\sum_{r=1}^{s}\sum_{j=1}^{q}\nabla_x\Phi_j(x, u^{(r)})\nabla_x\Phi_j(x, u^{(r)})^T\\ &+ \sum_{r=1}^{s}\sum_{j=1}^{q}\big(\Phi_j(x, u^{(r)}) - v_j^{(r)}\big)\,\nabla_{xx}^2\Phi_j(x, u^{(r)})\ . \quad (3.4)\end{aligned}$$

Here, $\nabla_{xx}^2\Phi_j$ denotes the Hessian of the function Φ_j with respect to the x variable only.

In principle, it is possible to apply a Newton iteration with this gradient and this Hessian, that is, to apply the iteration

$$\begin{aligned}H_f(x^{(k)})d^{(k)} &= -\nabla f(x^{(k)})\,,\\ x^{(k+1)} &= x^{(k)} + d^{(k)}\ .\end{aligned}$$

From a computational point of view, one difficulty can be the fact that in each iteration step the Hessian of Φ has to be computed, which may be difficult to obtain. One possibility of circumventing the evaluation of the Hessian is the usage of Quasi-Newton methods. Because of the special structure of H_f, however, is is possible to obtain a good estimate for values of x that are almost solutions of (3.2).

The Hessian $H_f(x)$ consists of two terms: In the first term, only the gradients of the functions Φ_j enter. The second term is a weighted sum of the Hessians of

the functions Φ_j, but the weights are precisely the residuals $\Phi_j(x, u^{(r)}) - v_j^{(r)}$). Therefore, if we are close to a solution of the system (3.2), the second term almost vanishes and the first term alone provides a good estimate of $H_f(x)$.

This idea of replacing H_f by

$$G(x) := \sum_{r=1}^{s} \sum_{j=1}^{q} \nabla_x \Phi_j(x, u^{(r)}) \nabla_x \Phi_j(x, u^{(r)})^T \tag{3.5}$$

for the definition of the update $d^{(k)}$ in the Newton iteration gives rise to the *Gauß–Newton method*, which is defined by the the iteration

$$\begin{aligned} G(x^{(k)}) d^{(k)} &= -\nabla f(x^{(k)}) \,, \\ x^{(k+1)} &= x^{(k)} + d^{(k)} \,. \end{aligned}$$

Obviously, this method can also be combined with a line search for determining a step size.

Remark 3.3.1. Note that neither Gauß nor Newton have considered this algorithm. The name comes rather from the fact that a variant of Newton's method is applied to the solution of a (non-linear) least squares problem, and Gauß is possibly the first who has applied least squares to parameter identification problems (despite that the first publication concerning least squares is due to Legendre). ∎

The convergence properties of the Gauß–Newton depend strongly on the properties (and validity) of the model Φ. In the case where a solution x^* of (3.2) exists, the equation $H_f(x^*) = G(x^*)$ holds. Therefore, if $H_f(x^*)$ is positive definite, one can use the results concerning the Newton iteration, which imply that the method converges (locally near x^*) at least super-linearly.

If the equation (3.2) has no solution, one cannot expect that the Hessian $H_f(x^*)$ at a minimiser of f coincides with $G(x^*)$. Thus the algorithm cannot be expected to converge even with super-linear speed. In practise, however, the method shows reasonable convergence properties, if the Hessians $\nabla_{xx} \Phi_j(x^*, u^{(r)})$ have entries that are much smaller than the inverse of the residual $|\Phi_j(x^*, u^{(r)}) - v_j^{(r)}|$ and the matrix $G(x^*)$ is well conditioned.

If these conditions are not satisfied, however, it can happen that the method does not converge at all, because then the vector $d^{(k)}$ may be almost orthogonal to the (negative) gradient of f and thus hardly a descent direction at all. This situation, however, often occurs in practise, in particular if the model Φ has the property that large variations of the parameter x may lead only to comparably small variations of the output $v = \Phi(x, u)$, that is, the problem of solving the equation (3.2) is ill-conditioned (or even ill-posed).

In order to ensure that the direction $d^{(k)}$ is a descent direction, it is possible to consider a regularisation of the equation $G(x)d = -\nabla f(x)$. Note that the matrix $G(x)$ is positive semi-definite, because

$$\begin{aligned} d^T G(x) d = \sum_{r=1}^{s} \sum_{j=1}^{q} d^T \nabla_x \Phi_j(x, u^{(r)}) \nabla_x \Phi_j(x, u^{(r)})^T d \\ = \sum_{r=1}^{s} \sum_{j=1}^{q} \langle d, \nabla_x \Phi_j(x, u^{(r)}) \rangle^2 \geq 0 \end{aligned}$$

for every $d \in \mathbb{R}^n$. Thus, for every $\lambda > 0$ the matrix $\lambda \operatorname{Id} + G(x)$ is strictly positive definite. In particular, the (strict) positive definiteness implies that a direction defined by the *regularised* matrix $\lambda \operatorname{Id} + G(x)$ will indeed be a descent direction for f. This idea gives rise to the *Levenberg–Marquardt* (or *regularised Gauß–Newton*) method, which is defined by the iteration

$$\begin{aligned} \big(\lambda_k \operatorname{Id} + G(x^{(k)})\big) d^{(k)} &= -\nabla f(x^{(k)}) \,, \\ x^{(k+1)} &= x^{(k)} + d^{(k)} \,, \end{aligned}$$

where $(\lambda_k)_{k \in \mathbb{N}}$ is a sequence of positive *regularisation* parameters.

The final question is, how to choose the regularisation parameters λ_k. One possibility is to use a constant sequence, that is, $\lambda_k = \lambda_0$ for some fixed λ_0 (which still has somehow to be defined, though), and add some flexibility to the method by performing a line search.

Similarly, one can define a sequence λ_k beforehand, for instance defining $\lambda_k = \lambda_0 / k$ for some fixed λ_0. In this case, it is advisable to choose the sequence in such a way that $\sum_k \lambda_k = +\infty$. A different class of parameter selection rules defines the step size λ_k using the step size λ_{k-1} and the residual $f(x_{k-1})$, possibly also λ_{k-2} and $f(x_{k-2})$. In these cases, no line search is necessary.

Finally, it is possible (and makes sense) to consider the Levenberg–Marquardt method as a special instance of a *trust region method* (see Section 3.4 below) and use the ideas to be developed there.

Remark 3.3.2. In the case of parameter estimation, the right hand side typically are data that are obtained by some measurement process. If the model is sufficiently accurate, then the only reason why one cannot obtain a solution of the problem is that the data cannot be measured with arbitrary precision, but instead are subject to some measurement errors.

Now assume that one knows that the measurement error is of size $\delta > 0$, that is, there exist some (hypothetical, unknown) true data $\hat{v}^{(r)}$, $r = 1, \dots, s$, such that the given data satisfy $\|v^{(r)} - \hat{v}^{(r)}\| \leq \delta$ for all r. Moreover, we assume that the model is consistent, that is, there exists a unique set of parameters $\hat{x} \in \mathbb{R}^n$ with $\Phi(\hat{x}, u^{(r)}) = \hat{v}^{(r)}$ for all r. Then

$$f(\hat{x}) = \sum_{r=1}^{s} \|\Phi(\hat{x}, u^{(r)}) - v^{(r)}\|^2 = \sum_{r=1}^{s} \|\hat{v}^{(r)} - v^{(r)}\|^2 \leq s\delta^2 \,.$$

In order to incorporate this quantitative knowledge about the true solution into the Levenberg–Marquardt method, it is possible to stop the iteration as soon as $f(x_k) \leq s\delta^2$. Then one can be sure that the obtained approximation x_k of $\hat{x}$ is consistent in the sense that the measured data $v^{(r)}$ can have originated from the parameters x_k and the input data $u^{(r)}$ with the noise level δ. In addition, if the equation (3.2) is ill-posed (or severely ill-conditioned), the results obtained this way can be expected to be better than the results obtained by actually minimising f. Also, under some additional conditions, it is possible to give some estimates for the error $\|x_k - \hat{x}\|$ in dependence of the noise level δ.

Note that this does not apply directly to the case of linear regression, where one typically assumes that the error in the data is due to some random effects (instead of *deterministic* measurement errors). Still, if one has some (statistical) model of the data error, it is also there possible to formulate a meaningful stopping criterion for the Levenberg–Marquardt method. ∎

3.4 Trust Region

The idea of trust region methods is to approximate the function f near the current iterate $x^{(k)}$ by some model $\tilde{f}$ and then to minimise this model on a ball centred at $x^{(k)}$ of radius $t^{(k)} > 0$. Here, the radius $t^{(k)}$ is chosen in such a way that the model $\tilde{f}$ looks like a good approximation of f on that ball. Put differently, one looks for a set, where the model $\tilde{f}$ is a valid description of the function f to be minimised and then computes the exact minimiser of the model $\tilde{f}$ on this *trust region*.

