Numerical Methods and Stochastics Part I: Numerical Methods

Lecture Notes Summerterm 2011

R. Verfürth

Fakultät für Mathematik, Ruhr-Universität Bochum

Contents

CHAPTER I

Boundary Value Problems for Ordinary Differential Equations

I.1. Initial Value Problems

In this and the following section we recall several properties of initial value problems for ordinary differential equations and of their numerical approximation. For further details we refer to [3].

I.1.1. Initial value problems for first order ordinary differential equations. Given an interval I, a subset D of $\mathbb{R}^d$, a function $f(t, y) : I \times D \to \mathbb{R}^d$, an *initial time* $t_0 \in I$ and an *initial value* $y_0 \in D$, an *initial value problem* consists in finding a differentiable function $y : I \to D$ such that

$$\boxed{\begin{aligned} &y'(t) = f(t, y(t)) \quad \text{for all } t \in I \quad &\textit{(differential equation)}\\ &y(t_0) = y_0 &\textit{(initial condition)}\end{aligned}}$$

Example I.1.1. Assume that the function y describes the size at time t of a population with constant death or birth rate λ. Then y solves the initial value problem

$$\begin{aligned} y'(t) &= \lambda y(t),\\ y(0) &= c,\end{aligned}$$

where

$$I = \mathbb{R},\ D = \mathbb{R},\ f(t, y) = \lambda y,\ t_0 = 0,\ y_0 = c.$$

The exact solution is

$$y(t) = ce^{\lambda t}.$$

Example I.1.2. The initial value problem

$$\begin{aligned} y'(t) &= \begin{pmatrix} \lambda & -\omega \\ \omega & \lambda \end{pmatrix} y(t)\\ y(0) &= \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}\end{aligned}$$

describes a *damped oscillation*. Here we have

$$I = \mathbb{R},\ D = \mathbb{R}^2,\ f(t, y) = \begin{pmatrix} \lambda & -\omega \\ \omega & \lambda \end{pmatrix} y,\ t_0 = 0,\ y_0 = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}.$$

The exact solution is

$$y(t) = e^{\lambda t} \begin{pmatrix} c_1 \cos(\omega t) - c_2 \sin(\omega t) \\ c_1 \sin(\omega t) + c_2 \cos(\omega t) \end{pmatrix}.$$

Example I.1.3. For the initial value problem

$$\begin{aligned} y'(t) &= y(t)^2 \\ y(0) &= 1 \end{aligned}$$

we have

$$I = \mathbb{R},\ D = \mathbb{R},\ f(t, y) = y^2,\ t_0 = 0,\ y_0 = 1.$$

The exact solution is

$$y(t) = \frac{1}{1-t}.$$

The solution *explodes*, when the time t approaches the value 1 from the left.

Example I.1.4. For the initial value problem

$$\begin{aligned} y'(t) &= \sqrt{|y(t)|} \\ y(0) &= 0 \end{aligned}$$

we have

$$I = \mathbb{R},\ D = \mathbb{R},\ f(t, y) = \sqrt{|y|},\ t_0 = 0,\ y_0 = 0.$$

It admits an *infinite number of solutions*; two of these are given by

$$\begin{aligned} y(t) &= 0 \quad \text{for all } t, \\ y(t) &= \begin{cases} 0 & \text{for } t < 0, \\ \frac{1}{4}t^2 & \text{for } t \geq 0. \end{cases} \end{aligned}$$

Figure I.1.1 shows a typical solution.

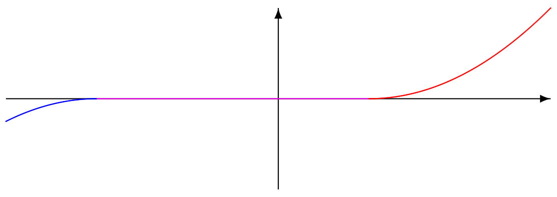

Figure I.1.1. Typical solution for example I.1.4

Example I.1.5. Higher order differential equations can be transformed into first order equations by introducing new unknowns. For the *mechanical system*, e.g.,

$$\begin{aligned} Mx''(t) + Rx'(t) + Kx(t) &= F(t) \\ x(0) &= x_0 \\ x'(0) &= v_0 \end{aligned}$$

in $\mathbb{R}^d$ one introduces the velocity $v(t) = x'(t)$ as a new unknown and thus obtains the equation

$$v'(t) = x''(t) = M^{-1}F(t) - M^{-1}Rv(t) - M^{-1}Kx(t).$$

This transforms the original differential equation of second order in $\mathbb{R}^d$ into a differential equation of first order in $\mathbb{R}^{2d}$ which corresponds to the data

$$\begin{aligned} y(t) &= \begin{pmatrix} x(t) \\ v(t) \end{pmatrix}, \\ f(t,y) &= \begin{pmatrix} 0 \\ M^{-1}F(t) \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ -M^{-1}K & -M^{-1}R \end{pmatrix} y. \end{aligned}$$

I.1.2. Existence and uniqueness of solutions. As demonstrated by examples I.1.3 and I.1.4, not every initial value problem admits a unique solution which is defined on the same interval as the right-hand side f. The following result gives a simple criterion for the existence of a unique solution of an initial value problem and the size of its existence interval. The criterion is not the sharpest possible. The condition concerning the right-hand side f can be relaxed to requiring the existence of a constant L such that

$$|f(t,y_1) - f(t,y_2)| \le L|y_1 - y_2| \quad \textit{(Lipschitz condition)}$$

holds for all $t \in I$ and all $y_1, y_2 \in D$. Continuously differentiable functions with a bounded derivative satisfy this condition. The function $y \mapsto |y|$ is an example of a non-differentiable function satisfying the Lipschitz condition. The functions f of examples I.1.3 and I.1.4 do not satisfy the Lipschitz condition.

If the function f is continuously differentiable w.r.t. the variable y, there is an interval $J = (t_-, t_+)$ with $t_0 \in J$ and a unique continuously differentiable function y on J which solves the initial value problem

$$\begin{aligned} y'(t) &= f(t, y(t)), \\ y(t_0) &= y_0. \end{aligned}$$

Moreover, either J equals I or $y(t)$ tends to the boundary of D when t approaches $t_\pm$.
If the derivative of f w.r.t. the variable y is bounded on $I \times D$, J equals I.

I.1.3. Dependence on the initial value. The dependence of the solution of an initial value problem on the initial value is crucial for solving boundary value problems. The following result gives a simple criterion whether the solution of an initial value problem is a differentiable function of the initial value and simultaneously shows how to compute the corresponding derivative. This will be crucial for the practical solution of boundary value problems with the *simple shooting method* in section I.4.1 and the *multiple shooting method* in section I.4.3.

> If f is twice continuously differentiable w.r.t. the variable y, the solution y of the initial value problem
> $$\begin{aligned} y'(t) &= f(t, y(t)) \\ y(t_0) &= y_0 \end{aligned}$$
> is a differentiable function of the initial value y_0, i.e.
> $$y(t) = y(t; y_0).$$
> The derivative $Z(t)$ of the function $y_0 \mapsto y(t; y_0)$ solves the initial value problem
> $$\begin{aligned} Z'(t) &= D_y f(t, y(t; y_0))Z(t), \\ Z(t_0) &= I. \end{aligned}$$
> Here, $D_y f(t, y)$ is the *Jacobi matrix* of f w.r.t. the variable y and I denotes the *identity matrix.*

EXAMPLE I.1.6. For the damped oscillation of example I.1.2 we have

$$f(t, y) = \begin{pmatrix} \lambda & -\omega \\ \omega & \lambda \end{pmatrix} y.$$

The Jacobi matrix of f is

$$D_y f(t, y) = \begin{pmatrix} \lambda & -\omega \\ \omega & \lambda \end{pmatrix}.$$

The function Z takes its values in the set of 2×2 matrices. The initial value problem for Z is

$$\begin{aligned} Z'(t) &= \begin{pmatrix} \lambda & -\omega \\ \omega & \lambda \end{pmatrix} Z(t), \\ Z(0) &= I \end{aligned}$$

or in components

$$\begin{aligned} z'_{1,1}(t) &= \lambda z_{1,1}(t) - \omega z_{2,1}(t), & z_{1,1}(0) &= 1, \\ z'_{1,2}(t) &= \lambda z_{1,2}(t) - \omega z_{2,2}(t), & z_{1,2}(0) &= 0, \\ z'_{2,1}(t) &= \omega z_{1,1}(t) + \lambda z_{2,1}(t), & z_{2,1}(0) &= 0, \\ z'_{2,2}(t) &= \omega z_{1,2}(t) + \lambda z_{2,2}(t), & z_{2,2}(0) &= 1. \end{aligned}$$

I.2. Numerical Methods for Initial Value Problems

I.2.1. Basic idea. The numerical solution of initial value problems is based on the following idea:

Approximate the solution y of the initial value problem at discrete times $t_0 < t_1 < t_2 < \dots$ and denote by η_i the approximation for $y(t_i)$. For $i = 0, 1, \dots$ successively compute η_{i+1} using f and η_i *(single step methods)* or using f and $\eta_i, \dots, \eta_{i-m}$ *(multi step methods)*.

In what follows we will only consider single step methods.

Many methods, in particular *Runge-Kutta methods*, are obtained by applying a suitable *quadrature formula* to the integral in the identity

$$\eta_{i+1} - \eta_i \approx y(t_{i+1}) - y(t_i) = \int_{t_i}^{t_{i+1}} f(s, y(s))ds.$$

For abbreviation, one denotes by $h_i = t_{i+1} - t_i$ the i-th step-size. In the simplest case the points $t_0, t_1, \dots$ are chosen *equidistant*, i.e. $h_i = h$ and $t_i = t_0 + ih$ for all i.

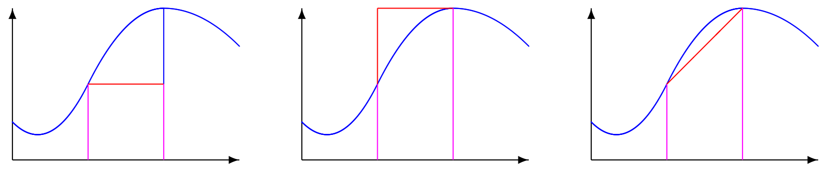

FIGURE I.2.1. Quadrature formulae for the explicit Euler method (left), the implicit Euler method (middle) and the trapezoidal rule (right)

I.2.2. Simplest methods. The simplest numerical methods for solving initial value problems are the *explicit Euler method*, the *implicit Euler method* and the *trapezoidal rule* which is also known as *Crank-Nicolson scheme*. Figure I.2.1 sketches the corresponding quadrature rules.

Explicit Euler method:

$$\begin{aligned} \eta_0 &= y_0, \\ \eta_{i+1} &= \eta_i + h_i f(t_i, \eta_i), \\ t_{i+1} &= t_i + h_i. \end{aligned}$$

Implicit Euler method:

$$\begin{aligned} \eta_0 &= y_0, \\ \eta_{i+1} &= \eta_i + h_i f(t_{i+1}, \eta_{i+1}), \\ t_{i+1} &= t_i + h_i. \end{aligned}$$

Trapezoidal rule alias Crank-Nicolson scheme:

$$\begin{aligned}\eta_0 &= y_0,\\ \eta_{i+1} &= \eta_i + \frac{h_i}{2}\Big[f(t_i,\eta_i) + f(t_{i+1},\eta_{i+1})\Big],\\ t_{i+1} &= t_i + h_i.\end{aligned}$$

I.2.3. Runge-Kutta methods. The three methods of the previous section are particular representatives of a larger class of methods, the so-called *Runge-Kutta methods.* These have the following general form:

Runge-Kutta method:

$$\begin{aligned}\eta_0 &= y_0,\\ \eta_{i,j} &= \eta_i + h_i \sum_{k=1}^{r} a_{jk} f(t_i + c_k h, \eta_{i,k}) \quad \text{for } j = 1,\ldots,r,\\ \eta_{i+1} &= \eta_i + h_i \sum_{k=1}^{r} b_k f(t_i + c_k h, \eta_{i,k}),\\ t_{i+1} &= t_i + h_i.\end{aligned}$$

Here, r, $0 \le c_1 \le \ldots \le c_r \le 1$, $(a_{jk})_{1\le j,k\le r}$ and $b_1, \ldots, b_r$ are given numbers characterizing the particular method. The number r is called *stage number* of the Runge-Kutta method.

The two Euler methods correspond to the choices $r = 1$ and $c_1 = 0$, $a_{11} = 0$, $b_1 = 1$ for the explicit Euler method and $c_1 = 1$, $a_{11} = 1$, $b_1 = 1$ for the implicit Euler method. The trapezoidal rule corresponds to $r = 2$ and $c_1 = 0$, $c_2 = 1$, $a_{11} = a_{12} = 0$, $a_{21} = a_{22} = \frac{1}{2}$, $b_1 = b_2 = \frac{1}{2}$.

The method is called *explicit*, if $a_{jk} = 0$ holds for all $k \ge j$. It is called *implicit*, if $a_{j,k} \ne 0$ holds for at least one $k \ge j$. The most prominent representative of explicit methods is the so-called *classical Runge-Kutta method* with stage number 4:

Classical Runge-Kutta method:

$$\begin{aligned}\eta_0 &= y_0\\ \eta_{i,1} &= \eta_i\\ \eta_{i,2} &= \eta_i + \frac{h_i}{2} f(t_i, \eta_{i,1})\\ \eta_{i,3} &= \eta_i + \frac{h_i}{2} f(t_i + \frac{h_i}{2}, \eta_{i,2})\end{aligned}$$

$$\begin{aligned}
\eta_{i,4} &= \eta_i + h_i f(t_i + \frac{h_i}{2}, \eta_{i,3}) \\
\eta_{i+1} &= \eta_i + \frac{h_i}{6}\Big\{ f(t_i, \eta_{i,1}) + 2f(t_i + \frac{h_i}{2}, \eta_{i,2}) \\
&\qquad + 2f(t_i + \frac{h_i}{2}, \eta_{i,3}) \\
&\qquad + f(t_i + h_i, \eta_{i,4})\Big\} \\
t_{i+1} &= t_i + h_i
\end{aligned}$$

When using an explicit method, the quantities $\eta_{i,1}, \ldots, \eta_{i,r}$ can be computed successively. When using an implicit method, the computation of $\eta_{i,1}, \ldots, \eta_{i,r}$ requires the solution of a (non-linear) system of equations with $r \cdot d$ equations and unknowns.