We note first that steepest descent combined with a line search can be seen as a special case of a trust region method. Indeed, consider the approximation of f near x by the affine function

$$d \mapsto \tilde{f}(x+d) := f(x) + \big\langle \nabla f(x), d \big\rangle \, .$$

Minimisation of $\tilde{f}$ over a ball of radius t centred at x yields

$$d := \arg\min\big\{ f(x) + \langle \nabla f(x), \tilde{d} \rangle : \tilde{d} \in \mathbb{R}^n, \ \|\tilde{d}\| \le t \big\} = -t \, \frac{\nabla f(x)}{\|\nabla f(x)\|} \, .$$

That is, we obtain precisely the steepest descent method with step size t. Now recall that one main criterion for the choice of the step size in linear search algorithm is Armijo's rule, which rejects a proposed step size t as too large, if the predicted decrease of the cost function is much larger than the actual decrease. This discrepancy between prediction and reality indicates that the chosen model is not very accurate at a distance t from x. Thus line search algorithms can be interpreted as test for the validity of the affine approximation $f(x+td) \approx f(x) + t\langle \nabla f(x), d \rangle$.

Usually, trust region methods do not work with linear approximations, but rather with quadratic ones, that is, approximations of the form

$$f(x+d) \approx \tilde{f}(x+d) := f(x) + \big\langle \nabla f(x), d \big\rangle + \frac{1}{2} d^T G(x) d \tag{3.6}$$

for some symmetric matrix $G(x) \in \mathbb{R}^{n \times n}$. For instance, one may use $G(x) = H_f(x)$, or, in the case of parameter estimation, the matrix G defined there. Note that we do not require the matrix G to be positive definite or even semi-definite.

With this approximation, one iteration of a trust region method typically looks as follows:

1. Compute a minimiser d of the approximation $\tilde{f}$ of f defined in (3.6) subject to the constraint $\|d\| \le t$.

2. Compute the size of the predicted and the actual update,

$$\begin{aligned} \Delta \tilde{f} &:= \tilde{f}(x+d) - \tilde{f}(x) = \langle \nabla f(x), d \rangle + \frac{1}{2} d^T G(x) d \, , \\ \Delta f &:= f(x+d) - f(x) \, . \end{aligned}$$

3. If the size of the actual update Δf is not much smaller than the predicted update $\Delta \tilde{f}$, define the new iterate as $x + d$. Else retain the old iterate x, but decrease the size t of the trust region.

Mimicking Armijo's rule, one usually rejects an update if $\Delta f > m\Delta\tilde{f}$, where $0 < m < 1$ is some constant fixed *a–priori* (note that $\Delta\tilde{f}$ will be negative, unless x itself minimises $\tilde{f}$). In contrast to steepest descent combined with line search, where, usually, the computation of the direction is separated from the computation of the step size, here both direction and step size are computed within the same iteration, but the iterate x is not necessarily updated in every step.

If one only uses the steps described above, the algorithm will not be very efficient, as the step size t is only allowed to decrease and never to increase. In practise, one usually increases the size of the trust region whenever an update step d is deemed acceptable. In addition, one usually increases and decreases the radius t by some fixed factor. Then one obtains the method summarised in Algorithm 10.

Data: a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$;
initial guess x_{init};
radius t_{init} of initial trust region;
parameters $0 < m < 1$ and $0 < a_1 < 1 < a_2$;
Result: $x^* \in \mathbb{R}^n$;

Initialisation: set $x^{(1)} := x_{\text{init}}$, $t^{(1)} := t_{\text{init}}$, $k = 1$;

while *convergence criterion not yet satisfied* **do**
  compute

$$d^{(k)} := \arg\min\left\{ \langle \nabla f(x^{(k)}), d\rangle + \frac{1}{2} d^T G(x^{(k)}) d : \|d\| \le t^{(k)} \right\} ;$$

  compute

$$\Delta^{(k)} := \frac{f(x^{(k)} + d^{(k)}) - f(x^{(k)})}{\langle \nabla f(x^{(k)}), d^{(k)}\rangle + \frac{1}{2} {d^{(k)}}^T G(x^{(k)}) d^{(k)}} ;$$

  if $\Delta^{(k)} > m$ **then**
    set $x^{(k+1)} := x^{(k)} + d^{(k)}$;
    set $t^{(k+1)} := a_2 t^{(k)}$;
  else
    set $x^{(k+1)} := x^{(k)}$;
    set $t^{(k+1)} := a_1 t^{(k)}$;
  end
  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$;

Algorithm 10: Sketch of a trust region method.

The major difficulty in the trust region method is the computation of the steps $d^{(k)}$. In contrast to steepest descent and also Newton's method, it is in general not possible to compute $d^{(k)}$ analytically. The difficulty is that we do not have a free optimisation problem, but rather the constrained problem

$$f(x) + \left\langle \nabla f(x), d \right\rangle + \frac{1}{2} d^T G(x) d \to \min \qquad \text{subject to } \|d\| \le t\,. \tag{3.7}$$

For that reason, it makes sense to look for optimisation methods for constrained problems.

Replacing the constraint $\|d\| \leq t$ by the equivalent, but differentiable, constraint $\|d\|^2 \leq t^2$, we consider the *Lagrangian* $L\colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$,

$$L(d, \lambda) := \tilde{f}(x + d) + \frac{\lambda}{2}\big(\|d\|^2 - t^2\big) .$$

For fixed λ satisfying $\lambda > -\lambda_{\min}$ with $\lambda_{\min}$ denoting the smallest eigenvalue of $G(x)$ the function $d \mapsto L(\cdot, \lambda)$ has a unique minimiser $d^*(\lambda)$, which satisfies

$$\big(G(x) + \lambda \operatorname{Id}\big)\, d^*(\lambda) = -\nabla f(x) . \tag{3.8}$$

Theorem 3.4.1. *Let $\lambda^* \geq \lambda_0 := \max\{0, -\lambda_{\min}\}$ and let $d^*(\lambda^*)$ be as in (3.8). If either $\big\|d^*(\lambda^*)\big\| < t$ and $\lambda^* = 0$, or $\big\|d^*(\lambda^*)\big\| = t$, then $d^*(\lambda^*)$ solves (3.7). Moreover, the mapping $\lambda \mapsto L(d^*(\lambda), \lambda)$ is concave on $[\lambda_0, +\infty)$, smooth, and attains its maximum at λ^*.*

This last result states that instead of solving (3.7) we may also maximise the concave function $\lambda \mapsto L(d^*(\lambda), \lambda)$, which can be done efficiently using Newton's method (for the minimisation of $-L(d^*(\cdot), \cdot)$), provided that the minimiser d of (3.7) satisfies $\|d\| = t$. To that end, we have to compute the first and second derivative of the mapping $\lambda \mapsto H(\lambda) := -L(d^*(\lambda), \lambda)$. First note that the equation $(G(x) + \lambda \operatorname{Id}) d^*(\lambda) = -\nabla f(x)$ implies that

$$H(\lambda) = -L(d^*(\lambda), \lambda) = \frac{1}{2}\big\langle \nabla f(x), (G(x) + \lambda \operatorname{Id})^{-1} \nabla f(x) \big\rangle + \lambda \frac{t^2}{2} .$$

Using the fact that

$$\partial_\lambda \Big[(G(x) + \lambda \operatorname{Id})^{-1} \nabla f(x) \Big] = -(G(x) + \lambda \operatorname{Id})^{-2} \nabla f(x) ,$$

we obtain

$$\begin{aligned} H'(\lambda) &= -\frac{1}{2}\big\langle \nabla f(x), (G(x) + \lambda \operatorname{Id})^{-2} \nabla f(x) \big\rangle + \frac{t^2}{2} \\ &= -\frac{1}{2}\|(G(x) + \lambda \operatorname{Id})^{-1} \nabla f(x)\|^2 + \frac{t^2}{2} \end{aligned}$$

As a consequence,

$$\begin{aligned} H''(\lambda) &= \big\langle (G(x) + \lambda \operatorname{Id})^{-1} \nabla f(x), (G(x) + \lambda \operatorname{Id})^{-2} \nabla f(x) \big\rangle \\ &= \big[(G(x) + \lambda \operatorname{Id})^{-1} \nabla f(x)\big]^T (G(x) + \lambda \operatorname{Id})^{-1} \big[(G(x) + \lambda \operatorname{Id})^{-1} \nabla f(x)\big] . \end{aligned}$$

It is also possible to solve the equation $\|d^*(\lambda)\| = t$ or the equivalent equation $1/\|d^*(\lambda)\| = 1/t$. Here, one can apply Newton's method for the solution of non-linear equations.

Remark 3.4.2. Obviously, the ideas of Quasi-Newton methods can be combined with a trust region method and one can also consider models of f, where $G(x)$ is built using a similar inductive strategy as for Quasi-Newton methods. Note, however, that here we want to approximate the Hessian itself and not its inverse. This can be achieved using the duality principle described in Remark 3.2.1.

Also, note that one of the strong points of the trust region method is that the positive definiteness of the matrix $G(x)$ (or the Hessian $H_f(x)$ for the basic method) is not required. ∎

Chapter 4

Conjugate Gradient Methods

4.1 Linear Conjugate Gradients

The conjugate gradient (CG) method is in its standard form a method for solving the linear equation

$$Ax = b\,,$$

where $A \in \mathbb{R}^{n\times n}$ is a symmetric and positive definite matrix. Because of the positive definiteness and symmetry of A, this equation can be reformulated as a quadratic optimisation problem. Define to that end the mapping $\Phi\colon \mathbb{R}^n \to \mathbb{R}$,

$$\Phi(x) := \frac{1}{2}x^T Ax - x^T b\,. \tag{4.1}$$

Then $\nabla\Phi(x) = Ax - b$, which shows that $x^* \in \mathbb{R}^n$ is a critical point of Φ, if and only if x^* satisfies $Ax^* = b$. Moreover, the Hessian of Φ is simply the positive definite matrix A, showing that Φ is strictly convex and therefore $x^* := A^{-1}b$ its unique minimiser.

There is more to the function Φ: Because A is positive definite, the mapping $\langle\cdot,\cdot\rangle_A\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$,

$$\langle x, y\rangle_A := x^T Ay = \langle x, Ay\rangle$$

defines a scalar product on $\mathbb{R}^n$. The corresponding norm

$$\|x\|_A := \sqrt{\langle x, x\rangle_A} = \sqrt{x^T Ax}$$

is called the *energy norm* induced by A. Rewriting the functional Φ, one can show that

$$\Phi(x) - \Phi(x^*) = \frac{1}{2}\|x - x^*\|_A^2\,.$$

Thus, minimising Φ is equivalent to minimising the squared distance to the solution of the equation $Ax = b$ measured in the energy norm.

In order to minimise Φ, we apply the ideas of the previous chapters. That is, we use an iterative method alternately finding descent directions $d^{(k)}$ and applying a line search along $d^{(k)}$. Now assume for the moment that we have

already found the k-th iterate $x^{(k)}$ and the descent direction $d^{(k)}$ along which we want to search for the next iterate. In Chapter 2, the reason for developing the sophisticated heuristics was that we did not have an analytic formula of the function to be minimised. Here the situation is different. Computing the derivative of Φ along $d^{(k)}$ we obtain

$$\begin{aligned}\partial_t \Phi(x^{(k)} + td^{(k)}) &= \big\langle \nabla\Phi(x^{(k)} + td^{(k)}), d^{(k)} \big\rangle \\ &= \big\langle A(x^{(k)} + td^{(k)}) - b, d^{(k)} \big\rangle = t\big\langle Ad^{(k)}, d^{(k)} \big\rangle + \big\langle Ax^{(k)} - b, d^{(k)} \big\rangle \,.\end{aligned}$$

Setting this derivative to 0, it follows with the abbreviation

$$r^{(k)} := Ax^{(k)} - b$$

that we should choose the step size $t^{(k)}$ as

$$t^{(k)} := -\frac{\langle r^{(k)}, d^{(k)} \rangle}{\langle Ad^{(k)}, d^{(k)} \rangle} \,.$$

The question remains, how to choose the descent direction $d^{(k)}$. We have seen in the first chapter that employing the gradient of Φ is not advisable, as the number of necessary iterations becomes unreasonably large. One problem is that often the descent directions will coincide with directions already chosen in a previous step. It would be preferable, if we were able to compute a step in one direction only once and then not to move again in this direction for the rest of the algorithm. For general non-quadratic minimisation problems, this is not possible; in the case of the quadratic functional Φ, it is.

In the CG method, the direction $d^{(k)}$ is chosen to be a linear combination of the previous direction $d^{(k-1)}$ and the residual $r^{(k)}$ (which is at the same time the gradient of Φ at $x^{(k)}$) in such a way that $d^{(k)}$ and $d^{(k-1)}$ are orthogonal with respect to the scalar product $\langle \cdot, \cdot \rangle_A$ (in other words: *conjugate* with respect to A). That is, one defines

$$d^{(k)} := r^{(k)} + \beta_k d^{(k-1)} \qquad \text{with } \beta_k \in \mathbb{R} \text{ such that } \langle Ad^{(k)}, d^{(k-1)} \rangle = 0 \,.$$

Explicitly, we obtain the coefficient

$$\beta_k = -\frac{\langle r^{(k)}, Ad^{(k-1)} \rangle}{\langle Ad^{(k-1)}, d^{(k-1)} \rangle} \,.$$

Definition 4.1.1. Let $r \in \mathbb{R}^n$. The k^{th} *Krylov-space* of A with respect to r is defined as

$$\mathcal{K}_k(A, r) := \operatorname{span}\big\{ r, Ar, A^2 r, \ldots, A^{k-1} r \big\} \,.$$

That is, $\mathcal{K}_k(A, r)$ is the space spanned by the vectors that one obtains by iteratively applying the matrix A to r. In particular, we set $\mathcal{K}_1(A, r) := \mathbb{R}r$ and $\mathcal{K}_0(A, r) := \{0\}$. ∎

Theorem 4.1.2. *Let $A \in \mathbb{R}^{n \times n}$ be symmetric and positive definite and let $\Phi\colon \mathbb{R}^n \to \mathbb{R}$ be as in (4.1). Let $x^{(0)} \in \mathbb{R}^n$ be arbitrary and consider the iteration (the CG method)*

$$x^{(k+1)} = x^{(k)} - \frac{\langle r^{(k)}, d^{(k)} \rangle}{\langle Ad^{(k)}, d^{(k)} \rangle} d^{(k)}$$

with $r^{(k)} := Ax^{(k)} - b$, $d^{(0)} = r^{(0)}$, *and*

$$d^{(k)} = r^{(k)} - \frac{\langle r^{(k)}, Ad^{(k-1)} \rangle}{\langle Ad^{(k-1)}, d^{(k-1)} \rangle} d^{(k-1)} ,$$

where we assume that the iteration is stopped if $r^{(k)} = 0$.

If $r^{(k)} = 0$, *then* $x^{(k)} = x^* = A^{-1}b$. *Else, the Krylov space* $\mathcal{K}_k(A, r^{(0)})$ *has dimension* k *and* $x^{(k)}$ *minimises* Φ *on the affine space* $x^{(0)} + \mathcal{K}_k(A, r^{(0)})$.

Corollary 4.1.3. *The CG method described in Theorem 4.1.2 finds the solution* x^* *of the equation* $Ax = b$ *in at most* n *steps.*

Some reformulation yields the method described in Algorithm 11.

Data: a positive definite and symmetric matrix $A \in \mathbb{R}^{n \times n}$;
some $b \in \mathbb{R}^n$;
an initial guess x_{init};
Result: the solution $x^* \in \mathbb{R}^n$ of $Ax = b$;
Initialisation: set $x^{(0)} := x_{\text{init}}$, $r^{(0)} = Ax^{(0)} - b$, $d^{(0)} = r^{(0)}$, $k = 0$;
while $r^{(k)} \neq 0$ **do**
  define

$$\alpha_k = -\frac{\|r^{(k)}\|^2}{\langle Ad^{(k)}, d^{(k)} \rangle} ;$$
$$x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)} ;$$
$$r^{(k+1)} = r^{(k)} + \alpha_k Ad^{(k)} ;$$
$$\beta_{k+1} = \frac{\|r^{(k+1)}\|^2}{\|r^{(k)}\|^2} ;$$
$$d^{(k+1)} = r^{(k+1)} + \beta_{k+1} d^{(k)} ;$$

  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$;

Algorithm 11: Conjugate Gradient method.

Remark 4.1.4. One of the big advantages of the conjugate gradient method is that it is in fact a black-box method: The matrix A is never needed in the computations; one only needs to know how to compute the product $Ad^{(k)}$. ■

Remark 4.1.5. Theoretically, solving an equation with the conjugate gradient method is not very efficient, because it requires more operations than for instance a Cholesky or LDL decomposition. In addition, the convergence after n steps is purely theoretical, as it only holds when all the computations are performed in exact arithmetic without any rounding errors. In practise, however, it performs much better, because one can, and should, make use of the advantage of iterative methods: One need not stop only when the residual equals zero, but rather when the residual is sufficiently small. ■

Remark 4.1.6. One problem of the CG method is that the convergence speed depends heavily on the condition of the matrix A. In order to improve the performance, usually the method is not applied directly to the equation $Ax = b$, but rather to a transformed problem $M^{-1}Ax = M^{-1}b$, where the (easily invertible) matrix M is chosen in such a way that $M^{-1}A$ is much better conditioned than A. This approach is called *preconditioning*.