Thanks to their high order and good stability, *strongly diagonal implicit Runge-Kutta methods* or *SDIRK methods* in short are of a particular practical importance. They have a lower diagonal matrix $(a_{ij})_{1 \le i,j \le r}$ with identical diagonal entries. The computation of $\eta_{i,1}$, ..., $\eta_{i,r}$ therefore requires the successive solution of r (non-linear) systems of equations with d equations and unknowns each. Moreover, all systems have the same Jacobi matrix which reduces the computational cost of Newton's method. Two examples of SDIRK methods with stage numbers 2 and 5, resp. are:

SDIRK2:

$$\begin{aligned}
\eta_0 &= y_0 \\
\eta_{i,1} &= \eta_i + \frac{3+\sqrt{3}}{6} h_i f(t_i + \frac{3+\sqrt{3}}{6} h_i, \eta_{i,1}) \\
\eta_{i,2} &= \eta_i - \frac{\sqrt{3}}{3} h_i f(t_i + \frac{3+\sqrt{3}}{6} h_i, \eta_{i,1}) \\
&\qquad + \frac{3+\sqrt{3}}{6} h_i f(t_i + \frac{3-\sqrt{3}}{6} h_i, \eta_{i,2}) \\
\eta_{i+1} &= \eta_i + \frac{h_i}{2}\Big\{ f(t_i + \frac{3+\sqrt{3}}{6} h_i, \eta_{i,1}) \\
&\qquad + f(t_i + \frac{3-\sqrt{3}}{6} h_i, \eta_{i,2})\Big\} \\
t_{i+1} &= t_i + h_i
\end{aligned}$$

SDIRK5:

$$\begin{aligned}
\eta_0 &= y_0 \\
\eta_{i,1} &= \eta_i + \frac{h_i}{4} f(t_i + \frac{1}{4}h_i, \eta_{i,1}) \\
\eta_{i,2} &= \eta_i + \frac{h_i}{4}\Big\{2f(t_i + \frac{1}{4}h_i, \eta_{i,1}) + f(t_i + \frac{3}{4}h_i, \eta_{i,2})\Big\} \\
\eta_{i,3} &= \eta_i + \frac{h_i}{100}\Big\{34f(t_i + \frac{1}{4}h_i, \eta_{i,1}) - 4f(t_i + \frac{3}{4}h_i, \eta_{i,2}) \\
&\qquad + 25f(t_i + \frac{11}{20}h_i, \eta_{i,3})\Big\} \\
\eta_{i,4} &= \eta_i + \frac{h_i}{2720}\Big\{742f(t_i + \frac{1}{4}h_i, \eta_{i,1}) - 137f(t_i + \frac{3}{4}h_i, \eta_{i,2}) \\
&\qquad + 75f(t_i + \frac{11}{20}h_i, \eta_{i,3}) + 680f(t_i + \frac{1}{2}h_i, \eta_{i,4})\Big\} \\
\eta_{i,5} &= \eta_i + \frac{h_i}{48}\Big\{50f(t_i + \frac{1}{4}h_i, \eta_{i,1}) - 49f(t_i + \frac{3}{4}h_i, \eta_{i,2}) \\
&\qquad + 375f(t_i + \frac{11}{20}h_i, \eta_{i,3}) - 340f(t_i + \frac{1}{2}h_i, \eta_{i,4}) \\
&\qquad + 12f(t_i + h_i, \eta_{i,5})\Big\} \\
\eta_{i+1} &= \eta_i + \frac{h_i}{48}\Big\{50f(t_i + \frac{1}{4}h_i, \eta_{i,1}) - 49f(t_i + \frac{3}{4}h_i, \eta_{i,2}) \\
&\qquad + 375f(t_i + \frac{11}{20}h_i, \eta_{i,3}) - 340f(t_i + \frac{1}{2}h_i, \eta_{i,4}) \\
&\qquad + 12f(t_i + h_i, \eta_{i,5})\Big\} \\
t_{i+1} &= t_i + h_i
\end{aligned}$$

I.2.4. Order of a single step method. The *order* is a measure for the quality of a single step method. It measures the error of *one* step of the method and is defined as follows:

A single step method has order $p > 0$ if

$$|y(t_1) - \eta_1| = O(h_1^{p+1}).$$

The error after an *arbitrary* number of steps of the single step method satisfies:

If the single step method has order p and if the right-hand side f is continuously differentiable w.r.t. the variable y

with bounded derivative, the error satisfies for all i

$$|y(t_i) - \eta_i| = O\Big(\big(\max_{1\le j\le i} h_j\big)^p\Big)$$

Both Euler methods have order 1. The Crank-Nicolson scheme is of order 2. The classical Runge-Kutta method has order 4. The methods SDIRK2 and SDIRK5 given above are of order 3 and 4, respectively. There are Runge-Kutta methods of arbitrarily high order.

I.2.5. Stability of a single step method. The order of a single step method describes its *asymptotic* behaviour for step-sizes tending to zero. In practice, however, one of course uses a given finite step-size. Therefore, the single step method should yield a *qualitatively* correct solution for an as large as possible range of step-sizes. This requirement is described by the concept of *stability*.

Example I.2.1. When numerically solving the initial value problem

$$\begin{aligned} y'(t) &= -100y(t) \\ y(0) &= 1 \end{aligned}$$

with exact solution $y(t) = e^{-100t}$ by the two Euler methods and the Crank-Nicolson scheme with constant step-size h we obtain

$$\eta_i = \begin{cases} \big(1-100h\big)^i & \text{for the explicit Euler method,} \\ \big(1+100h\big)^{-i} & \text{for the implicit Euler method,} \\ \big(\frac{1-50h}{1+50h}\big)^i & \text{for the Crank-Nicolson scheme.} \end{cases}$$

Hence, we observe:

- The explicit Euler method only yields a decaying solution if the step-size is less than $\frac{1}{50}$.
- The implicit Euler method and the Crank-Nicolson scheme yield decaying solutions for *all* step-sizes.

Explicit methods cannot by stable. But there are stable implicit Runge-Kutta methods of arbitrarily high order.

The following examples demonstrate the effect of good stability properties.

Example I.2.2. Consider the damped oscillation

$$\begin{aligned} y'(t) &= \begin{pmatrix} -0.9 & -6.3 \\ 6.3 & -0.9 \end{pmatrix} y(t) \\ y(0) &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{aligned}$$

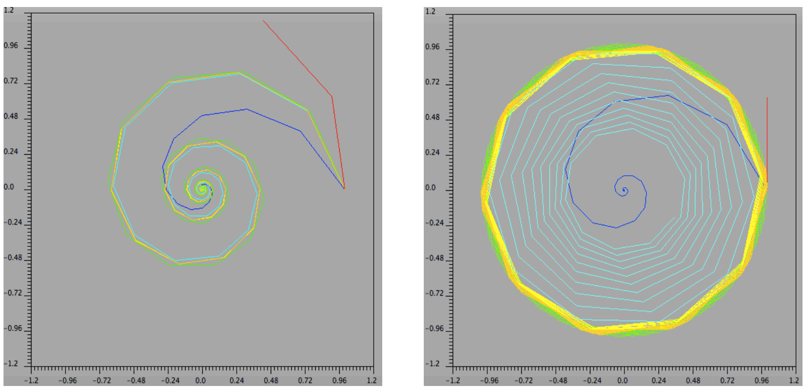

FIGURE I.2.2. 100 steps with step-size $h = 0.1$ of the explicit Euler-method (red), implicit Euler method (blue), the trapezoidal rule (green), the classical Runge-Kutta method (yellow), the SDIRK2 method (turquoise) and the SDIRK5 method (orange) for the initial value problem of a damped oscillation of example I.2.2 (left) and an undamped oscillation of example I.2.3 (right). The explicit Euler method leaves the frame within few steps.

with exact solution

$$y(t) = e^{-0.9t} \begin{pmatrix} \cos(6.3t) \\ \sin(6.3t) \end{pmatrix}.$$

The function $t \mapsto y(t)$ describes a contracting spiral with centre at the origin. The left part of figure I.2.2 shows the result of the two Euler methods (red and blue), the trapezoidal rule (green), the classical Runge-Kutta method (yellow), the SDIRK2 (turquoise) and the SDIRK5 method (orange) for 100 steps with constant step-size $h = 0.1$. The explicit Euler method (red) explodes within few steps and leaves the frame of the figure since the step-size is too large. In order to obtain a qualitatively correct solution with this method one would need a step-size which is smaller by about a factor 10. The implicit Euler method (blue) damps the solution too much. This effect diminishes when reducing the step-size but persists in principle. The other methods yield qualitatively acceptable solutions.

EXAMPLE I.2.3. Consider the undamped oscillation

$$y'(t) = \begin{pmatrix} 0 & -6.3 \\ 6.3 & 0 \end{pmatrix} y(t)$$

$$y(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

with exact solution

$$y(t) = \begin{pmatrix} \cos(6.3t) \\ \sin(6.3t) \end{pmatrix}.$$

The function $t \mapsto y(t)$ describes a circle with radius 1 and centre at the origin. The right part of figure I.2.2 shows the result of the two Euler methods (red and blue), the trapezoidal rule (green), the classical Runge-Kutta method (yellow), the SDIRK2 (turquoise) and the SDIRK5 method (orange) for 100 steps with constant step-size $h = 0.1$. The explicit Euler method (red) explodes within few steps and leaves the frame of the figure since the step-size is too large. This effect persists with decreasing step-size although more and more weakly. The implicit Euler method (blue) damps the solution too much. This effect diminishes when reducing the step-size but persists in principle. The other methods yield qualitatively acceptable solutions.

I.3. Boundary Value Methods

I.3.1. General form of boundary value problems. Given an interval I in $\mathbb{R}$, two distinct points a and b in I, a subset D of $\mathbb{R}^d$, a function $f(t, y) : I \times D \to \mathbb{R}^d$ and a function $r(u, v) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$, a *boundary value problem* consists in finding a differentiable function $y : I \to D$ such that

$$\boxed{\begin{aligned} y'(t) &= f(t, y(t)) \quad \text{for all } t \in I \quad \textit{(differential equation)} \\ r(y(a), y(b)) &= 0 \qquad\qquad\qquad\qquad \textit{(boundary condition)} \end{aligned}}$$

Example I.3.1. The boundary value problem

$$\begin{aligned} y'(t) &= \begin{pmatrix} \lambda & -\omega \\ \omega & \lambda \end{pmatrix} y(t) \\ y_1(0) &= 1 \\ y_1(\frac{\pi}{2\omega}) &= 0 \end{aligned}$$

for a *damped oscillation* corresponds to the data

$$\begin{aligned} I = \mathbb{R}, \quad a = 0, \quad b &= \frac{\pi}{2\omega}, \quad D = \mathbb{R}^2, \\ f(t, y) &= \begin{pmatrix} \lambda & -\omega \\ \omega & \lambda \end{pmatrix} y, \\ r(u, v) &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} u + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} v - \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \end{aligned}$$

The exact solution is

$$y(t) = e^{\lambda t} \begin{pmatrix} \cos(\omega t) \\ \sin(\omega t) \end{pmatrix}.$$

Example I.3.2. In analogy to initial value problems, boundary value problems for higher order differential equations can be transformed into boundary value problems for first order equations by introducing new unknowns. For the *mechanical system*, e.g.,

$$\begin{aligned} Mx''(t) + Rx'(t) + Kx(t) &= F(t) \\ x(0) &= x_0 \\ x(L) &= x_L \end{aligned}$$

in $\mathbb{R}^d$ we introduce the velocity $v(t) = x'(t)$ as a new unknown and thus obtain a boundary value problem corresponding to the data

$$\begin{aligned} I = \mathbb{R}, \quad a = 0, \quad b &= L, \quad D = \mathbb{R}^{2d}, \\ y(t) &= \begin{pmatrix} x(t) \\ v(t) \end{pmatrix}, \\ f(t,y) &= \begin{pmatrix} 0 \\ M^{-1}F(t) \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ -M^{-1}K & -M^{-1}R \end{pmatrix} y, \\ r(u,v) &= \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} u + \begin{pmatrix} 0 & 0 \\ I & 0 \end{pmatrix} v - \begin{pmatrix} x_0 \\ x_L \end{pmatrix}. \end{aligned}$$

I.3.2. Eigenvalue and free boundary problems. Some problems which at first sight are no boundary value problems nevertheless fit into the framework of the preceding section. The most prominent examples are eigenvalue and free boundary problems.

Example I.3.3. We are looking for a function $u : [a,b] \to \mathbb{R}$ and a number $\lambda \in \mathbb{R}$ such that

$$\begin{aligned} u'(t) &= g(t, u(t)), \\ \rho(u(a), u(b), \lambda) &= 0, \end{aligned}$$

where g and ρ are given functions. Interpreting the number λ as a constant function, this *eigenvalue problem* corresponds to a boundary value problem with the data

$$\begin{aligned} I = \mathbb{R}, \quad D &= \mathbb{R}^2, \\ y(t) &= \begin{pmatrix} u(t) \\ \lambda \end{pmatrix}, \\ f(t,y) &= \begin{pmatrix} g(t, y_1) \\ 0 \end{pmatrix}, \\ r(u,v) &= \rho(u_1, v_1, v_2). \end{aligned}$$

Example I.3.4. We are looking for a number $\beta > 0$ and a function $u : [0, \beta] \to \mathbb{R}$ such that

$$\begin{aligned} u'(s) &= g(s, u(s)), \\ \rho(u(0), u(\beta)) &= 0, \end{aligned}$$

where g and ρ again are given functions. This is a *free boundary problem*, since the point β is part of the unknown solution. Interpreting the number β as a constant function and judiciously introducing a new variable t, this problem corresponds to a boundary value problem with the data

$$\begin{aligned} I = \mathbb{R}, \quad a &= 0, \quad b = 1, \quad D = \mathbb{R}^2, \\ y(t) &= \begin{pmatrix} u(t\beta) \\ \beta \end{pmatrix}, \\ t &= \frac{s}{y_2}, \\ f(t,y) &= \begin{pmatrix} y_2 g(ty_2, y_1) \\ 0 \end{pmatrix}, \\ r(u,v) &= \rho(u_1, v_1). \end{aligned}$$

I.3.3. Existence and uniqueness. Contrary to initial value problems there is no general existence and uniqueness result for boundary value problems. Instead the solvability and the eventual number of solutions of a given boundary value problem depend on the particular problem and the interplay of the differential equation and the boundary condition. This is illustrated by the following example.