In theory, a slightly different approach is required, as the matrix $M^{-1}A$ will not be symmetric any more. To that end one can consider a Cholesky factorisation $M = LL^T$ of the positive definite and symmetric matrix M and then apply the CG method to the equation $L^{-1}AL^{-T}\hat{x} = L^{-1}b$ and solve $L^T x = \hat{x}$. It is possible to rewrite the resulting algorithm in a such a way that one only needs the original matrix M and never its factorisation. ∎

4.2 Non-linear Conjugate Gradients

In principle, it is also possible to apply the idea of the CG method to non-quadratic optimisation problems. One chooses search directions $d^{(k)}$ as a suitable linear combination of the negative gradient $\nabla f(x^{(k)})$ and the previous search direction $d^{(k-1)}$, that is, one defines

$$d^{(k)} = -\nabla f(x^{(k)}) + \beta_k d^{(k-1)}$$

for some suitable $\beta_{k-1} \in \mathbb{R}$, and then performs a line search along $d^{(k)}$ to obtain the next iterate $x^{(k+1)}$.

The question remains how to choose β_{k-1}, as conjugacy has no real meaning in the non-quadratic case (note that the Hessian of f does not remain constant). There are two common choices. The first, Fletcher–Reeves (F-R), simulates the quadratic case. It uses

$$\beta_k := \frac{\|\nabla f(x^{(k)})\|^2}{\|\nabla f(x^{(k-1)})\|^2} .$$

The second, Polak–Ribière (P-R), tries to obtain conjugacy with respect to the (unknown) matrix $H_f(x^{(k+1)})$. It is defined by

$$\beta_k := \frac{\left\langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), \nabla f(x^{(k)}) \right\rangle}{\|\nabla f(x^{(k-1)})\|^2} .$$

In theory, (F-R) is more reliable and there exist examples where (P-R) does not converge. In most practical applications, (P-R) converges faster than (F-R).

There is a close connection between the CG method and the Quasi-Newton method with limited memory described in Remark 3.2.5: If one uses BFGS, but re-initialises the matrix $W^{(k)}$ in each step to the identity (that is, one only stores the vectors s_k and $W^{(k)}y_k$), then the resulting algorithm can be seen as a CG method where the coefficients β_{k-1} are chosen as

$$\beta_k = \frac{\left\langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), \nabla f(x^{(k)}) \right\rangle}{\left\langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), d^{(k-1)} \right\rangle} .$$

Data: a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$;
an initial guess x_{init};
Result: $x^* \in \mathbb{R}^n$;

Initialisation: set $x^{(0)} := x_{\text{init}}$, choose some threshold $\varepsilon > 0$;
set $d^{(0)} := -\nabla f(x^{(0)})$, $k = 0$;

while $\|\nabla f(x^{(k)})\| > \varepsilon$ **do**
  find a good step size $t^{(k)}$ by applying a line search with direction $d^{(k)}$;
  set $x^{(k+1)} := x^{(k)} + t^{(k)} d^{(k)}$;
  define $d^{(k+1)} = -\nabla f(x^{(k+1)}) + \beta_{k+1} d^{(k)}$;
  if $\left\langle d^{(k+1)}, \nabla f(x^{(k+1)}) \right\rangle \geq 0$ **then**
    $d^{(k+1)} \leftarrow -\nabla f(x^{(k+1)})$;
  end
  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$;

Algorithm 12: Non-quadratic Conjugate Gradient method.

Chapter 5

Constrained Optimisation

We now treat the solution of constrained optimisation problems, where we aim for the minimisation of a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$ over some subset $C \subset \mathbb{R}^n$. Moreover, we assume that the set C is given by means of equality and inequality constraints such that

$$C = \big\{x \in \mathbb{R}^n : c_j(x) \leq 0 \text{ for } j \in I \text{ and } c_j(x) = 0 \text{ for } j \in E\big\}$$

for some (sufficiently smooth) functions $c_j\colon \mathbb{R}^n \to \mathbb{R}$. As for the cost function f, we assume that also the constraints are only given through some black box, which, given $x \in \mathbb{R}^n$, returns the values $c_j(x)$, $\nabla c_j(x)$, and $H_{c_j}(x)$.

5.1 Equality Constraints

We first consider the situation that is easier to treat, where we have only equality constraints, say $c_j\colon \mathbb{R}^n \to \mathbb{R}$, $j = 1, \dots, m$. Then we have to solve the problem

$$f(x) \to \min \qquad \text{subject to} \qquad c_j(x) = 0 \text{ for all } j = 1, \dots, m\,. \tag{5.1}$$

As before, we restrict ourselves to the search of *local* minima of f on C. In contrast to the setting of free optimisation, however, for constrained optimisation the condition $\nabla f(x^*) = 0$ is not necessary for x^* to be a minimiser of f on C. Indeed, while it is still necessary that the derivative of f in directions along C vanishes, the function f the function f may vary freely in directions that lead away from the set C. Now note that the negative gradients $-\nabla c_j(x)$ of the functions c_j surely lead away from C, as $-\nabla c_j$ is the direction of steepest descent of c_j and the set C is the zero set of all functions c_j. If the set C and the functions c_j are sufficiently "nice" (the constraints are *qualified* at x^*), then these are in fact the only directions leading away from C. In this case, a necessary condition for x^* to minimise f on C is that $\nabla f(x^*)$ is a linear combination of the negative gradients $-\nabla c_j(x^*)$. In other words: there exist parameters λ_j, $j = 1, \dots, m$, such that

$$\nabla f(x^*) = -\sum_{j=1}^{m} \lambda_j \nabla c_j(x^*)\,.$$

Definition 5.1.1. The mapping $L\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$,

$$L(x,\lambda) := f(x) + \sum_{j=1}^{m} \lambda_j c_j(x)$$

is called the *Lagrangian* of the problem (5.1). The parameter $\lambda \in \mathbb{R}^m$ is called *Lagrange multiplier.* ∎

Proposition 5.1.2. *Assume that x^* is a local minimiser of f subject to the constraints $c_j(x^*) = 0$ for all j (that is, $c_j(x^*) = 0$ for every j and there exists $\varepsilon > 0$ such that $f(x^*) \le f(x)$ for every $x \in \mathbb{R}^n$ satisfying $|x - x^*| < \varepsilon$ and $c_j(x) = 0$ for every j). Assume moreover that the cost function f and the constraints c_j are C^2 and that the vectors $\nabla c_j(x^*)$, $j = 1, \dots, m$, are linearly independent. Then there exists $\lambda^* \in \mathbb{R}^m$ such that the* Karush–Kuhn–Tucker (KKT) condition

$$\nabla_x L(x^*, \lambda^*) = \nabla f(x^*) + \sum_{j=1}^{m} \lambda_j^* \nabla c_j(x^*) = 0 \tag{5.2}$$

hold. The pair (x^*, λ^*) is called a primal–dual solution *of* (5.1).

Example 5.1.3. Consider the function $c\colon \mathbb{R}^2 \to \mathbb{R}$, $c(x_1, x_2) = x_2^2 - x_1^3$. Then the set $C = \big\{(x_1, x_2) \in \mathbb{R}^2 : c(x_1, x_2) = 0\big\}$ consists of those pairs (x_1, x_2) satisfying $x_1 \ge 0$ and $x_2 = \pm x_1^{3/2}$. This set has a singularity at the point $(0,0)$, although the function c is everywhere differentiable. At $(0,0)$, however, we have $\nabla c(x_1, x_2) = (0,0)$, which is not linearly independent.[1] If $f\colon \mathbb{R}^2 \to \mathbb{R}$ is any function whose minimum on C is attained at $(0,0)$, then a Lagrange multiplier can exist only if $(0,0)$ is already a stationary point of f. For instance the function $f\colon \mathbb{R}^2 \to \mathbb{R}$, $f(x_1, x_2) = x_1$ attains its unique minimum on C at the point $(0,0)$, but $\nabla f(0,0) = (1,0)$. ∎

Example 5.1.4. Consider the functions $c_j\colon \mathbb{R}^2 \to \mathbb{R}$, $i = 1, 2$,

$$c_1(x_1, x_2) = x_1^2 + x_2^2 - 1\,, \qquad c_2(x_1, x_2) = (x_1 - 2)^2 + x_2^2 - 1\,.$$

The zero set of the first function is a circle of radius one centred at $(0,0)$, the zero set of the second function a circle of radius one centred at $(2,0)$. In particular, the set C defined by the functions c_1 and c_2 consists of only the point $(1,0)$. However, we have $\nabla c_1(1,0) = (2,0)$, whereas $\nabla c_2(1,0) = (-2,0)$, which shows that the two gradients are not linearly independent; the linear space spanned by these vectors is only $\mathbb{R}(1,0)$.

Now let $f\colon \mathbb{R}^2 \to \mathbb{R}$ be any function. Then its minimum on C obviously has to be attained at $(1,0)$, the only point in C. A Lagrange multiplier for the minimisation of f on C, however, can only exist, if $\nabla f(1,0)$ is an element of $\mathbb{R}(1,0)$, which need not be the case. For instance, the function $f(x_1, x_2) = x_2$ has the gradient $\nabla f(1,0) = (0,1)$, which cannot be written as a linear combination of $\nabla c_1(1,0)$ and $\nabla c_2(1,0)$, showing that in this case no Lagrange multipliers can exist. ∎

[1] Recall that a set $\{v_1, \dots, v_k\}$ of vectors is linearly independent, if the condition $\sum_j \lambda_j v_j = 0$ implies that $\lambda_j = 0$ for every j; else it is called linearly dependent. In the case of a set $\{v_1\}$ containing the single element v_1, linear independence means that $\lambda_1 v_1 = 0$ implies that $\lambda_1 = 0$. This holds true, if and only if $v_1 \neq 0$. Thus the set $\{v_1\}$ is linearly independent if $v_1 \neq 0$, but linearly dependent if $v_1 = 0$.

In the following, we will always assume that the conditions of Proposition 5.1.2 are satisfied, that is, the vectors $\nabla c_j(x^*)$ are linearly independent. Denote now for ease of notation by $c\colon \mathbb{R}^n \to \mathbb{R}^m$ the vector valued function whose components are the constraints c_j. Then the last proposition states that we should search for solutions $(x^*, \lambda^*) \in \mathbb{R}^n \times \mathbb{R}^m$ of the system

$$\begin{aligned} \nabla_x L(x^*, \lambda^*) &= 0\,, \\ c(x^*) &= 0\,, \end{aligned} \tag{5.3}$$

in order to find local minimisers of f on C, or, equivalently, that we should search for stationary points of the Lagrangian $L(x, \lambda)$.

The solution of (5.3) can for instance be found by Newton's method for the solution of non-linear equations. That is, we iteratively linearise (5.3) and solve the ensuing linear system for the primal–dual pair (x, λ). Let $\nabla c\colon \mathbb{R}^n \to \mathbb{R}^{n\times m}$ be the matrix of the derivatives of c, that is,

$$\big(\nabla c(x)\big)_{ij} = \partial_i c_j(x)\ .$$

Moreover, denote by $\nabla^2_{xx} L\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{n\times n}$ the Hessian of the Lagrangian with respect to the x variable, which equals

$$\nabla^2_{xx} L(x, \lambda) = H_f(x) + \sum_{j=1}^m \lambda_j H_{c_j}(x)\ .$$

Then Newton's method consists in choosing some initial values $x^{(1)} \in \mathbb{R}^n$ and $\lambda^{(1)} \in \mathbb{R}^m$ and inductively setting

$$x^{(k+1)} := x^{(k)} + d^{(k)}\,, \qquad\qquad \lambda^{(k+1)} := \lambda^{(k)} + \mu^{(k)}\,,$$

where the pair $(d^{(k)}, \mu^{(k)}) \in \mathbb{R}^n \times \mathbb{R}^m$ solves the system

$$\begin{pmatrix} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) & \nabla c(x^{(k)}) \\ \nabla c(x^{(k)})^T & 0 \end{pmatrix} \begin{pmatrix} d^{(k)} \\ \mu^{(k)} \end{pmatrix} = - \begin{pmatrix} \nabla_x L(x^{(k)}, \lambda^{(k)}) \\ c(x^{(k)}) \end{pmatrix}\ . \tag{5.4}$$

Now recall that

$$\nabla_x L(x^{(k)}, \lambda^{(k)}) = \nabla f(x^{(k)}) + \nabla c(x^{(k)})\lambda^{(k)}\ .$$

Moreover,

$$\begin{pmatrix} \nabla c(x^{(k)})\lambda^{(k)} \\ 0 \end{pmatrix} = \begin{pmatrix} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) & \nabla c(x^{(k)}) \\ \nabla c(x^{(k)})^T & 0 \end{pmatrix} \begin{pmatrix} 0 \\ \lambda^{(k)} \end{pmatrix}\ .$$

Inserting these expressions in (5.4) we obtain the equivalent formulation

$$\begin{pmatrix} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) & \nabla c(x^{(k)}) \\ \nabla c(x^{(k)})^T & 0 \end{pmatrix} \begin{pmatrix} d^{(k)} \\ \lambda^{(k)} + \mu^{(k)} \end{pmatrix} = - \begin{pmatrix} \nabla_x f(x^{(k)}) \\ c(x^{(k)}) \end{pmatrix}\,,$$

or, using the fact that $\lambda^{(k)} + \mu^{(k)} = \lambda^{(k+1)}$,

$$\begin{pmatrix} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) & \nabla c(x^{(k)}) \\ \nabla c(x^{(k)})^T & 0 \end{pmatrix} \begin{pmatrix} d^{(k)} \\ \lambda^{(k+1)} \end{pmatrix} = - \begin{pmatrix} \nabla_x f(x^{(k)}) \\ c(x^{(k)}) \end{pmatrix}\ .$$

Data: a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$;
a initial guesses x_{init} and λ_{init};
constraints $c\colon \mathbb{R}^n \to \mathbb{R}^m$
Result: a primal–dual pair $(x^*, \lambda^*) \in \mathbb{R}^n \times \mathbb{R}^m$;

Initialisation: set $x^{(1)} := x_{\text{init}}$, $\lambda^{(1)} := \lambda_{\text{init}}$, $k = 1$;

while *convergence criterion not yet satisfied* **do**
  define $d^{(k)}$ and $\lambda^{(k+1)}$ by solving the equation

$$\begin{pmatrix} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) & \nabla c(x^{(k)}) \\ \nabla c(x^{(k)})^T & 0 \end{pmatrix} \begin{pmatrix} d^{(k)} \\ \lambda^{(k+1)} \end{pmatrix} = - \begin{pmatrix} \nabla_x f(x^{(k)}) \\ c(x^{(k)}) \end{pmatrix}$$

  where $L(x, \lambda) = f(x) + \lambda^T c(x)$;
  define $x^{(k+1)} := x^{(k)} + d^{(k)}$;
  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$ and $\lambda^* := \lambda^{(k)}$;

Algorithm 13: Newton's method for problems with equality constraints.

Thus one sees that the Lagrange multiplier plays a far less important role than the primal variable x. In Newton's algorithm, it basically only appears in the term $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$.

See Algorithm 13 for a summary of Newton's method.

Note that the linear system to be solved in each step of the constrained Newton method has a completely different character from those solved in the unconstrained case. In the latter situation, we could expect the matrix describing the system to be positive definite in a neighbourhood of the solution, which allowed the application of the CG method for its solution, but also a Cholesky or LDL-decomposition. In the constrained setting, in contrast, the matrix *cannot* be positive definite. Also, it can only be positive semi-definite, if $\nabla c(x^{(k)}) = 0$, a situation that we have excluded *a–priori*. Thus, neither CG nor Cholesky are possible (to be precise, CG is *possible*, but not necessarily convergent).

5.1.1 Line Search

Similarly as Newton's method for unconstrained problems, also the constrained version enjoys local quadratic convergence (both to the primal solution x^* and the dual solution λ^*). On the other hand, it also shares all the problems of the unconstrained method, in particular its tendency to diverge if the initial guess is not already quite close to the actual solution. Here, this is even a larger problem than in the unconstrained case: While it is often possible to choose a good initial primal value $x^{(1)}$ (which need not satisfy the constraints), finding a good initial guess for the multiplier $\lambda^{(1)}$ can be difficult, because an interpretation of the multiplier often is not readily available.[2] Thus, it seems reasonable to combine the method with some line search algorithm. In the following, a method is described that performs a line search only for the update of the primal variable

[2]Note, however, that in certain economic situations, where the constraints describe bounds on the availability of certain specified goods, it is possible to interpret the corresponding Lagrange multipliers as fair prices of said goods.

$x^{(k)}$, while the (less important) dual variable $\lambda^{(k)}$ is updated as in the original algorithm.

When trying to perform a line search for a constrained minimisation problem, one faces a major difficulty: All line search algorithms are based on the fact that one aims for the minimisation of some functional on $\mathbb{R}^n$. Here this is not the case, as we have the additional restriction that the minimiser x^* should satisfy $c(x^*) = 0$. Put differently, the system (5.3) we try to solve is not the optimality condition of some function $g\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$. Even though we only perform a line search for the update of the x variable, we cannot simply use the cost function f, because it cannot tell us anything about the closeness to the set C, which we require. In order to use a line search algorithm nevertheless, we have to artificially introduce some cost or merit function $G\colon \mathbb{R}^n \to \mathbb{R}$ that tells us whether some proposed step size is adequate. This approach, however, can only work, if the merit function has the same local minima as the original constrained problem or, at least, if every original solution is also a minimiser of G.