Example I.3.5. Consider the boundary value problem

$$y'(t) = \begin{pmatrix} 0 & -\omega \\ \omega & 0 \end{pmatrix} y(t)$$
$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} y(0) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} y(L) = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

for an *undamped oscillation*. The general solution of the differential equation is

$$y(t) = \begin{pmatrix} c_1 \cos(\omega t) - c_2 \sin(\omega t) \\ c_1 \sin(\omega t) + c_2 \cos(\omega t) \end{pmatrix}.$$

The data $L = \frac{2\pi}{\omega}$, $\alpha = 0$, $\beta = 1$ lead to the contradictory conditions $c_1 = 0$ and $c_1 = 1$. Hence, the corresponding boundary value problem doesn't admit a solution. The data $L = \frac{2\pi}{\omega}$, $\alpha = 0$, $\beta = 0$, on the other hand, lead to the single condition $c_1 = 0$. Hence, c_2 is arbitrary and the corresponding boundary value problem admits an infinite number of solutions.

I.4. Shooting Methods

I.4.1. The simple shooting method. The simple shooting method is the simplest method for solving a general boundary value problem. The underlying idea can be explained as follows:

- Denote by $y(t;s)$ the solution of the *initial value problem*
$$\begin{aligned} y'(t) &= f(t, y(t)), \\ y(a;s) &= s. \end{aligned}$$
- Then $y(t;s)$ solves the boundary *boundary value problem*
$$\begin{aligned} y'(t) &= f(t, y(t)), \\ r(y(a), y(b)) &= 0 \end{aligned}$$
if and only if
$$r(s, y(b;s)) = 0.$$
- Determine a zero of the function
$$F(s) = r(s, y(b;s))$$
using *Newton's method*.
- The derivative $DF(s)$ of F at the point s is
$$DF(s) = D_u r(s, y(b;s)) + D_v r(s, y(b;s)) Z(b;s),$$
where Z solves the *initial value problem*
$$\begin{aligned} Z'(t;s) &= D_y f(t, y(t;s)) Z(t;s) \\ Z(a;s) &= I \end{aligned}$$
with I denoting the identity matrix.
- Solve the initial value problems for $y(t;s)$ and $Z(t;s)$ approximately with a numerical method for initial value problems as described in section I.3. In doing so ensure that both methods use the same set of grid points t_i.

This idea gives rise to the following algorithm:

Algorithm I.4.1. (Simple shooting method)

(0) *Given an initial guess* $s^{(0)} \in \mathbb{R}^d$ *and a tolerance* ε. *Set* $i = 0$.

(1) *Using a numerical method for initial value problems compute an approximation* $\eta^{(i)}(t)$ *for the solution* $y^{(i)}$ *of the initial value problem*
$$\begin{aligned} y^{(i)\prime}(t) &= f(t, y^{(i)}(t)), \\ y^{(i)}(a) &= s^{(i)}. \end{aligned}$$
Set
$$F^{(i)} = r(s^{(i)}, \eta^{(i)}(b)).$$

(2) *If* $\|F^{(i)}\| \le \varepsilon$ stop, *otherwise continue with step* (3).

(3) *Using the same method and grid points as in step* (1) *compute an approximation* $\zeta^{(i)}(t)$ *for the solution* $Z^{(i)}$ *of the initial value problem*
$$\begin{aligned} Z^{(i)\prime}(t) &= D_y f(t, \eta^{(i)}(t)) Z^{(i)}(t), \\ Z^{(i)}(a) &= I. \end{aligned}$$

Set

$$D^{(i)} = D_u r(s^{(i)}, \eta^{(i)}(b)) + D_v r(s^{(i)}, \eta^{(i)}(b))\zeta^{(i)}(b).$$

(4) *Solve the linear system of equations*

$$D^{(i)}\Delta s^{(i)} = -F^{(i)}.$$

Set

$$s^{(i+1)} = s^{(i)} + \Delta s^{(i)},$$

increase i by 1 *and return to step* (1).

The simple shooting method has the following properties:

The initial value problems in step (1) have d unknowns.
The initial value problems in step (3) have d^2 unknowns.
The initial value problems in step (3) are linear.
The linear systems in step (4) have d equations and unknowns.
Newton's method should be damped.
If Newton's method converges, the convergence is quadratic.

I.4.2. A warning example. Even if the boundary value problem admits a unique solution, the simple shooting method can completely break down. This is illustrated by the following example. The breakdown of the simple shooting method is due to the fact that solutions of initial value problems with close-by initial values may run away with an exponential rate. In this sense boundary value problems may be ill-posed.

Example I.4.2. Consider the boundary value problem

$$\begin{aligned} y'(t) &= \begin{pmatrix} 0 & 1 \\ 110 & 1 \end{pmatrix} y(t), \\ y_1(0) &= 1, \\ y_1(10) &= 1. \end{aligned}$$

The exact solution is

$$y(t) = c_1 e^{-10t} \begin{pmatrix} 1 \\ -10 \end{pmatrix} + c_2 e^{11t} \begin{pmatrix} 1 \\ 11 \end{pmatrix}$$

with

$$c_1 = \frac{e^{110} - 1}{e^{110} - e^{-100}}, \quad c_2 = \frac{1 - e^{-100}}{e^{110} - e^{-100}}.$$

The solution of the associated initial value problem with initial value s is

$$y(t; s) = \frac{11s_1 - s_2}{21} e^{-10t} \begin{pmatrix} 1 \\ -10 \end{pmatrix} + \frac{10s_1 + s_2}{21} e^{11t} \begin{pmatrix} 1 \\ 11 \end{pmatrix}.$$

The correct initial value for the solution of the boundary value problem is

$$s^* = \begin{pmatrix} 1 \\ -10 + 21 \cdot \frac{1-e^{-100}}{e^{110}-e^{-100}} \end{pmatrix}.$$

The wrong initial value

$$\widetilde{s} = \begin{pmatrix} 1 \\ -10 + 10^{-9} \end{pmatrix}$$

with a relative error of 10^{-10} yields the wrong boundary value

$$y_1(10; \widetilde{s}) \approx 10^{37}.$$

Thus one looses 47 digits!

I.4.3. The multiple shooting method. Example I.4.2 shows that the simple shooting method may completely break down since solutions to different initial values may separate with an exponential rate. It, however, also indicates a way for avoiding this undesirable phenomenon: Only solve initial value problems on small time intervals. This leads to the following divide-and-conquer-type approach:

- Subdivide the interval $[a, b]$ by introducing intermediate points $a = \tau_1 < \tau_2 < \ldots < \tau_m = b$.
- For $s_1, \ldots, s_m \in \mathbb{R}^d$ denote by $y(t; \tau_k, s_k)$ the solution of the initial value problem
$$\begin{aligned} y'(t) &= f(t, y(t)), \\ y(\tau_k; s_k) &= s_k. \end{aligned}$$
- Define a piecewise function $\widetilde{y}$ by
$$\begin{aligned} \widetilde{y}(t) &= y(t; \tau_k, s_k) \text{ for } \tau_k \le t < \tau_{k+1},\ 1 \le k \le m-1, \\ \widetilde{y}(\tau_m) &= s_m. \end{aligned}$$
- Then $\widetilde{y}$ solves the boundary value problem
$$\begin{aligned} y'(t) &= f(t, y(t)) \\ r(y(a), y(b)) &= 0 \end{aligned}$$
if and only if
$$\begin{aligned} y(\tau_{k+1}; \tau_k, s_k) &= s_{k+1} \quad \text{for } 1 \le k \le m-1, \\ r(s_1, s_m) &= 0. \end{aligned}$$
- This yields a system of equations
$$F(s_1, \ldots, s_m) = 0,$$
which can be solved with Newton's method.
- The computation of the derivative of F requires the solution of initial value problems on the intervals $[\tau_k, \tau_{k+1}]$.

Each step of Newton's method requires the solution of a linear system of equations

$$DF\Delta s = -F$$

with $m \cdot d$ equations and unknowns. Since DF has the particular form

$$DF = \begin{pmatrix} G_1 & -I & & & \\ & G_2 & -I & & 0 \\ 0 & & \ddots & \ddots & \\ & & & G_{m-1} & -I \\ A & 0 & \cdots & 0 & B \end{pmatrix}$$

with suitable $d \times d$ matrices G_1, ..., G_{m-1}, A and B, the linear system takes the form

$$\begin{aligned} G_1\Delta s_1 - \Delta s_2 &= -F_1 \\ G_2\Delta s_2 - \Delta s_3 &= -F_2 \\ \vdots &= \vdots \\ G_{m-1}\Delta s_{m-1} - \Delta s_m &= -F_{m-1} \\ A\Delta s_1 + B\Delta s_m &= -F_m. \end{aligned}$$

Hence, the unknowns Δs_2, ..., Δs_m can be eliminated successively and one obtains the linear system

$$(A + BG_{m-1} \dots G_1)\Delta s_1 = -F_m - B\sum_{j=1}^{m-1}(\prod_{i=j+1}^{m-1} G_i)F_j$$

with d equations for the d components of Δs_1.

These ideas and observations give rise to the following algorithm:

Algorithm I.4.3. (Multiple shooting method)

(0) *Given m points $a = \tau_1 < \ldots < \tau_m = b$, m vectors $s_1^{(0)}, \ldots, s_m^{(0)} \in \mathbb{R}^d$ and a tolerance ε. Set $i = 0$.*

(1) *Using a numerical method for initial value problems determine approximations $\eta^{(i,j)}(t)$, $1 \le j \le m-1$, for the solutions $y^{(i,j)}$ of the initial value problems*

$$\begin{aligned} y^{(i,j)'}(t) &= f(t, y^{(i,j)}(t)), \\ y^{(i,j)}(\tau_j) &= s_j^{(i)} \end{aligned}$$

for $1 \le j \le m-1$. Set

$$\begin{aligned} F_j^{(i)} &= \eta^{(i,j)}(\tau_{j+1}) - s_{j+1}^{(i)} \quad \text{for } 1 \le j \le m-1, \\ F_m^{(i)} &= r(s_1^{(i)}, s_m^{(i)}). \end{aligned}$$

(2) *If*

$$\sum_{j=1}^{m} \|F_j^{(i)}\| \le \varepsilon$$

stop. *Otherwise continue with step* (3).

(3) *Using the same numerical method and grid points as in step* (1) *compute approximations* $\zeta^{(i,j)}(t)$ *for the solutions* $Z^{(i,j)}$ *of the initial value problems*

$$\begin{aligned} Z^{(i,j)\prime}(t) &= D_y f(t, \eta^{(i,j)}(t)) Z^{(i,j)}(t) \\ Z^{(i,j)}(\tau_j) &= I \end{aligned}$$

for $1 \le j \le m-1$. *Set*

$$G_j^{(i)} = \zeta^{(i,j)}(\tau_{j+1})$$

for $1 \le j \le m-1$ *and*

$$\begin{aligned} A^{(i)} &= D_u r(s_1^{(i)}, s_m^{(i)}), \\ B^{(i)} &= D_v r(s_1^{(i)}, s_m^{(i)}). \end{aligned}$$

(4) *Compute the matrix*

$$H^{(i)} = A^{(i)} + B^{(i)} G_{m-1}^{(i)} \cdot \ldots \cdot G_1^{(i)}$$

and the vector

$$\varphi^{(i)} = -F_m^{(i)} - B^{(i)} \sum_{j=1}^{m-1} \Big(\prod_{l=j+1}^{m-1} G_l^{(i)} \Big) F_j^{(i)}.$$

Solve the linear system of equations

$$H^{(i)} \Delta s_1^{(i)} = \varphi^{(i)}$$

and recursively compute the vectors

$$\Delta s_{k+1}^{(i)} = G_k^{(i)} \Delta s_k^{(i)} + F_k^{(i)}$$

for $1 \le k \le m-1$. *Set*

$$s_k^{(i+1)} = s_k^{(i)} + \Delta s_k^{(i)}$$

for $1 \le k \le m$, *increase* i *by* 1 *and return to step* (1).

The multiple shooting method has the following properties:

When using the same number of grid points on the complete interval $[a, b]$, the solution of the initial value problems in the simple and the multiple shooting method requires the same number of arithmetic operations.
The initial value problems on the subintervals can be solved in parallel.
When lacking any additional information, the intermediate points $\tau_1, \ldots, \tau_m$ can be chosen equidistant.

I.5. Finite Difference Methods

I.5.1. Sturm-Liouville problems. In this and the following section we consider particular boundary value problems, the so-called *Sturm-Liouville problems.* Here, we are given a continuously differentiable function $p : [0,1] \to \mathbb{R}$ with

$$\underline{p} = \min_{0 \le x \le 1} p(x) > 0$$

and a continuous function $q : [0,1] \to \mathbb{R}$ with

$$\underline{q} = \min_{0 \le x \le 1} q(x) > 0$$

and we are looking for a twice continuously differentiable function $u : [0,1] \to \mathbb{R}$ with

$$\boxed{\begin{aligned} -(pu')' + qu = f \quad \text{in } (0,1) \quad & \textit{(differential equation)} \\ u(0) = 0,\ u(1) = 0 \quad & \textit{(boundary condition)} \end{aligned}}$$

Example I.5.1. Often Sturm-Liouville problems are given in the general form

$$\begin{aligned} -(pu')' + qu &= f \quad \text{in } (a,b) \\ u(a) = \alpha,\ u(b) &= \beta. \end{aligned}$$

This can be transformed into the above particular form with $a = 0$, $b = 1$, $\alpha = 0$, $\beta = 0$ as follows: Seek u in the form

$$u(x) = \alpha + \frac{\beta - \alpha}{b - a}(x - a) + v(\frac{x - a}{b - a})$$

with

$$v(0) = 0,\ v(1) = 0$$

and introduce a new variable

$$t = \frac{x - a}{b - a}.$$

I.5.2. Difference quotients. The difference methods of this section are based on the *symmetric difference quotient*:

$$\boxed{\partial_h \varphi(x) = \frac{1}{h}\big[\varphi(x + \frac{h}{2}) - \varphi(x - \frac{h}{2})\big].}$$

Taylor's formula yields

$$\boxed{\partial_h \varphi(x) = \varphi'(x) + \frac{h^2}{24}\varphi'''(x + \theta h)}$$

with a suitable $\theta \in (-\frac{1}{2}, \frac{1}{2})$.