Definition 5.1.5. Let x^* be a local minimiser of the constrained problem (5.1). A function $G\colon \mathbb{R}^n \to \mathbb{R}$ is an *exact penalty function at* x^*, if x^* is a local minimiser of G. ∎

One possibility for a merit function is to use, for some $\sigma > 0$, the function $G_\sigma\colon \mathbb{R}^n \to \mathbb{R}$ defined as

$$G_\sigma(x) := f(x) + \sigma|c(x)|\,, \tag{5.5}$$

where $|\cdot|$ denotes the Euclidean norm on $\mathbb{R}^m$. Thus G_σ at the same time penalises a large cost function f and deviations from the set C (through the term $|c(x)|$). Then the following result holds:

Lemma 5.1.6. *Assume that the pair (x^*, λ^*) is a primal-dual solution of* (5.3) *and $\nabla^2_{xx} L(x^*, \lambda^*) > 0$. If $\sigma > |\lambda^*|$, then x^* is a local minimiser of G_σ. In other words, in this case the penalty function G_σ is exact at x^*.*

Assuming that we are already close to a primal-dual solution of the minimisation problem we want to solve, the last result indicates that we should choose the parameter σ in such way that it is larger than the norm of the current Lagrange multiplier. Because the size of the Lagrange multiplier will usually change during the iteration, this also means that the parameter σ might also change over the iterations. In order to apply the line search algorithms of Chapter 2, we still need the gradient of the merit function G_σ. We have

$$\nabla G_\sigma(x) = \nabla f(x) + \sigma \nabla c(x)\, \frac{c(x)}{|c(x)|}\,.$$

A sketch of resulting method is given in Algorithm 14.

There are still some issues with Algorithm 14. For one, no indication is given how one should choose the parameters $\sigma^{(k)}$. Theoretical convergence results can for instance be derived under the conditions that $\sigma^{(k)} > |\lambda^{(k+1)}| + \hat{\sigma}$ for some fixed $\hat{\sigma} > 0$ independent of the index k, that $\sigma^{(k)} \geq \sigma^{(k-1)}$ for all k and the values of $\sigma^{(k)}$ only change a finite number of times. In practise, however, it will be better if the parameters are also allowed to decrease; else the convergence may turn out to be overly slow.

Data: a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$;
a initial guesses x_{init} and λ_{init};
constraints $c\colon \mathbb{R}^n \to \mathbb{R}^m$
Result: a primal–dual pair $(x^*, \lambda^*) \in \mathbb{R}^n \times \mathbb{R}^m$;

Initialisation: set $x^{(1)} := x_{\text{init}}$, $\lambda^{(1)} := \lambda_{\text{init}}$, $k = 1$;

while *convergence criterion not yet satisfied* **do**
  define $d^{(k)}$ and $\lambda^{(k+1)}$ by solving the equation

$$\begin{pmatrix} \nabla^2_{xx} L(x^{(k)}, \lambda^{(k)}) & \nabla c(x^{(k)}) \\ \nabla c(x^{(k)})^T & 0 \end{pmatrix} \begin{pmatrix} d^{(k)} \\ \lambda^{(k+1)} \end{pmatrix} = - \begin{pmatrix} \nabla_x f(x^{(k)}) \\ c(x^{(k)}) \end{pmatrix}$$

  where $L(x, \lambda) = f(x) + \lambda^T c(x)$;
  choose some $\sigma^{(k)} > |\lambda^{(k+1)}|$;
  find some suitable step-size $t^{(k)}$ by performing a line search in direction $d^{(k)}$ for the merit function $G_{\sigma^{(k)}}$ with initialisation $t = 1$;
  define $x^{(k+1)} := x^{(k)} + t^{(k)} d^{(k)}$;
  $k \leftarrow k + 1$;
end
define $x^* := x^{(k)}$ and $\lambda^* := \lambda^{(k)}$;

Algorithm 14: Newton's method with line search for problems with equality constraints.

Second, $d^{(k)}$ need not necessarily be a descent direction for $G_{\sigma^{(k)}}$. In the special case where $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$ is positive definite and $\sigma^{(k)} > |\lambda^{(k+1)}|$, however, it is. While the positive definiteness can usually not be guaranteed, it is possible to replace the matrix $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$ by some positive definite approximation $M^{(k)}$ similarly as for quasi-Newton methods and compute the direction $d^{(k)}$ using this approximation $M^{(k)}$. Then at least the descent property of the direction $d^{(k)}$ holds. There is, however, no canonical method for the definition of the matrices $M^{(k)}$. The problem of finding practicable constructions is still a matter of ongoing research.

Finally, it may happen that the line search, though yielding a larger neighbourhood around the solution (x^*, λ^*) in which the method converges, destroys the quadratic convergence of the unmodified Newton method. It may happen that the step size $t^{(k)} = 1$, which should be optimal sufficiently close to the solution, might be rejected by any sensible line search. The problem is that the search direction and the merit function are not directly related. One possible remedy is to modify the merit function and use, for some $\mu \in \mathbb{R}^m$ and $\sigma > 0$, the function

$$G_{\mu,\sigma} := f(x) + \mu^T c(x) + \sigma |c(x)| \,. \tag{5.6}$$

If μ is sufficiently close to λ^* and $\sigma > |\lambda^* - \mu|$, also the penalisation with $G_{\mu,\sigma}$ is exact at x^* and $d^{(k)}$ is a descent direction if $\nabla^2_{xx} L(x^{(k)}, \lambda^{(k)})$ (or $M^{(k)}$) is positive definite and $\sigma \geq |\lambda^{(k+1)} - \mu|$. Moreover, the step size $t = 1$ will be acceptable for a line search with Armijo's rule provided that $0 < m_1 < 1/2$.

A different possibility is to use the *augmented Lagrangian*, defined as

$$\tilde{G}_{\mu,\sigma} := f(x) + \mu^T c(x) + \frac{\sigma}{2} |c(x)|^2 \tag{5.7}$$

as merit function. This function is exact at x^*, if $\mu = \lambda^*$ and σ is sufficiently large. In contrast to (5.6), however, exactness fails to hold, if μ does not coincide with the optimal Lagrange parameter λ^*.

Remark 5.1.7. The method described above strictly separates the definition of the search direction (which uses the KKT conditions) and the objective function for the line search. In principle, however, one could also use a unified approach, where search direction and step size are both defined by any of the functions defined in (5.5), (5.6), and (5.7). While at first glance this seems reasonable, this approach also has its problems. The functions in (5.5) and (5.6) are differentiable only outside of the set C and therefore Newton's method most probably will not converge quadratically. In the case of the augmented Lagrangian, which is as smooth as the functions f and c, a different problem arises: Exactness only holds, if the parameter μ equals the unknown Lagrangian λ^*. Thus, the augmented Lagrangian only works if one changes the parameter μ, and therefore the function to be minimised, during the algorithm. Also, the parameter σ has to be large in order to ensure exactness, which can slow down the convergence of the method. ■

5.2 Equality and Inequality Constraints

The situation is considerably more difficult, if we have not only equality constraints, but also inequality constraints. That is, we have to solve

$$f(x) \to \min \qquad \text{subject to} \qquad \begin{cases} c_j(x) = 0 & \text{for } j \in E\,, \\ c_j(x) \le 0 & \text{for } j \in I\,, \end{cases} \tag{5.8}$$

where the functions c_j, $j \in E$, describe the equality constraints and the functions c_j, $j \in I$, the inequality constraints. The equality constraints can be treated in precisely the same manner as in the setting discussed in Section 5.1. The inequality constraints, however, require more consideration.

Assume now that x^* is a local solution of (5.8) and $j \in I$. Then either we have $c_j(x^*) = 0$ (the constraint is *active* at x^*) or $c_j(x^*) < 0$ (the constraint is *inactive* at x^*). In the latter case, the condition $c_j(x) \le 0$ is satisfied in a whole neighbourhood of x^* and thus can (locally) be discarded. On the other hand, if $c_j(x^*) = 0$, then the derivative of the cost function f in direction $\nabla c_j(x^*)$ may be negative without violating the local minimality of f.

Denote in the following by

$$I^0(x^*) = \big\{ j \in I : c_j(x^*) = 0 \big\}$$

the set of active constraints at x^*. Then the considerations above show that the gradient of f at x^* may have the form

$$\nabla f(x^*) = - \sum_{j \in E \cup I^0} \lambda_j \nabla c_j(x^*)$$

with $\lambda_j \ge 0$ whenever $j \in I^0$.

The next result shows that, for *qualified* constraints, such a representation of $\nabla f(x^*)$ is indeed a necessary condition for a local solution of the restricted

minimisation problem. Again, we define the Lagrangian $L\colon \mathbb{R}^n \times \mathbb{R}^{E\cup I} \to \mathbb{R}$ as

$$L(x,\lambda) := f(x) + \sum_{j\in E\cup I} \lambda_j c_j(x)\,.$$

Proposition 5.2.1. *Assume that x^* is a local minimiser of* (5.8) *and that f and the constraints c_j are C^2. If the vectors $\nabla c_j(x^*)$, $j \in E \cup I^0$, are linearly independent, then there exists $\lambda^* \in \mathbb{R}^{E\cup I}$ such that the KKT-condition*

$$\left.\begin{aligned} \nabla f(x^*) + \sum_{j\in E\cup I} \lambda_j^* \nabla c_j(x^*) &= 0\,,\\ c_j(x^*) &= 0 \quad \textit{for } j \in E\,,\\ c_j(x^*) &\le 0 \quad \textit{for } j \in I\,,\\ \lambda_j^* &\ge 0 \quad \textit{for } j \in I^0\,,\\ \lambda_j^* &= 0 \quad \textit{for } j \in I \setminus I^0 \end{aligned}\right\} \tag{5.9}$$

holds. The pair (x^*,λ^*) is called a primal–dual *solution of* (5.8).