I.5.3. Difference discretization. The idea of the difference discretization can be described as follows:

- Replace derivatives by the difference quotient ∂_h

$$
\begin{aligned}
&\big(-(pu')'\big)(x)\\
&\approx \big(-\partial_h(pu')\big)(x)\\
&= \frac{1}{h}\big[p(x-\frac{h}{2})u'(x-\frac{h}{2}) - p(x+\frac{h}{2})u'(x+\frac{h}{2})\big]\\
&\approx \frac{1}{h}\big[p(x-\frac{h}{2})\partial_h u(x-\frac{h}{2}) - p(x+\frac{h}{2})\partial_h u(x+\frac{h}{2})\big]\\
&= \frac{1}{h^2}\big[p(x-\frac{h}{2})(u(x)-u(x-h))\\
&\qquad - p(x+\frac{h}{2})(u(x+h)-u(x))\big]
\end{aligned}
$$

- Impose the resulting equations only in a set of *grid points* ih with $h=\frac{1}{n+1}$ und $1\le i\le n$.

ALGORITHM I.5.2. (Difference discretization)

(0) *Choose a mesh-size* $h=\frac{1}{n+1}$.

(1) *For* $1\le i\le n$ *set*

$$f_i = f(ih),\ q_i = q(ih),\ p_{i\pm\frac12} = p(ih \pm \frac{h}{2}).$$

(2) *Determine* $u_0, \ldots, u_{n+1}$ *such that*

$$u_0 = 0,\ u_{n+1} = 0$$

and

$$
\begin{aligned}
f_i = &-\frac{1}{h^2}p_{i-\frac12}u_{i-1} + \Big(\frac{1}{h^2}\big[p_{i-\frac12}+p_{i+\frac12}\big] + q_i\Big)u_i\\
&-\frac{1}{h^2}p_{i+\frac12}u_{i+1}
\end{aligned}
$$

holds for $1\le i\le n$.

(3) *Denote by* u_h *the continuous piecewise linear function which equals* u_i *at grid point* ih*(see figure I.5.1).*

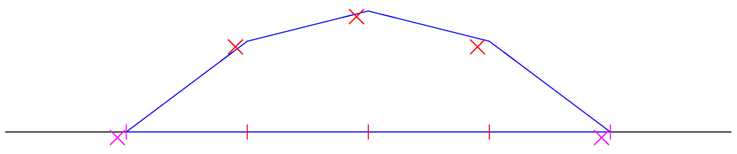

FIGURE I.5.1. Continuous piecewise linear interpolation

The difference discretization has the following properties:

The difference discretization gives rise to a linear system of equations with n equations for the n unknowns $u_1, \ldots, u_n$.
The matrix of the linear system is symmetric, positive definite and tridiagonal with positive diagonal elements and negative off-diagonal elements.
The linear system admits a unique solution.
The solution of the linear system with Gaussian elimination or Cholesky decomposition requires $O(n)$ operations.

I.5.4. Error estimates. The following *a priori error estimates* can be proven for the solution of the difference discretization:

Suppose that $\underline{q} > 0$, $\underline{p} > 0$, p is three times continuously differentiable and the solution u of the Sturm-Liouville problem is four times continuously differentiable, then the following *a priori error estimate* holds

$$\max_{0 \le x \le 1} |u(x) - u_h(x)| \le ch^2.$$

The constant c depends on the lower bound $\underline{q}$ for q, the derivatives up to order 3 of p and the derivatives up to order 4 of u.

I.6. Variational Methods

The assumptions

- $\underline{q} > 0$,
- p three times continuously differentiable,
- u four times continuously differentiable

of the previous section are far too restrictive for most practical applications. They are overcome by the variational methods of the current section.

I.6.1. Idea of the variational formulation. The basic idea of the variational formulation of Sturm-Liouville problems can be described as follows:

- Multiply the differential equation with a continuously differentiable function v with $v(0) = 0$ and $v(1) = 0$:
$$-(pu')'(x)v(x) + q(x)u(x)v(x) = f(x)v(x)$$
for $0 \le x \le 1$.
- Integrate the result from 0 to 1:
$$\int_0^1 \big[-(pu')'(x)v(x) + q(x)u(x)v(x)\big]dx = \int_0^1 f(x)v(x)dx.$$

- Use integration by parts for the term containing derivatives:

$$\begin{aligned}
&-\int_0^1 (pu')'(x)v(x)dx \\
&= p(0)u'(0)\underbrace{v(0)}_{=0} - p(1)u'(1)\underbrace{v(1)}_{=0} + \int_0^1 p(x)u'(x)v'(x)dx \\
&= \int_0^1 p(x)u'(x)v'(x)dx.
\end{aligned}$$

To put these ideas on a profound basis we must better specify the properties of the functions u and v. Classical properties such as continuous differentiability are too restrictive; the notion 'derivative' must be generalised in a suitable way. In view of the discretization the new notion should in particular cover piecewise differentiable functions.

I.6.2. Weak derivatives. The above considerations lead to the notion of a *weak derivative*. It is motivated by the following observation: Integration by parts yields for all continuously differentiable functions u and v satisfying $v(0) = 0$ and $v(1) = 0$

$$\begin{aligned}
\int_0^1 u'(x)v(x)dx &= u(1)\underbrace{v(1)}_{=0} - u(0)\underbrace{v(0)}_{=0} - \int_0^1 u(x)v'(x)dx \\
&= -\int_0^1 u(x)v'(x)dx.
\end{aligned}$$

A function u is called *weakly differentiable* with *weak derivative* w, if every continuously differentiable function v with $v(0) = 0$ and $v(1) = 0$ satisfies

$$\int_0^1 w(x)v(x)dx = -\int_0^1 u(x)v'(x)dx.$$

Example I.6.1. Every continuously differentiable function is weakly differentiable and the weak derivative equals the classical derivative. Every continuous, piecewise continuously differentiable function is weakly differentiable and the weak derivative equals the classical piecewise derivative.
The function $u(x) = 1 - |2x - 1|$ is weakly differentiable with weak derivative

$$w(x) = \begin{cases} 2 & \text{for } 0 < x < \frac{1}{2} \\ -2 & \text{for } \frac{1}{2} < x < 1 \end{cases}$$

(cf. figure I.6.1). Notice that the value $w(\frac{1}{2})$ is arbitrary.

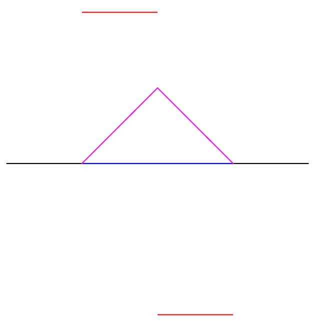

FIGURE I.6.1. Function $u(x) = 1 - |2x - 1|$ (magenta) with its weak derivative (red)

I.6.3. Sobolev spaces and norms. Variational formulations and finite element methods are based on *Sobolev spaces*.

The L^2-*norm* is defined by

$$\|v\| = \left\{ \int_0^1 |v(x)|^2 dx \right\}^{\frac{1}{2}}.$$

$L^2(0,1)$ denotes the *Lebesgue space* of all functions v with finite L^2-norm $\|v\|$.
$H^1(0,1)$ is the *Sobolev space* of all functions v in $L^2(0,1)$, whose weak derivative exists and is contained in $L^2(0,1)$ too.
$H_0^1(0,1)$ denotes the *Sobolev space* of all functions v in $H^1(0,1)$ which satisfy $v(0) = 0$ and $v(1) = 0$.

EXAMPLE I.6.2. Every bounded function is contained in $L^2(0,1)$. The function $v(x) = \frac{1}{\sqrt{x}}$ is not contained in $L^2(0,1)$, since the integral of $\frac{1}{x} = v(x)^2$ is not finite.
Every continuously differentiable function is contained in $H^1(0,1)$.
Every continuous, piecewise continuously differentiable function is contained in $H^1(0,1)$.
The function $v(x) = 1 - |2x - 1|$ is contained in $H_0^1(0,1)$ (cf. figure I.6.1).
The function $v(x) = 2\sqrt{x}$ is not contained in $H^1(0,1)$, since the integral of $\frac{1}{x} = \big(v'(x)\big)^2$ is not finite.

Notice that, in contrast to several dimensions, all functions in $H^1(0,1)$ are continuous.

I.6.4. Variational formulation of the Sturm-Liouville problem. The variational formulation of the Sturm-Liouville problem is given by:

Find $u \in H_0^1(0,1)$ such that for all $v \in H_0^1(0,1)$ there holds
$$\int_0^1 \big[p(x)u'(x)v'(x) + q(x)u(x)v(x)\big]dx = \int_0^1 f(x)v(x)dx.$$

It has the following properties:

It admits a unique solution.
Its solution is the unique *minimum* in $H_0^1(0,1)$ of the *energy function*
$$\frac{1}{2}\int_0^1 \big[p(x)u'(x)^2 + q(x)u(x)^2\big]dx - \int_0^1 f(x)u(x)dx.$$

I.6.5. Finite element spaces. The discretization of the above variational problem is based on finite element spaces. For their definition denote by $\mathcal{T}$ an arbitrary *partition* of the interval $(0,1)$ into non-overlapping sub-intervals and by $k \geq 1$ an arbitrary polynomial degree.

$S^{k,0}(\mathcal{T})$ denotes the *finite element space* of all continuous functions which are piecewise polynomials of degree k on the intervals of $\mathcal{T}$.
$S_0^{k,0}(\mathcal{T})$ is the *finite element space* of all functions v in $S^{k,0}(\mathcal{T})$ which satisfy $v(0) = 0$ and $v(1) = 0$.

I.6.6. Finite element discretization of the Sturm-Liouville problem. The finite element discretization of the Sturm-Liouville problem is given by:

Find a *trial function* $u_\mathcal{T} \in S_0^{k,0}(\mathcal{T})$ such that every *test function* $v_\mathcal{T} \in S_0^{k,0}(\mathcal{T})$ satisfies
$$\int_0^1 \big[p(x)u_\mathcal{T}'(x)v_\mathcal{T}'(x) + q(x)u_\mathcal{T}(x)v_\mathcal{T}(x)\big]dx = \int_0^1 f(x)v_\mathcal{T}(x)dx.$$

It has the following properties:

It admits a unique solution.
Its solution is the unique *minimum* in $S_0^{k,0}(\mathcal{T})$ of the *energy function.*
After choosing a basis for $S_0^{k,0}(\mathcal{T})$ it amounts to a linear

system of equations with $k \cdot \sharp\mathcal{T} - 1$ unknowns and a tridiagonal symmetric positive definite matrix, the so-called *stiffness matrix*.
Integrals are usually approximately evaluated using a quadrature formula.
In most case one chooses $k = 1$ (*linear elements*) or $k = 2$ (*quadratic elements*).
One usually chooses a *nodal basis* for $S_0^{k,0}(\mathcal{T})$ (cf. figure I.6.2 and section I.6.7).

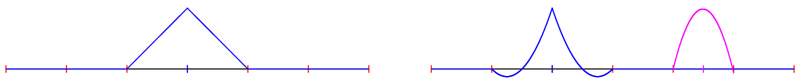

FIGURE I.6.2. Nodal basis functions for linear elements (left, blue) and for quadratic elements (right, endpoints of intervals blue and midpoints of intervals magenta)

I.6.7. Nodal basis functions. The *nodal basis functions* for linear elements are those functions which take the value 1 at exactly one endpoint of an interval and which vanish at all other endpoints of intervals (cf. left part of figure I.6.2).
The *nodal basis functions* for quadratic elements are those functions which take the value 1 at exactly one endpoint or midpoint of an interval and which vanish at all other endpoints and midpoints of intervals (right part of figure I.6.2, endpoints of intervals blue and midpoints of intervals magenta).

I.6.8. Error estimates. Denoting by $h_\mathcal{T}$ the maximal length of intervals in $\mathcal{T}$, one can prove the following *a priori error estimates* for the finite element discretization of the Sturm-Liouville problem:

Suppose that q is non-negative, $\underline{p} > 0$ and the derivative of p and the first and second derivative of the solution u of the Sturm-Liouville problem are in $L^2(0,1)$, then the following *a priori error estimate* holds

$$\|u' - u'_\mathcal{T}\| \le c_1 h_\mathcal{T},$$
$$\|u - u_\mathcal{T}\| \le c_2 h_\mathcal{T}^2.$$

The constants c_1 und c_2 depend on $\underline{p}$, the derivative of p, the maximum of q and the first and second derivatives of u.

CHAPTER II

Prerequisites for Finite Element and Finite Volume Methods

II.1. Sobolev Spaces

II.1.1. Reaction-diffusion equation. As a motivation for the Sobolev and finite element spaces, which will be introduced in this and the following section, consider the following *reaction-diffusion equation*

$$\begin{aligned} -\operatorname{div}(A\nabla u) + \alpha u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \Gamma \end{aligned}$$

which is a multi-dimensional generalization of the Sturm-Liouville problem of sections I.5 and I.6. Here, Ω is a polyhedron in $\mathbb{R}^d$ with $d = 2$ or $d = 3$, $A(x)$ is a symmetric positive definite, $d \times d$ matrix for every x in Ω and $\alpha(x)$ a non-negative number for every x in Ω.

II.1.2. The divergence theorem and integration by parts in several dimensions. Similarly to section I.6 we want to derive a variational formulation of the reaction-diffusion equation. Again this will be based on integration by parts.