An equivalent way of writing this system, is the following set of equalities and inequalities, which hides the set of active constraints in the last equation:

$$\left.\begin{aligned} \nabla f(x^*) + \sum_{j\in E\cup I} \lambda_j^* \nabla c_j(x^*) &= 0\,,\\ c_j(x^*) &= 0 \quad \text{for } j \in E\,,\\ c_j(x^*) &\le 0 \quad \text{for } j \in I\,,\\ \lambda_j^* &\ge 0 \quad \text{for } j \in I\,,\\ \sum_{j\in I} \lambda_j^* c_j(x^*) &= 0\,. \end{aligned}\right\} \tag{5.10}$$

Because $\lambda_j^* \ge 0$ and $c_j(x^*) \le 0$ for every $j \in I$, the terms $\lambda_j^* c_j(x^*)$ are all non-positive. Thus the sum can only be equal to zero, if every one of them is. In particular this implies that $\lambda_j^* = 0$ whenever $c_j(x^*) < 0$, which is precisely the last condition in (5.9).

In order to formulate a Newton like method for solving (5.10), we rewrite the equality and inequality conditions for x^* in such a way that they appear as mere equalities. To that end, we define for each vector $v \in \mathbb{R}^{E\cup I}$ the vector $v^\# \in \mathbb{R}^{E\cup I}$ whose components are

$$(v^\#)_j := \begin{cases} v_j & \text{if } j \in E\,,\\ \max\{v_j, 0\} & \text{if } j \in I\,. \end{cases}$$

With this definition, one can write (5.10) as

$$\left.\begin{aligned} \nabla_x L(x^*,\lambda^*) &= 0\,,\\ c(x^*)^\# &= 0\,,\\ \lambda_j^* &\ge 0 \quad \text{for } j \in I\,,\\ \sum_{j\in I} \lambda_j^* c_j(x^*) &= 0\,. \end{aligned}\right\} \tag{5.11}$$

Because we have an inequality constraint and the mapping $c^\#$ is not differentiable, Newton's method should not work for the system (5.11). We apply it nevertheless. That is, we consider the iteration $x^{(k+1)} = x^{(k)} + d^{(k)}$ and $\lambda^{(k+1)} = \lambda^{(k)} + \mu^{(k)}$, where the steps $d^{(k)} \in \mathbb{R}^n$ and $\mu^{(k)} \in \mathbb{R}^{E\cup I}$ are obtained by linearising (5.11) whenever possible. Carrying out this linearisation (whenever possible), we obtain the system

$$\begin{aligned}
\nabla_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)} + \nabla c(x^{(k)}) \mu^{(k)} &= -\nabla_x L(x^{(k)}, \lambda^{(k)}) \,, \\
\big(c(x^{(k)}) + \nabla c(x^{(k)}) d^{(k)}\big)^\# &= 0 \,, \\
\lambda_j^{(k)} + \mu_j^{(k)} &\geq 0 \qquad\qquad \text{for } j \in I \,, \\
\sum_{j\in I} (\lambda_j^{(k)} + \mu_j^{(k)}) c_j(x^{(k)}) + \sum_{j\in I} \lambda_j^{(k)} \nabla c_j(x^{(k)}) d_j^{(k)} &= 0 \,.
\end{aligned}$$

While this system is quite difficult to solve, it is possible to approximate it by a system with better properties. To that end, we add to the left hand side of the last equation the term $\sum_{j\in I} \mu_j^{(k)} \nabla c_j(x^{(k)}) d_j^{(k)}$. This is reasonable if we are already close to the solution, because then the steps $\mu^{(k)}$ and $d^{(k)}$ will be small, and this new term is the only one that is quadratic in the update, all other terms being at most linear. Then we can replace the terms $\lambda^{(k)} + \mu^{(k)}$ by $\lambda^{(k+1)}$ and obtain, similar as in the case with only equality constraints, the system

$$\begin{aligned}
\nabla_{xx} L(x^{(k)}, \lambda^{(k)}) d^{(k)} + \nabla c(x^{(k)}) \lambda^{(k+1)} &= -\nabla f(x^{(k)}) \,, \\
\big(c(x^{(k)}) + \nabla c(x^{(k)}) d^{(k)}\big)^\# &= 0 \,, \\
\lambda_j^{(k+1)} &\geq 0 \qquad\qquad \text{for } j \in I \,, \\
\sum_{j\in I} \lambda_j^{(k+1)} c_j(x^{(k)}) + \sum_{j\in I} \lambda_j^{(k+1)} \nabla c_j(x^{(k)}) d_j^{(k)} &= 0 \,.
\end{aligned}$$

The solutions of this system is at the same time the primal–dual solutions of the *quadratic programme*

$$\nabla f(x^{(k)})^T d + \frac{1}{2} d^T \nabla_{xx} L(x^{(k)}, \lambda^{(k)}) d \to \min$$

subject to

$$\begin{aligned}
c_j(x^{(k)}) + \nabla c_j(x^{(k)})^T d &= 0 \quad \text{for } j \in E \,, \\
c_j(x^{(k)}) + \nabla c_j(x^{(k)})^T d &\leq 0 \quad \text{for } j \in I \,.
\end{aligned}$$

For problems of this kind, several solution algorithms exist.

Chapter 6

Interior Point Methods

The idea of interior point methods is to solve a constrained minimisation problem $f(x) \to \min$ subject to $x \in C$ by approaching the true minimiser, which will usually lie on the boundary of C, from the interior. To that end, one introduces a *barrier function* $g\colon \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ such that g is finite precisely in the interior of C and $g(x)$ tends to infinity as x approaches the boundary of C. Then, for any given parameter $\eta > 0$, one can minimise the function $\eta f + g$ over $\mathbb{R}^n$ and obtains some point $x^*_\eta \in C$. As η increases, these points should form a continuous curve, which, because more and more emphasis is put on the function f, should hopefully converge to the minimiser x^* of the constrained minimisation problem. In this chapter, we will shortly discuss one family of problems, where such an approach provably works and also can be quite efficient: linear programs.

6.1 Barrier Functions

Definition 6.1.1. Let $g\colon \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a convex function such that the *domain* $\mathcal{D}_g$ of g, that is, the set of points $x \in \mathbb{R}^n$ with $g(x) < \infty$, is non-empty and open, and assume that g is three times differentiable on $\mathcal{D}_g$. Define for every x, $d \in \mathbb{R}^n$ the restricted function $g_{x,d}\colon \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$ by $g_{x,d}(t) := g(x + td)$. The function g is called *self-concordant*, if

$$|g'''_{x,d}(t)| \leq 2(g''_{x,d}(t))^{3/2}$$

whenever $x+td \in \mathcal{D}_g$, and $\lim_{k\to\infty} g(x_k) = +\infty$ whenever a sequence $\{x_k\}_{k\in\mathbb{N}} \subset \mathcal{D}_g$ converges to a point in the boundary of $\mathcal{D}_g$.

The function g is a *barrier function*, if it is self-concordant, locally elliptic, and

$$\theta_g := \sup_{x\in\mathcal{D}_g} \left\langle \nabla g(x), H_g(x)^{-1} \nabla g(x) \right\rangle < \infty \,. \qquad \blacksquare$$

The most important barrier function is the *logarithmic barrier function* $g\colon \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$,

$$g(x) := -\sum_{j=1}^{n} \ln(x_j) \,,$$

with domain $\mathcal{D}_g = \{x \in \mathbb{R}^n : x > 0\}$. Its gradient at $x \in \mathcal{D}_g$ is the vector with entries $-1/x_j$ and its Hessian the diagonal matrix with diagonal entries $1/x_j^2$. Therefore we have

$$\langle \nabla g(x), H_g(x)^{-1} \nabla g(x) \rangle = n$$

for every $x \in \mathcal{D}_g$, and hence

$$\theta_g = n \,.$$

Definition 6.1.2. Assume that g is a barrier function and $\mathcal{D}_g$ is bounded. Then the (necessarily unique) minimiser of g is called the *analytic centre* for g. ∎

6.2 Barrier Methods

In the following, we consider the solution of linear programs of the form

$$\langle c, x \rangle \to \min \qquad \text{subject to } Ax \leq b \,, \tag{6.1}$$

where $c \in \mathbb{R}^n$ is some cost vector, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ with $m > n$ describe the linear constraints, and the inequality $Ax \leq b$ is understood componentwise. In addition, we assume that the set $\mathcal{P}(A, b) := \{x \in \mathbb{R}^n : Ax \leq b\}$ is non-empty and bounded, implying that (6.1) attains a solution for every $c \in \mathbb{R}^n$.