To explain this further we first recall the definition of the *divergence* of a vector-field

$$\boxed{\operatorname{div}\mathbf{w} = \sum_{i=1}^{d} \frac{\partial w_i}{\partial x_i}}$$

and the *divergence theorem* alias *Gauß theorem*

$$\boxed{\int_\Omega \operatorname{div}\mathbf{w}\,dx = \int_\Gamma \mathbf{w}\cdot\mathbf{n}\,dS.}$$

Applying the divergence theorem to $\mathbf{w} = v(A\nabla u)$ yields

$$\begin{aligned} \int_\Omega v \operatorname{div}(A\nabla u)dx + \int_\Omega \nabla v \cdot A\nabla u dx &= \int_\Omega \operatorname{div}(vA\nabla u)dx \\ &= \int_\Omega \operatorname{div}\mathbf{w} dx \end{aligned}$$

$$\begin{aligned} &= \int_\Gamma \mathbf{w}\cdot\mathbf{n}dS \\ &= \int_\Gamma v\mathbf{n}\cdot A\nabla u dS. \end{aligned}$$

If $v = 0$ on Γ, this in particular implies

$$\int_\Omega \nabla v\cdot A\nabla u dx = -\int_\Omega v\operatorname{div}(A\nabla u)dx.$$

On the other hand, applying the divergence theorem to $\mathbf{w} = uv\mathbf{e}_i$, where $\mathbf{e}_i$ denotes the i-th unit vector, gives

$$\begin{aligned} \int_\Omega \frac{\partial u}{\partial x_i}vdx + \int_\Omega u\frac{\partial v}{\partial x_i}dx &= \int_\Omega \frac{\partial(uv)}{\partial x_i}dx \\ &= \int_\Omega \operatorname{div}\mathbf{w}dx \\ &= \int_\Gamma \mathbf{w}\cdot\mathbf{n}dS \\ &= \int_\Gamma uv\mathbf{n}_i dS. \end{aligned}$$

If $u = 0$ or $v = 0$ on Γ, this in particular implies

$$\int_\Omega \frac{\partial u}{\partial x_i}vdx = -\int_\Omega u\frac{\partial v}{\partial x_i}dx.$$

II.1.3. Idea of the variational formulation. The idea of the variational formulation of the reaction-diffusion equation can be described as follows:

- Multiply the differential equation with a continuously differentiable function v with $v = 0$ on Γ to obtain
$$-\operatorname{div}(A\nabla u)(x)v(x) + \alpha(x)u(x)v(x) = f(x)v(x)$$
for $x \in \Omega$.
- Integrate the result over Ω
$$\int_\Omega \big[-\operatorname{div}(A\nabla u)v + \alpha uv\big]dx = \int_\Omega fvdx.$$
- Use integration by parts for the term containing derivatives
$$-\int_\Omega \operatorname{div}(A\nabla u)vdx = \int_\Omega \nabla v\cdot A\nabla u dx.$$

In order to put these ideas on a sound basis the following problems must be settled:

- The properties of the functions u and v must be stated more precisely.
- Classical properties such as ‘continuously differentiable’ are too restrictive.
- The notion ‘derivative’ must be generalized.

- In view of the discrete problems, piecewise differentiable functions should be differentiable in the new weaker sense.

II.1.4. Weak derivatives. The second integration by parts formula of section II.1.2 motivates the following notion of a weak derivative which generalizes the classical partial derivative.

The function u is said to be *weakly differentiable w.r.t.* x_i with *weak derivative* w_i, if every continuously differentiable function v with $v = 0$ on Γ satisfies

$$\int_\Omega w_i v dx = -\int_\Omega u \frac{\partial v}{\partial x_i} dx.$$

If u is weakly differentiable w.r.t. to all variables $x_1, \ldots, x_d$, we call u *weakly differentiable* and write ∇u for the vector $(w_1, \ldots, w_d)$ of the weak derivatives.

Example II.1.1. Every function which is continuously differentiable in the classical sense is weakly differentiable and its classical derivative coincides with the weak derivative.
Every continuous piecewise differentiable function is weakly differentiable and its weak derivative is the piecewise classical derivative.
The function $|x|$ is not differentiable in $(-1, 1)$, but it is differentiable in the weak sense. Its weak derivative is the piecewise constant function which equals -1 on $(-1, 0)$ and 1 on $(0, 1)$.

II.1.5. Sobolev spaces and norms. Here, we will only introduce the first order Sobolev space. Its definition is based on the notion of weak derivatives introduced above and of the *Lebesgue space* $L^2(\Omega)$:

$\|v\| = \left\{ \int_\Omega |v|^2 dx \right\}^{\frac{1}{2}}$ denotes the L^2-norm.
$L^2(\Omega)$ is the *Lebesgue space* of all functions v with finite L^2-norm $\|v\|$.
$H^1(\Omega)$ is the *Sobolev space* of all functions v in $L^2(\Omega)$, which are weakly differentiable and for which $|\nabla v|$, the Euclidean norm of ∇v, is in $L^2(\Omega)$.
$H_0^1(\Omega)$ is the *Sobolev space* of all functions v in $H^1(\Omega)$ with $v = 0$ on Γ.

Higher order Sobolev spaces can be defined in a similar way by first introducing higher order weak derivatives similarly to section II.1.4 and by then considering all functions in $L^2(\Omega)$ which have all their weak derivatives up to a given order contained in $L^2(\Omega)$.

Example II.1.2. Every bounded function is in $L^2(\Omega)$.
Every continuously differentiable function is in $H^1(\Omega)$.

A piecewise differentiable function is in $H^1(\Omega)$, if and only if it is globally continuous.
Functions in $H^1(\Omega)$ must not admit point values (see examples II.1.3 and II.1.4 below).

EXAMPLE II.1.3 (Radially symmetric functions in $\mathbb{R}^2$). Denote by Ω the circle with radius 1 and centre at the origin. Given a real number $\alpha \in \mathbb{R}$ set

$$v_\alpha(x,y) = \left(x^2 + y^2\right)^{\frac{\alpha}{2}}.$$

We then have

$$\int_\Omega v_\alpha^2 dxdy = 2\pi \int_0^1 r^{2\alpha} r dr < \infty$$
$$\iff 2\alpha + 1 > -1 \iff \alpha > -1$$

and

$$\int_\Omega |\nabla v_\alpha|^2 dxdy = 2\pi \int_0^1 \alpha^2 r^{2\alpha - 2} r dr < \infty$$
$$\iff 2\alpha - 1 > -1 \iff \alpha > 0.$$

Hence we conclude that $v_\alpha \in H^1(\Omega)$ if and only if $\alpha > 0$.
Notice that $v(x) = \ln(|\ln(\sqrt{x^2 + y^2})|)$ is in $H^1(\Omega)$ but has no finite value at the origin.

EXAMPLE II.1.4 (Radially symmetric functions in $\mathbb{R}^3$). Denote by Ω the ball with radius 1 and centre at the origin. Given a real number $\alpha \in \mathbb{R}$ set

$$v_\alpha(x,y,z) = \left(x^2 + y^2 + z^2\right)^{\frac{\alpha}{2}}.$$

We then have

$$\int_\Omega v_\alpha^2 dxdydz = 4\pi \int_0^1 r^{2\alpha} r^2 dr < \infty$$
$$\iff 2\alpha + 2 > -1 \iff \alpha > -\frac{3}{2}$$

and

$$\int_\Omega |\nabla v_\alpha|^2 dxdydz = 4\pi \int_0^1 \alpha^2 r^{2\alpha - 2} r^2 dr < \infty$$
$$\iff 2\alpha > -1 \iff \alpha > -\frac{1}{2}.$$

Hence we conclude that $v_\alpha \in H^1(\Omega)$ if and only if $\alpha > -\frac{1}{2}$. In particular the function $v(x) = \left(x^2 + y^2 + y^2\right)^{-\frac{1}{8}}$ is in $H^1(\Omega)$ but has no finite value at the origin.

II.1.6. Variational problem. The variational formulation of the reaction-diffusion equation now takes the form:

Find $u \in H_0^1(\Omega)$ such that for all $v \in H_0^1(\Omega)$
$$\int_\Omega \big[\nabla v \cdot A \nabla u + \alpha u v\big] dx = \int_\Omega f v dx.$$

It has the following properties:

The variational problem admits a unique solution.
The solution of the variational problem is the unique *minimum* in $H_0^1(\Omega)$ of the *energy function*
$$\frac{1}{2}\int_\Omega \big[\nabla u \cdot A \nabla u + \alpha u^2\big] dx - \int_\Omega f u dx.$$

II.1.7. Convective derivatives. General second order elliptic differential equations also contain so-called *convective derivatives* of the form $\mathbf{a} \cdot \nabla u$. They give rise to the additional term $\displaystyle\int_\Omega \mathbf{a} \cdot \nabla u v$ on the left-hand side of the variational problem. Now, the solution of the variational problem cannot be interpreted as the minimum of an energy function.

II.1.8. Neumann boundary conditions. The boundary condition $u = 0$ on Γ considered so far is usually called a *Dirichlet boundary condition*. Physically it describes a clamped membrane or prescribed temperature profile. In practice of course also free membranes or problems with a prescribed heat flux have to be modelled. This requires a so-called *Neumann boundary condition* of the form $\mathbf{n} \cdot A \nabla u = g$ on Γ_N where g is a given flux and Γ_N is a subset of Γ which may equal Γ. It gives rise to the additional term $\displaystyle\int_{\Gamma_N} g v$ on the right-hand side of the variational problem.

II.1.9. Weak divergence. Many engineering problems, e.g. the equations of linearized elasticity, require so-called mixed energy principles which are based on spaces different from the Sobolev spaces considered so far. The most prominent representative of these spaces is $H(\operatorname{div};\Omega)$. It is based on the notion of a *weak divergence*:

A vector-field $\mathbf{u} : \Omega \subset \mathbb{R}^d \to \mathbb{R}^d$ is said to have the *weak divergence* $w : \Omega \to \mathbb{R}$ if every continuously differentiable scalar function v satisfies

$$\int_\Omega wv = -\int_\Omega \mathbf{u} \cdot \nabla v.$$

If $\mathbf{u}$ has the weak divergence w, one writes $w = \operatorname{div} \mathbf{u}$.

Notice that if $\mathbf{u}$ is continuously differentiable, it has a weak divergence which coincides with the classical divergence.

The space $H(\operatorname{div}; \Omega)$ is now defined by

$$H(\operatorname{div}; \Omega) = \left\{ \mathbf{u} : \Omega \to \mathbb{R}^d : \mathbf{u} \in L^2(\Omega)^d \text{ and } \operatorname{div} \mathbf{u} \in L^2(\Omega) \right\}.$$

Note that a piecewise differentiable vector-field is in $H(\operatorname{div}; \Omega)$, if and only if its normal component is continuous across interfaces.

II.2. Finite Element Spaces

II.2.1. Basic idea. The basic idea of the finite element method can be described as follows:

- Subdivide Ω into non-overlapping simple sub-domains called elements such as triangles, parallelograms, tetrahedra or parallelepipeds, ... *(partition)*.
- In the variational problem replace the space $H_0^1(\Omega)$ by a finite dimensional subspace consisting of continuous functions which are element-wise polynomials *(finite element space)*.
- This gives rise to a linear system of equations for the approximation $u_{\mathcal{T}}$ of the solution u of the differential equation.

In order to make these ideas operative, we will address the following topics in what follows:

- construction of the partition,
- construction and properties of the finite element spaces,
- building of the linear system of equations.

Methods for efficiently solving the discrete problems will be the subject of chapter III.

II.2.2. Finite element partitions. In what follows $\mathcal{T}$ denotes a *partition* of the computational domain Ω into subsets called *elements* and labelled K. It has to satisfy the following conditions:

- $\Omega \cup \Gamma$ is the union of all elements in $\mathcal{T}$.
- *(Affine equivalence)* Each $K \in \mathcal{T}$ is either a triangle or a parallelogram, if $d = 2$, or a tetrahedron or a parallelepiped, if $d = 3$.

- *(Admissibility)* Any two elements in $\mathcal{T}$ are either disjoint or share a vertex or a complete edge or – if $d = 3$ – a complete face (see figure II.2.1).

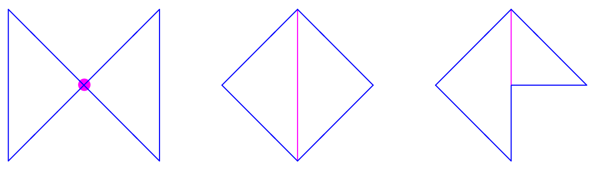

FIGURE II.2.1. Admissible (left and middle) and not admissible partition (right)

In two dimensions triangles and parallelograms may be mixed (cf. figure II.2.2). In three dimensions tetrahedrons and parallelepipeds can be mixed provided prismatic elements are also incorporated.
The condition of affine equivalence may be dropped. It, however, considerably simplifies the analysis since it implies constant Jacobians for all element transformations.
The admissibility is necessary to ensure the continuity of the finite element functions and thus the inclusion of the finite element spaces in $H_0^1(\Omega)$.
If the admissibility is violated, the continuity of the finite element functions must be enforced which leads to a more complicated implementation.
Curved boundaries can be approximated by piecewise straight lines or planes.

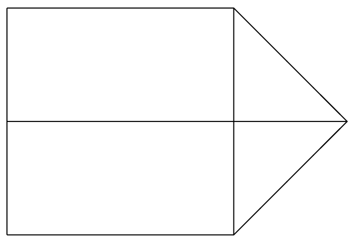

FIGURE II.2.2. Mixture of triangular and quadrilateral elements

II.2.3. Finite element spaces. For any multi-index $\alpha \in \mathbb{N}^d$ we set for abbreviation

$$|\alpha|_1 = \alpha_1 + \ldots + \alpha_d,$$
$$|\alpha|_\infty = \max\{\alpha_i : 1 \le i \le d\},$$
$$x^\alpha = x_1^{\alpha_1} \cdot \ldots \cdot x_d^{\alpha_d}.$$

Denote by

$$\widehat{K} = \{\widehat{x} \in \mathbb{R}^d : x_1 + \ldots + x_d \leq 1, x_i \geq 0, 1 \leq i \leq d\}$$

the *reference simplex* for a partition into triangles or tetrahedra and by

$$\widehat{K} = [0,1]^d$$

the *reference cube* for a partition into parallelograms or parallelepipeds. Then every element $K \in \mathcal{T}$ is the image of $\widehat{K}$ under an affine mapping F_K. For every integer number k set

$$R_k(\widehat{K}) = \begin{cases} \operatorname{span}\{x^\alpha : |\alpha|_1 \leq k\} & \text{,if } K \text{ is the reference simplex,} \\ \operatorname{span}\{x^\alpha : |\alpha|_\infty \leq k\} & \text{,if } K \text{ is the reference cube} \end{cases}$$

and set

$$R_k = R_k(K) = \left\{\widehat{p} \circ F_K^{-1} : \widehat{p} \in \widehat{R}_k\right\}.$$

With this notation we define finite element spaces by

$$\boxed{\begin{aligned} S^{k,-1}(\mathcal{T}) &= \left\{\varphi : \Omega \to \mathbb{R} : \varphi\Big|_K \in R_k(K) \text{ for all } K \in \mathcal{T}\right\}, \\ S^{k,0}(\mathcal{T}) &= S^{k,-1}(\mathcal{T}) \cap C(\overline{\Omega}), \\ S_0^{k,0}(\mathcal{T}) &= S^{k,0}(\mathcal{T}) \cap H_0^1(\Omega) = \left\{\varphi \in S^{k,0}(\mathcal{T}) : \varphi = 0 \text{ on } \Gamma\right\}. \end{aligned}}$$

Note, that k may be 0 for the first space, but must be at least 1 for the other spaces.
The global continuity ensures the inclusions $S^{k,0}(\mathcal{T}) \subset H^1(\Omega)$ and $S_0^{k,0}(\mathcal{T}) \subset H_0^1(\Omega)$.
The polynomial degree k may vary from element to element. This, however, leads to a more complicated implementation.