Now let $g \colon \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be any barrier function with

$$\mathcal{D}_g = \{x \in \mathbb{R}^n : Ax < b\} \,.$$

A typical choice for such a barrier function would be

$$g(x) = -\sum_{j=1}^{m} \ln\bigl((b - Ax)_j\bigr) \,, \tag{6.2}$$

in which case the inequality

$$\theta_g \leq m$$

holds.

The algorithms presented in the following try to follow the *central path*, which is the path in $\mathbb{R}^n$ consisting of the minimisers x_η^* of the functionals

$$g_\eta(x) := \eta \langle c, x \rangle + g(x) \,.$$

It starts for $\eta = 0$ at the analytic centre of g and converges for $\eta \to \infty$ to the solution of the linear programme (6.1). Moreover, one can show that

$$\langle c, x_\eta^* \rangle \leq \frac{\theta_g}{\eta} + \inf\{\langle c, x \rangle : Ax \leq b\} \,.$$

Thus all the elements of the central path solve the linear programme (6.1) approximately up to a value of θ_g / η.

6.2.1 Short-step Method

The short-step method follows the central path by alternately performing a Newton step for the functional g_η and increasing the parameter η by a small amount. That is, one starts with some parameter $\eta > 0$ and an element x that is sufficiently close to x^*_η. Then one multiplies η with some $\beta > 1$ obtaining a new parameter $\tilde\eta = \beta\eta$, defines the next iterate $\tilde x$ by performing a Newton step for the function $g_{\tilde\eta}$, and repeats the procedure.

This approach will work if, first, one can make precise what "sufficiently close to x^*_η" means, and, second, one can find some condition on β that will guarantee that $\tilde x$ again is sufficiently close to $x^*_{\tilde\eta}$. The point of barrier functions (or, more general, self-concordant functions) is that this is indeed possible. Moreover, the condition one obtains depends only on the value θ_g.

In order to simplify the notation, we define for $x \in \mathcal{D}_g$ the norm

$$\|z\|_x := \sqrt{\langle z, H_g(x)\, z\rangle}\,. \tag{6.3}$$

Moreover, note that the gradient of g_η at $x \in \mathcal{D}_g$ equals $\eta c + \nabla g(x)$ and the Hessian is simply the Hessian $H_g(x)$ of g.

Proposition 6.2.1. *Let $x \in \mathcal{D}_g$ and $\eta > 0$. Assume that*

$$1 - \frac{1}{8\sqrt{\theta_g}} \le \beta \le 1 + \frac{1}{8\sqrt{\theta_g}}\,,$$

and denote by $\tilde x$ is the result of a Newton step for $g_{\beta\eta}$ starting at x. That is, $\tilde x = x + d$, where d solves the system

$$H_g(x)\, d = -\beta\eta c - \nabla g(x)\,.$$

If

$$\|x - x^*_\eta\|_x \le \frac{1}{6}$$

then also

$$\|\tilde x - x^*_{\beta\eta}\|_x \le \frac{1}{6}\,.$$

Moreover,

$$\langle c, x\rangle \le \frac{6}{5}\frac{\theta_g}{\eta} + \inf\bigl\{\langle c, x\rangle : Ax \le b\bigr\}\,.$$

While this result gives us a good criterion for how to increase η, the closeness condition $\|x - x^*_\eta\|_x \le 1/6$ is not that useful, as it depends on the unknown point x^*_η. The following result, however, provides a criterion only depending on x.

Lemma 6.2.2. *Let $x \in \mathcal{D}_g$ satisfy*

$$\bigl\|H_g(x)^{-1}\,\nabla g(x)\bigr\|_x = \bigl\langle \nabla g(x), H_g(x)^{-1}\,\nabla g(x)\bigr\rangle \le \frac{1}{36}$$

and let

$$\eta = \frac{1}{12\|H_g(x)^{-1}c\|_x} = \frac{1}{12\sqrt{\langle c, H_g(x)^{-1}c\rangle}}\,.$$

Then

$$\|x - x^*_\eta\|_x \le \frac{1}{6}\,.$$

It remains to find some $x \in \mathcal{D}_g$ that satisfies the condition of the previous lemma. Then this point can be used for starting the sequence of Newton steps and updates of the parameter η. Note that the condition is in particular satisfied at the analytic centre of g, where the gradient of g vanishes. Thus we only have to find any point sufficiently close to the analytic centre.

To that end, assume that $\tilde{x}$ is any point in $\mathcal{D}_g$. Then $\tilde{x}$ minimises the functional

$$x \mapsto -\big\langle \nabla g(\tilde{x}), x \big\rangle + g(x) \ .$$

Put differently, the point $\tilde{x}$ lies on the central path for the barrier function g and the cost function $x \mapsto -\big\langle \nabla g(\tilde{x}), x \big\rangle$. Denote therefore, for $\nu \geq 0$, by $\tilde{x}^*_\nu$ the minimiser of the functional

$$\tilde{g}_\nu(x) := -\nu \big\langle \nabla g(\tilde{x}), x \big\rangle + g(x) \ .$$

Then $\tilde{x}^*_1 = \tilde{x}$. Moreover, the path leads for $\nu \to 0$ to the analytic centre of g. In order to trace back the path to its starting point, we can use use the same idea as for finding the minimiser of the linear programme, but instead of increasing the parameter ν before each Newton step, we decrease it by a small amount. Again, Proposition 6.2.1 shows that decreasing by a factor of at most $1 - 1/(8\sqrt{\theta_g})$ will be possible.

Data: $c \in \mathbb{R}^n$, a barrier function g, a point $x_{\text{init}} \in \mathcal{D}_g$, a stopping parameter $\eta_{\max}$;
Result: $x^* \in \mathbb{R}^n$;

Initialisation: set $x := x_{\text{init}}$;

define $\nu := 1$;
define $\xi := \nabla g(x)$;
repeat
  replace $\nu \leftarrow \nu - \nu/(8\sqrt{\theta_g})$;
  define d by solving the equation

$$H_g(x)\, d = \nu\xi - \nabla g(x) \ ;$$

  replace $x \leftarrow x + d$;
until $\langle \nu\xi - \nabla g(x), d \rangle > 1/36$;
define

$$\eta = \Big(12\sqrt{\langle c, H_g^{-1}(x)c \rangle}\Big)^{-1} \ ;$$

repeat
  replace $\eta \leftarrow \eta + \eta/(8\sqrt{\theta_g})$;
  define d by solving the equation

$$H_g(x)\, d = -\eta c - \nabla g(x)$$

  replace $x \leftarrow x + d$;
until $\eta > \eta_{\max}$;
define $x^* := x$;

Algorithm 15: Short-step barrier method.

Combining the results stated above, we arrive at Algorithm 15, the *short-step barrier method*. One can show that it needs $O\big(\sqrt{\theta_g}\ln(\theta_g/\varepsilon)\big)$ steps in order to obtain a result whose value is at most by ε larger than the optimal value of the linear programme. Note that the precise value of θ_g need not be known for the application of the method. Instead, it is sufficient to have knowledge of an upper bound τ for θ_g and then increase and decrease the values of η and ν by factors of $1 + 1/(8\sqrt{\tau})$ and $1 - 1/(8\sqrt{\tau})$, respectively. For instance, in the case of the linear constraints $Ax \leq b$ with $b \in \mathbb{R}^m$ and the logarithmic barrier function (6.2), one can use the estimate $\theta_g \leq m$.

6.2.2 Long-step Method

The main problem of the short-step method is the very small increase of the parameter η. For linear constraints $Ax \leq b$ with $b \in \mathbb{R}^m$ and the logarithmic barrier function (6.2), one will typically increase η only by the factor $1 + 1/(8\sqrt{m})$, which leads to a huge number of iterations if the number of constraints is large.

One is therefore tempted to try to increase η by a much larger amount $\beta \gg 1$ in each iteration. Then, however, one cannot expect to obtain an iterate that is again close to the central path. Because the element $x^*_{\beta\eta}$ on the central path minimises the functional $g_{\beta\eta}$, it is possible to use the Newton method in order to return sufficiently close to the central path. Again one can use the norm $\|\cdot\|_x$ defined in (6.3) for determining when to stop the inner Newton iteration. For instance, the criterion $\|d\|_x \leq 1/4$, where d denotes the Newton direction, yields a convergent method.

There is, however, an additional complication. If one starts the Newton iteration far from the central path, then one cannot expect it to converge. In fact, one cannot even guarantee that the Newton iterates stay within the domain of g. Thus it is necessary to combine the Newton iteration with a line search, in order to obtain a convergent method. Theoretically, the resulting method is worse than the short-step method by a factor of $\sqrt{\theta_g}$; in practise, one can expect that it converges considerably faster.