Example II.2.1. For a triangle, we have

$$\begin{aligned} R_1(K) &= \operatorname{span}\{1, x_1, x_2\}, \\ R_2(K) &= \operatorname{span}\{1, x_1, x_2, x_1^2, x_1x_2, x_2^2\}. \end{aligned}$$

For a parallelogram on the other hand, we have

$$\begin{aligned} R_1(K) &= \operatorname{span}\{1, x_1, x_2, x_1x_2\}, \\ R_2(K) &= \operatorname{span}\{1, x_1, x_2, x_1x_2, x_1^2, x_1^2x_2, x_1^2x_2^2, x_1x_2^2, x_2^2\}. \end{aligned}$$

II.2.4. Discrete problem. The finite element discretization of the reaction-diffusion equation of section II.1.1 and of its variational formulation of section II.1.6 is given by:

Find a *trial function* $u_{\mathcal{T}} \in S_0^{k,0}(\mathcal{T})$ such that for all *test functions* $v_{\mathcal{T}} \in S_0^{k,0}(\mathcal{T})$

$$\int_\Omega [\nabla v_{\mathcal{T}} \cdot A \nabla u_{\mathcal{T}} + \alpha u_{\mathcal{T}} v_{\mathcal{T}}] dx = \int_\Omega f v_{\mathcal{T}} dx.$$

It has the following properties:

The discrete problem admits a unique solution.
The solution of the discrete problem is the unique *minimum* in $S_0^{k,0}(\mathcal{T})$ of the *energy function*

$$\frac{1}{2} \int_\Omega [\nabla u \cdot A \nabla u + \alpha u^2] dx - \int_\Omega f u dx.$$

After choosing a basis for $S_0^{k,0}(\mathcal{T})$ the discrete problem amounts to a linear system of equations with $\approx k^d N_{\mathcal{T}}$ equations and unknowns where $N_{\mathcal{T}}$ is the number of elements.

II.2.5. Degrees of freedom. The basis functions of the finite element spaces $S^{k,0}(\mathcal{T})$ are defined by their *nodal degrees of freedom* $\mathcal{N}_{\mathcal{T},k}$. These are build by gluing together the element-wise degrees of freedom $\mathcal{N}_{K,k}$ by setting

$$\mathcal{N}_{\mathcal{T},k} = \bigcup_{K \in \mathcal{T}} \mathcal{N}_{K,k}.$$

Figures II.2.3 and II.2.4 show the element degrees and nodal degrees, resp. for different values of k.

Notice that the functions in $S^{k,0}(\mathcal{T})$ are uniquely defined by their values in $\mathcal{N}_{\mathcal{T},k}$ thanks to the admissibility of $\mathcal{T}$.

II.2.6. Nodal basis functions. The *nodal basis function* $\lambda_{z,k}$ associated with a vertex $z \in \mathcal{N}_{\mathcal{T},k}$ is uniquely defined by the conditions

$$\begin{aligned} \lambda_{z,k} &\in S^{k,0}(\mathcal{T}), \\ \lambda_{z,k}(z) &= 1, \\ \lambda_{z,k}(y) &= 0 \qquad \text{for all } y \in \mathcal{N}_{\mathcal{T},k} \setminus \{z\}. \end{aligned}$$

Figure II.2.5 shows a typical function $\lambda_{z,1}$.

The nodal basis functions have the following properties:

$\{\lambda_{z,k} : z \in \mathcal{N}_{\mathcal{T},k}\}$ is a *basis* for $S^{k,0}(\mathcal{T})$.
$\{\lambda_{z,k} : z \in \mathcal{N}_{\mathcal{T},k} \setminus \Gamma\}$ is a *basis* for $S_0^{k,0}(\mathcal{T})$, i.e. the degrees

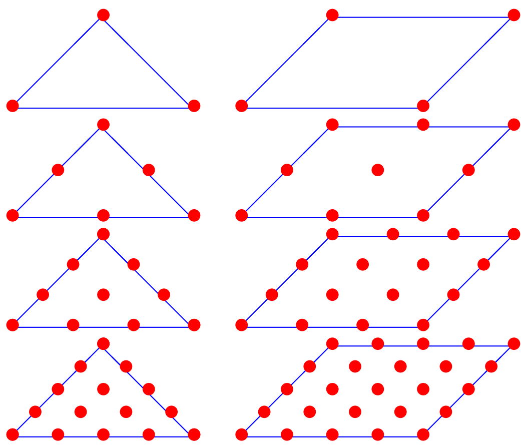
FIGURE II.2.3. Element degrees of freedom $\mathcal{N}_{K,k}$ for $k = 1, \ldots, 4$ and a triangle and a parallelogram

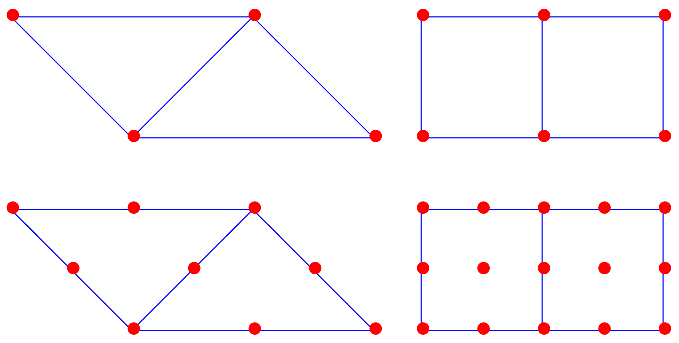
FIGURE II.2.4. Nodal degrees of freedom $\mathcal{N}_{\mathcal{T},k}$ for $k = 1, 2$ and a patch $\mathcal{T}$ consisting of two triangles or two parallelograms

of freedom on the boundary Γ are suppressed.
$\lambda_{z,k}$ vanishes outside the union of all elements that share the vertex z.
The stiffness matrix is sparse.

II.2.7. Evaluation of the nodal basis functions. Building the stiffness matrix and the load vector of the discrete problem requires the evaluation of the functions $\lambda_{z,k}$ and their derivatives. This can be achieved in two ways:

- transformation to a *reference element* $\widehat{K}$,
- reduction to the first order nodal basis functions $\lambda_{z,1}$ and direct evaluation of these using the element geometry.

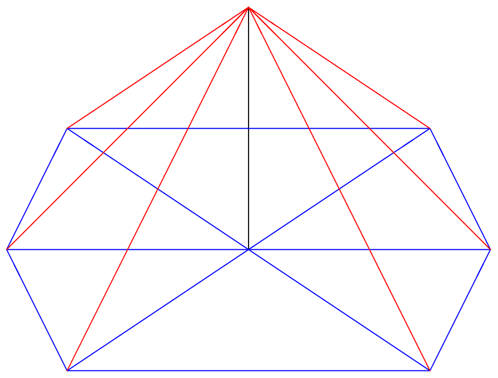

FIGURE II.2.5. Nodal basis function $\lambda_{z,1}$

In the first approach one proceeds as follows:

- Determine the nodal basis functions $\widehat{\lambda}_{\widehat{z},k}$ for the reference element $\widehat{K}$.
- Determine an *affine transformation*
$$\widehat{K} \ni \widehat{x} \mapsto x = b_K + B_K \widehat{x}$$
of the reference element $\widehat{K}$ onto the current element K.
- Compute $\lambda_{z,k}$ from $\widehat{\lambda}_{\widehat{z},k}$ using the affine transformation by setting
$$\lambda_{z,k}(x) = \widehat{\lambda}_{\widehat{z},k}(\widehat{x}).$$

Figure II.2.6 shows the commonly used reference elements in two and three dimensions.

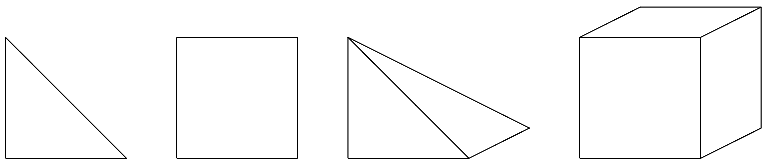

FIGURE II.2.6. Reference triangle, square, tetrahedron and cube (from left to right)

EXAMPLE II.2.2. For the reference triangle the nodal basis functions $\widehat{\lambda}_{\widehat{z},1}$ for the vertices are
$$1 - x - y, \quad x, \quad y.$$
The nodal basis functions $\widehat{\lambda}_{\widehat{z},2}$ for the vertices are
$$(1 - x - y)(1 - 2x - 2y), \quad x(2x - 1), \quad y(2y - 1)$$

and for the mid-points of edges

$$4x(1-x-y), \quad 4xy, \quad 4y(1-x-y).$$

For the reference square the nodal basis functions $\widehat{\lambda}_{\widehat{z},1}$ for the vertices are

$$(1-x)(1-y), \quad x(1-y), \quad xy, \quad (1-x)y.$$

The nodal basis functions $\widehat{\lambda}_{\widehat{z},2}$ for the vertices are

$$\begin{gathered}(1-2x)(1-x)(1-2y)(1-y), \quad x(2x-1)(1-2y)(1-y), \\ x(2x-1)y(2y-1), \quad (1-2x)(1-x)y(2y-1)\end{gathered}$$

and for the mid-points of edges

$$\begin{gathered}4x(1-x)(1-y)(1-2y), \quad 4x(2x-1)y(1-y), \\ 4x(1-x)y(2y-1), \quad 4y(1-y)(1-2x)(1-x)\end{gathered}$$

and for the barycentre

$$16x(1-x)y(1-y).$$

Example II.2.3. The vector b_K and the matrix B_K of the affine transformation of a triangle, parallelogram or tetrahedron are (see figure II.2.7 for the enumeration of the vertices)

$$\begin{aligned} b_K &= \mathbf{a}_0, \quad B_K = \big(\mathbf{a}_1 - \mathbf{a}_0\,,\, \mathbf{a}_2 - \mathbf{a}_0\big), \\ b_K &= \mathbf{a}_0, \quad B_K = \big(\mathbf{a}_1 - \mathbf{a}_0\,,\, \mathbf{a}_3 - \mathbf{a}_0\big), \\ b_K &= \mathbf{a}_0, \quad B_K = \big(\mathbf{a}_1 - \mathbf{a}_0\,,\, \mathbf{a}_2 - \mathbf{a}_0\,,\, \mathbf{a}_3 - \mathbf{a}_0\big). \end{aligned}$$

Similar formulae hold for parallelepipeds.

Example II.2.4. The first order nodal basis functions $\lambda_{\mathbf{a}_i,1}$ corresponding to a vertex $\mathbf{a}_i$ of a triangle, parallelogram or tetrahedron are given by (see figure II.2.7 for the enumeration of the vertices)

$$\begin{aligned} &\frac{\det(x - \mathbf{a}_{i+1}\,,\, \mathbf{a}_{i+2} - \mathbf{a}_{i+1})}{\det(\mathbf{a}_i - \mathbf{a}_{i+1}\,,\, \mathbf{a}_{i+2} - \mathbf{a}_{i+1})}, \\ &\frac{\det(x - \mathbf{a}_{i+2}\,,\, \mathbf{a}_{i+3} - \mathbf{a}_{i+2})}{\det(\mathbf{a}_i - \mathbf{a}_{i+2}\,,\, \mathbf{a}_{i+3} - \mathbf{a}_{i+2})} \cdot \frac{\det(x - \mathbf{a}_{i+2}\,,\, \mathbf{a}_{i+1} - \mathbf{a}_{i+2})}{\det(\mathbf{a}_i - \mathbf{a}_{i+2}\,,\, \mathbf{a}_{i+1} - \mathbf{a}_{i+2})}, \\ &\frac{\det(x - \mathbf{a}_{i+1}, \mathbf{a}_{i+2} - \mathbf{a}_{i+1}, \mathbf{a}_{i+3} - \mathbf{a}_{i+1})}{\det(\mathbf{a}_i - \mathbf{a}_{i+1}, \mathbf{a}_{i+2} - \mathbf{a}_{i+1}, \mathbf{a}_{i+3} - \mathbf{a}_{i+1})}. \end{aligned}$$

Here, all indices have to be taken modulo the number of vertices of the current element. Similar expressions hold for parallelepipeds with 3 factors corresponding to 3 tetrahedra.

Example II.2.5. The second order nodal basis function $\lambda_{\mathbf{a}_i,2}$ corresponding to a vertex $\mathbf{a}_i$ of a triangle is given by

$$\lambda_{\mathbf{a}_i,2} = \lambda_{\mathbf{a}_i,1}[\lambda_{\mathbf{a}_i,1} - \lambda_{\mathbf{a}_{i+1},1} - \lambda_{\mathbf{a}_{i+2},1}].$$

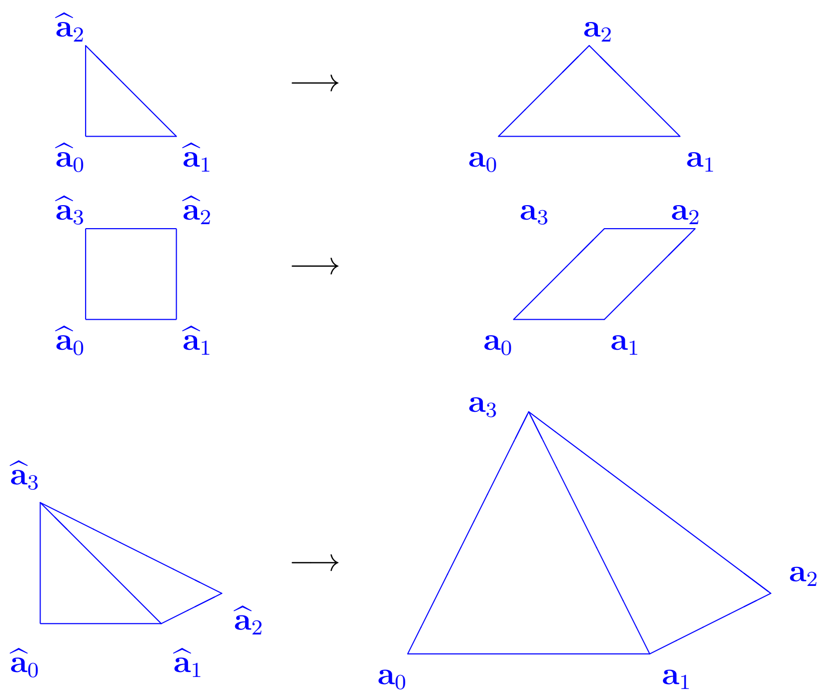

FIGURE II.2.7. Affine transformation of a triangle, parallelogram and tetrahedron

Similarly, the function $\lambda_{z,2}$ corresponding to the mid-point z of the edge connecting vertices $\mathbf{a}_i$ and $\mathbf{a}_{i+1}$ is

$$\lambda_{z,2} = 4\lambda_{\mathbf{a}_i,1}\lambda_{\mathbf{a}_{i+1},1}.$$

For a parallelogram one obtains for a vertex $\mathbf{a}_i$

$$\lambda_{\mathbf{a}_i,2} = \lambda_{\mathbf{a}_i,1}[\lambda_{\mathbf{a}_i,1} - \lambda_{\mathbf{a}_{i+1},1} + \lambda_{\mathbf{a}_{i+2},1} - \lambda_{\mathbf{a}_{i+3},1}],$$

for the mid-point z of the edge connecting vertices $\mathbf{a}_i$ and $\mathbf{a}_{i+1}$

$$\lambda_{z,2} = 4\lambda_{\mathbf{a}_i,1}[\lambda_{\mathbf{a}_{i+1},1} - \lambda_{\mathbf{a}_{i+2},1}]$$

and for the barycentre y

$$\lambda_{y,2} = 16\lambda_{\mathbf{a}_0,1}\lambda_{\mathbf{a}_2,1}.$$

II.2.8. Evaluation of integrals. The exact evaluation of the integrals appearing in the entries of the stiffness matrix and load vector often is too expensive or even impossible. They are therefore approximately evaluated using a suitable *quadrature formula*:

$$\boxed{\int_K \varphi dx \approx Q_k(\varphi) = \sum_{q \in \mathcal{Q}_K} c_q \varphi(q).}$$

In order to avoid that this spoils the accuracy of the finite element discretization, the quadrature formula must have the *order* $2k-2$, where k is the element degree, i.e.

$$\int_K \varphi dx = Q_K(\varphi) \quad \text{for all } \varphi \in R_{2k-2}(K).$$

The order 0 is sufficient for linear elements; order 2 is sufficient for quadratic elements.

Example II.2.6. The data

$$\mathcal{Q}_K \quad \text{barycentre of } K,$$
$$c_q = |K|$$

and

$$\mathcal{Q}_K \quad \text{mid-points of edges of } K,$$
$$c_q = \frac{1}{3}|K| \quad \text{for alle } q$$

yield quadrature formulae of order 1 and 2, resp. for triangles. Here, $|K|$ denotes the area of K.
The data

$$\mathcal{Q}_K \quad \text{barycentre of } K,$$
$$c_q = |K|$$

and

$$\mathcal{Q}_K \quad \text{vertices, mid-points of edges and barycentre of } K,$$
$$c_q = \begin{cases} \frac{1}{36}|K| & \text{if } q \text{ is a vertex} \\ \frac{4}{36}|K| & \text{if } q \text{ is a mid-point of edge} \\ \frac{16}{36}|K| & \text{if } q \text{ is the barycentre} \end{cases}$$

yield quadrature formulae of order 1 and 2, resp. for parallelograms. Here, $|K|$ again denotes the area of K.
Similar formulae can be derived for tetrahedra and parallelepipeds.

II.2.9. Convective derivatives. Convective derivatives lead to a non-symmetric stiffness matrix. They often give rise to unphysical oscillations of the numerical solution. To avoid these oscillations special modifications such as *upwinding* or *streamline Petrov-Galerkin stabilization* must be introduced (cf. [4, §II.3] and [5, §II.3.1.2]).

II.2.10. Neumann boundary conditions. The so-called *Neumann boundary condition* $\mathbf{n} \cdot A\nabla u = g$ on $\Gamma_N \subset \Gamma$ gives rise to an additional term $\int_{\Gamma_N} g v_{\mathcal{T}} dS$ on the right-hand side of the discrete problem. The additional entries of the load vector are taken into account when sweeping through the elements. Moreover, degrees of freedom associated with points on the Neumann boundary Γ_N are additional unknowns.

II.3. Finite Volume Methods

II.3.1. Systems in divergence form. Finite volume methods are tailored for *systems in divergence form* where we are looking for a vector field $\mathbf{U}$ defined on a subset Ω of $\mathbb{R}^d$ having values in $\mathbb{R}^m$ which satisfies the differential equation

$$\boxed{\begin{aligned} \frac{\partial \mathbf{M}(\mathbf{U})}{\partial t} + \operatorname{div} \underline{\mathbf{F}}(\mathbf{U}) &= \mathbf{g}(\mathbf{U}, x, t) && \text{in } \Omega \times (0,\infty) \\ \mathbf{U}(\cdot, 0) &= \mathbf{U}_0 && \text{in } \Omega. \end{aligned}}$$

Here,

g: the *source*, is a vector field on $\mathbb{R}^m \times \Omega \times (0,\infty)$ with values in $\mathbb{R}^m$,
M: the *mass*, is a vector field on $\mathbb{R}^m$ with values in $\mathbb{R}^m$,
$\underline{\mathbf{F}}$: the *flux* is a matrix valued function on $\mathbb{R}^m$ with values in $\mathbb{R}^{m \times d}$ and
$\mathbf{U}_0$: the *initial value*, is a vector field on Ω with values in $\mathbb{R}^m$.

The differential equation of course has to be completed with suitable boundary conditions. These, however, will be ignored in what follows.

Notice that the divergence has to be taken row-wise

$$\operatorname{div} \underline{\mathbf{F}}(\mathbf{U}) = \Big(\sum_{j=1}^{d} \frac{\partial \underline{\mathbf{F}}(\mathbf{U})_{i,j}}{\partial x_j} \Big)_{1 \le i \le m}.$$

The flux $\underline{\mathbf{F}}$ can be slit into two contributions

$$\underline{\mathbf{F}} = \underline{\mathbf{F}}_{\text{adv}} + \underline{\mathbf{F}}_{\text{visc}}.$$

$\underline{\mathbf{F}}_{\text{adv}}$is called *advective flux* and contains no derivatives. $\underline{\mathbf{F}}_{\text{visc}}$ is called *viscous flux* and contains spacial derivatives. The advective flux models transport or convection phenomena while the viscous flux is responsible for diffusion phenomena.

Example II.3.1. A linear parabolic equation of 2nd order

$$\frac{\partial u}{\partial t} - \operatorname{div}(A \nabla u) + \mathbf{a} \cdot \nabla u + \alpha u = f,$$

is a system in divergence form with

$$\begin{aligned} &m = 1, && \mathbf{U} = u, && \mathbf{M}(\mathbf{U}) = u, \\ &\underline{\mathbf{F}}_{\text{adv}}(\mathbf{U}) = \mathbf{a} u, && \underline{\mathbf{F}}_{\text{visc}}(\mathbf{U}) = -A \nabla u, && \mathbf{g}(\mathbf{U}) = f - \alpha u + (\operatorname{div} \mathbf{a}) u. \end{aligned}$$

Example II.3.2. *Burger's equation*

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0$$

is a system in divergence form with

$$m = d = 1, \qquad \mathbf{u} = u, \qquad \mathbf{M}(\mathbf{U}) = u,$$

$$\underline{\mathbf{F}}_{\mathrm{adv}}(u) = \frac{1}{2}u^2, \qquad \underline{\mathbf{F}}_{\mathrm{visc}}(\mathbf{U}) = 0, \qquad \mathbf{g}(\mathbf{U}) = 0.$$

Other important examples of systems in divergence form are the *Euler equations* and *Navier-Stokes equations* for non-viscous respective viscous fluids. Here we have $d = 2$ or $d = 3$ and $m = d+2$. The vector $\mathbf{U}$ consists of the density, velocity and the internal energy of the fluid (cf. [4, §IV.3]).

II.3.2. Basic idea of the finite volume method. Choose a time step $\tau > 0$ and a partition $\mathcal{T}$ of Ω consisting of arbitrary non-overlapping polyhedra. Here, the elements may have more complicated shapes than in the finite element method (see figures II.3.1 and II.3.2). Moreover, admissibility is no longer required.

Now we choose an integer $n \geq 1$ and an element $K \in \mathcal{T}$ and keep both fixed in what follows. First we integrate the differential equation on $K \times [(n-1)\tau, n\tau]$

$$\int_{(n-1)\tau}^{n\tau} \int_K \frac{\partial \mathbf{M}(\mathbf{U})}{\partial t} dx dt + \int_{(n-1)\tau}^{n\tau} \int_K \operatorname{div} \underline{\mathbf{F}}(\mathbf{U}) dx dt$$
$$= \int_{(n-1)\tau}^{n\tau} \int_K \mathbf{g}(\mathbf{U}, x, t) dx dt.$$

Next we use integration by parts for the terms on the left-hand side

$$\int_{(n-1)\tau}^{n\tau} \int_K \frac{\partial \mathbf{M}(\mathbf{U})}{\partial t} dx dt = \int_K \mathbf{M}(\mathbf{U}(x, n\tau)) dx$$
$$- \int_K \mathbf{M}(\mathbf{U}(x, (n-1)\tau)) dx,$$
$$\int_{(n-1)\tau}^{n\tau} \int_K \operatorname{div} \underline{\mathbf{F}}(\mathbf{U}) dx dt = \int_{(n-1)\tau}^{n\tau} \int_{\partial K} \underline{\mathbf{F}}(\mathbf{U}) \cdot \mathbf{n}_K dS dt.$$

For the following steps we assume that $\mathbf{U}$ is piecewise constant with respect to space and time. We denote by $\mathbf{U}_K^n$ and $\mathbf{U}_K^{n-1}$ the value of $\mathbf{U}$ on K at times $n\tau$ und $(n-1)\tau$, respectively. Then we have

$$\int_K \mathbf{M}(\mathbf{U}(x, n\tau)) dx \approx |K| \mathbf{M}(\mathbf{U}_K^n)$$
$$\int_K \mathbf{M}(\mathbf{U}(x, (n-1)\tau)) dx \approx |K| \mathbf{M}(\mathbf{U}_K^{n-1})$$
$$\int_{(n-1)\tau}^{n\tau} \int_{\partial K} \underline{\mathbf{F}}(\mathbf{U}) \cdot \mathbf{n}_K dS dt \approx \tau \int_{\partial K} \underline{\mathbf{F}}(\mathbf{U}_K^{n-1}) \cdot \mathbf{n}_K dS$$
$$\int_{(n-1)\tau}^{n\tau} \int_K \mathbf{g}(\mathbf{U}, x, t) dx dt \approx \tau |K| \mathbf{g}(\mathbf{U}_K^{n-1}, x_K, (n-1)\tau).$$

Here, $|K|$ denotes the area of K, if $d = 2$, or the volume of K, if $d = 3$, respectively.

In a last step we approximate the boundary integral for the flux by a *numerical flux*

$$\begin{aligned}\tau \int_{\partial K} &\underline{\mathbf{F}}(\mathbf{U}_K^{n-1}) \cdot \mathbf{n}_K dS \\ \approx \tau &\sum_{\substack{K' \in \mathcal{T} \\ \partial K \cap \partial K' \in \mathcal{E}}} |\partial K \cap \partial K'| \mathbf{F}_{\mathcal{T}}(\mathbf{U}_K^{n-1}, \mathbf{U}_{K'}^{n-1}).\end{aligned}$$

Here, $\partial K \cap \partial K' \in \mathcal{E}$ means that K and K' share an edge, if $d = 2$, or a face, if $d = 3$, and $|\partial K \cap \partial K'|$ denotes the length respective area of the common boundary of $K \cap K'$.

All together we obtain the following *finite volume method*

> For every element $K \in \mathcal{T}$ compute
>
> $$\mathbf{U}_K^0 = \frac{1}{|K|} \int_K \mathbf{U}_0(x).$$
>
> For $n = 1, 2, \ldots$ successively compute for every element $K \in \mathcal{T}$ the quantity $\mathbf{U}_K^n$ such that
>
> $$\begin{aligned}\mathbf{M}(\mathbf{U}_K^n) = \mathbf{M}(\mathbf{U}_K^{n-1})& \\ - \tau &\sum_{\substack{K' \in \mathcal{T} \\ \partial K \cap \partial K' \in \mathcal{E}}} \frac{|\partial K \cap \partial K'|}{|K|} \mathbf{F}_{\mathcal{T}}(\mathbf{U}_K^{n-1}, \mathbf{U}_{K'}^{n-1}) \\ + \tau &\mathbf{g}(\mathbf{U}_K^{n-1}, x_K, (n-1)\tau).\end{aligned}$$

This method may be modified as follows:

- The time step may be variable.
- The partition of Ω may change from one time step to the other.
- The approximation $\mathbf{U}_K^n$ must not be piecewise constant.

In order to obtain an operating discretization, we still have to make precise the following topics:

- construction of $\mathcal{T}$,
- choice of $\underline{\mathbf{F}}_{\mathcal{T}}$.

Moreover we have to take into account boundary conditions. This item, however, will not be addressed in what follows.

II.3.3. Construction of dual finite volume meshes. For constructing the finite volume mesh $\mathcal{T}$, we start from a standard finite element partition $\widetilde{\mathcal{T}}$ which satisfies the conditions of section II.2.2. Then we subdivide each element $\widetilde{K} \in \widetilde{\mathcal{T}}$ into smaller elements by either

- drawing the perpendicular bisectors at the mid-points of edges of $\widetilde{K}$ (cf. figure II.3.1) or by
- connecting the barycentre of $\widetilde{K}$ with its mid-points of edges (cf. figure II.3.2).

Then the elements in $\mathcal{T}$ consist of the unions of all small elements that share a common vertex in the partition $\widetilde{\mathcal{T}}$.

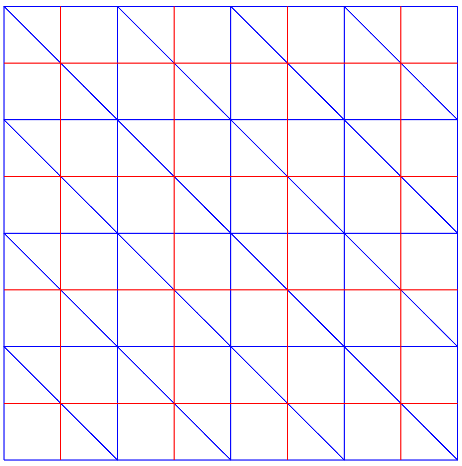

FIGURE II.3.1. Dual mesh (red) via perpendicular bisectors of primal mesh (blue)

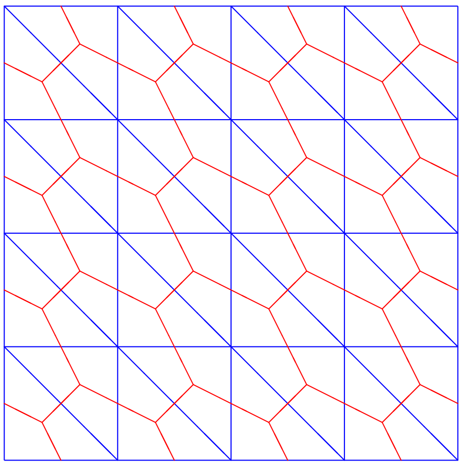

FIGURE II.3.2. Dual mesh (red) via barycentres of primal mesh (blue)

Thus the elements in $\mathcal{T}$ can be associated with the vertices in $\mathcal{N}_{\widetilde{\mathcal{T}}}$ (see left part of figure II.3.3). Moreover, we may associate with each edge or face in $\mathcal{E}_{\mathcal{T}}$ exactly two vertices in $\mathcal{N}_{\widetilde{\mathcal{T}}}$ such that the line connecting these vertices intersects the given edge or face, respectively (see right part of figure II.3.3).

The first construction has the advantage that this intersection is orthogonal. But this construction also has some disadvantages which are not present with the second construction:

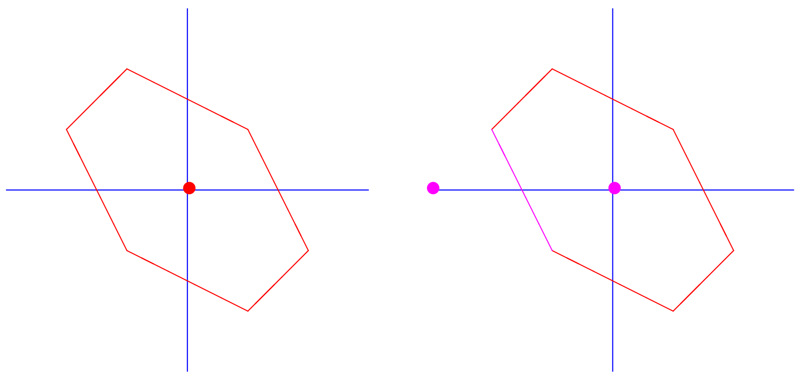

FIGURE II.3.3. Volume of the dual mesh associated with a vertex of the primal mesh (left) and vertices of the primal mesh associated with an edge of the dual mesh (right)

- The perpendicular bisectors of a triangle may intersect in a point outside the triangle. The intersection point is within the triangle only if its largest angle is at most a right one.
- The perpendicular bisectors of a quadrilateral may not intersect at all. They intersect in a common point inside the quadrilateral only if it is a rectangle.
- The first construction has no three dimensional analogue.

II.3.4. Construction of numerical fluxes. For the construction of numerical fluxes we assume that $\mathcal{T}$ is a dual mesh corresponding to a primal finite element partition $\widetilde{\mathcal{T}}$. With every edge or face E of $\mathcal{T}$ we denote by K_1 and K_2 the adjacent volumes, by $\mathbf{U}_1$ and $\mathbf{U}_2$ the values $\mathbf{U}_{K_1}^{n-1}$ and $\mathbf{U}_{K_2}^{n-1}$, respectively and by x_1, x_2 vertices of $\widetilde{\mathcal{T}}$such that the segment $\overline{x_1\, x_2}$ intersects E (see right part of figure II.3.3).

As in the analytical case, we split the numerical flux $\underline{\mathbf{F}}_{\mathcal{T}}(\mathbf{U}_1, \mathbf{U}_2)$ into a *viscous numerical flux* $\underline{\mathbf{F}}_{\mathcal{T},\mathrm{visc}}(\mathbf{U}_1, \mathbf{U}_2)$ and an *advective numerical flux* $\underline{\mathbf{F}}_{\mathcal{T},\mathrm{adv}}(\mathbf{U}_1, \mathbf{U}_2)$ which are constructed separately.

We first construct the *numerical viscous fluxes*. To this end we introduce a local coordinate system $\eta_1, \ldots, \eta_d$ such that η_1 is parallel to $\overline{x_1\, x_2}$ and such that the remaining coordinates are tangential to E (see figure II.3.4). Next we express all derivatives in $\underline{\mathbf{F}}_{\mathrm{visc}}$ in terms of partial derivatives corresponding to the new coordinates and suppress all derivatives which do not pertain to η_1. Finally we approximate derivatives corresponding to η_1 by differences of the form $\frac{\varphi_1 - \varphi_2}{|x_1 - x_2|}$.

We now construct the *numerical advective fluxes*. To this end we denote by $\mathbf{n}_{K_1}$ the unit outward normal of K_1 and by

$$C(\mathbf{V}) = D(\underline{\mathbf{F}}_{\mathrm{adv}}(\mathbf{V}) \cdot \mathbf{n}_{K_1}) \in \mathbb{R}^{m \times m}$$

the derivative of $\underline{\mathbf{F}}_{\mathrm{adv}}(\mathbf{V}) \cdot \mathbf{n}_{K_1}$ with respect to $\mathbf{V}$ and suppose that this matrix can be diagonalized, i.e., there is an invertible matrix $Q(\mathbf{V}) \in$

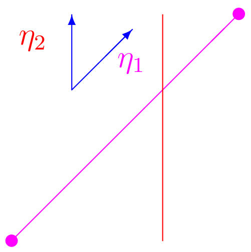

FIGURE II.3.4. Local coordinate system for the approximation of viscous fluxes

$\mathbb{R}^{m\times m}$ and a diagonal matrix $\Delta(\mathbf{V}) \in \mathbb{R}^{m\times m}$ such that

$$Q(\mathbf{V})^{-1}C(\mathbf{V})Q(\mathbf{V}) = \Delta(\mathbf{V}).$$

This assumption is, e.g., satisfied for the Euler and Navier-Stokes equations. With any real number z we then associate its positive and negative part

$$z^+ = \max\{z, 0\}, \quad z^- = \min\{z, 0\}$$

and set

$$\begin{aligned} \Delta(\mathbf{V})^\pm &= \operatorname{diag}\bigl(\Delta(\mathbf{V})^\pm_{11}, \ldots, \Delta(\mathbf{V})^\pm_{mm}\bigr), \\ C(\mathbf{V})^\pm &= Q(\mathbf{V})\Delta(\mathbf{V})^\pm Q(\mathbf{V})^{-1}. \end{aligned}$$

With these notations the *Steger-Warming scheme* for the approximation of advective fluxes is given by

$$\boxed{\mathbf{F}_{\mathcal{T},\mathrm{adv}}(\mathbf{U}_1, \mathbf{U}_2) = C(\mathbf{U}_1)^+\mathbf{U}_1 + C(\mathbf{U}_2)^-\mathbf{U}_2.}$$

A better approximation is the *van Leer scheme*

$$\boxed{\begin{aligned} &\mathbf{F}_{\mathcal{T},\mathrm{adv}}(\mathbf{U}_1, \mathbf{U}_2) \\ &= \Bigl[\frac{1}{2}C(\mathbf{U}_1) + C(\frac{1}{2}(\mathbf{U}_1 + \mathbf{U}_2))^+ - C(\frac{1}{2}(\mathbf{U}_1 + \mathbf{U}_2))^-\Bigr]\mathbf{U}_1 \\ &\quad + \Bigl[\frac{1}{2}C(\mathbf{U}_2) - C(\frac{1}{2}(\mathbf{U}_1 + \mathbf{U}_2))^+ + C(\frac{1}{2}(\mathbf{U}_1 + \mathbf{U}_2))^-\Bigr]\mathbf{U}_2. \end{aligned}}$$

Both approaches require the computation of $D\underline{\mathbf{F}}_{\mathrm{adv}}(\mathbf{V}) \cdot \mathbf{n}_{K_1}$ together with its eigenvalues and eigenvectors for suitable values of $\mathbf{V}$. In general the van Leer scheme is more costly than the Steger-Warming scheme since it requires three evaluations of $C(\mathbf{V})$ instead of two. For the Euler and Navier-Stokes equations, however, this extra cost can be avoided profiting from the particular structure $\underline{\mathbf{F}}_{\mathrm{adv}}(\mathbf{V}) \cdot \mathbf{n}_{K_1} = C(\mathbf{V})\mathbf{V}$ of these equations.

Example II.3.3. When applied to Burger's equation of example II.3.2 the Steger-Warming scheme takes the form

$$\underline{\mathbf{F}}_{\mathcal{T},\mathrm{adv}}(u_1,u_2) = \begin{cases} u_1^2 & \text{if } u_1 \ge 0, u_2 \ge 0 \\ u_1^2 + u_2^2 & \text{if } u_1 \ge 0, u_2 \le 0 \\ u_2^2 & \text{if } u_1 \le 0, u_2 \le 0 \\ 0 & \text{if } u_1 \le 0, u_2 \ge 0 \end{cases}$$

while the van Leer scheme reads

$$\underline{\mathbf{F}}_{\mathcal{T},\mathrm{adv}}(u_1,u_2) = \begin{cases} u_1^2 & \text{if } u_1 \ge -u_2 \\ u_2^2 & \text{if } u_1 \le -u_2. \end{cases}$$

II.3.5. TVD and ENO schemes. The convergence analysis of finite volume methods is based on compactness arguments, in particular the concept of *compensated compactness*. This requires to bound the *total variation* of the numerical approximation and to avoid unphysical oscillations. This leads to the concept of *t*otal *v*ariation *d*iminishing *TVD* and *e*ssentially *n*on-*o*scillating *ENO* schemes. Corresponding material may be found under the names of Enquvist, LeVeque, Osher, Roe, Tadmor, ...

II.3.6. Relation to finite element methods. The fact that the elements of a dual mesh can be associated with the vertices of a finite element partition gives a link between finite volume and finite element methods:

> Consider a function φ that is piecewise constant on the dual mesh $\mathcal{T}$, i.e. $\varphi \in S^{0,-1}(\mathcal{T})$. With φ we associate a continuous piecewise linear function $\Phi \in S^{1,0}(\widetilde{\mathcal{T}})$ corresponding to the finite element partition $\widetilde{\mathcal{T}}$ such that $\Phi(x_K) = \varphi_K$ for the vertex $x_K \in \mathcal{N}_{\widetilde{\mathcal{T}}}$ corresponding to $K \in \mathcal{T}$.

This link considerably simplifies the analysis of finite volume methods and suggests a very simple and natural approach to a posteriori error estimation and mesh adaptivity for finite volume methods (cf. [5, §II.4.11.5]).

CHAPTER III

Efficient Solvers for Linear Systems of Equations

III.1. Properties of Direct and Iterative Solvers

III.1.1. A model problem. To get an overview of the particularities of the solution of finite element problems, we consider a simple, but instructive model problem

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \Gamma \end{aligned}$$

with Ω denoting the unit square $(0,1)^2$ $(d = 2)$ or the unit cube $(0,1)^3$ $(d = 3)$ discretized by linear elements on a mesh that consists of squares $(d = 2)$ or cubes $(d = 3)$ with edges of length $h = \frac{1}{n}$.

The number of unknowns is

$$N_h = \Big(\frac{1}{n-2}\Big)^d.$$

The stiffness matrix L_h is symmetric positive definite and sparse; every row contains at most 3^d non-zero elements. The total number of non-zero entries in L_h is

$$e_h = 3^d N_h.$$

The ratio of non-zero entries to the total number of entries in L_h is

$$p_h = \frac{e_h}{N_h^2} \approx 3^d N_h^{-1}.$$

The stiffness matrix is a band matrix with bandwidth

$$b_h = h^{-d+1} \approx N_h^{1-\frac{1}{d}}.$$

Therefore the Gaussian elimination, the LR-decomposition or the Cholesky decomposition require

$$s_h = b_h N_h \approx N_h^{2-\frac{1}{d}}$$

bytes for storage and

$$z_h = b_h^2 N_h \approx N_h^{3-\frac{2}{d}}$$

arithmetic operations.

These numbers are collected in table III.1.1. It clearly shows that direct methods are not suited for the solution of large finite element problems both with respect to the storage requirement as with respect

TABLE III.1.1. Storage requirement and arithmetic operations of the Cholesky decomposition applied to the linear finite element discretization of the model problem on $(0,1)^d$

d	h	N_h	e_h	b_h	s_h	z_h
2	$\frac{1}{16}$	225	$1.1 \cdot 10^3$	15	$3.3 \cdot 10^3$	$7.6 \cdot 10^5$
	$\frac{1}{32}$	961	$4.8 \cdot 10^3$	31	$2.9 \cdot 10^4$	$2.8 \cdot 10^7$
	$\frac{1}{64}$	$3.9 \cdot 10^3$	$2.0 \cdot 10^4$	63	$2.5 \cdot 10^5$	$9.9 \cdot 10^8$
	$\frac{1}{128}$	$1.6 \cdot 10^4$	$8.0 \cdot 10^4$	127	$2.0 \cdot 10^6$	$3.3 \cdot 10^{10}$
3	$\frac{1}{16}$	$3.3 \cdot 10^3$	$2.4 \cdot 10^4$	225	$7.6 \cdot 10^5$	$1.7 \cdot 10^8$
	$\frac{1}{32}$	$3.0 \cdot 10^4$	$2.1 \cdot 10^5$	961	$2.8 \cdot 10^7$	$2.8 \cdot 10^{10}$
	$\frac{1}{64}$	$2.5 \cdot 10^5$	$1.8 \cdot 10^6$	$3.9 \cdot 10^3$	$9.9 \cdot 10^8$	$3.9 \cdot 10^{12}$
	$\frac{1}{128}$	$2.0 \cdot 10^6$	$1.4 \cdot 10^7$	$1.6 \cdot 10^4$	$3.3 \cdot 10^{10}$	$5.3 \cdot 10^{14}$

to the computational work. Therefore one usually uses iterative methods for the solution of large finite element problems. Their efficiency is essentially determined by the following considerations:

- The exact solution of the finite element problem is an approximation of the solution of the differential equation, which is the quantity of interest, with an error $O(h^k)$ where k is the polynomial degree of the finite element space. Therefore it is sufficient to compute an approximate solution of the discrete problem which has the same accuracy.
- If the mesh $\mathcal{T}_1$ is a global or local refinement of the mesh $\mathcal{T}_0$, the interpolate of the approximate discrete solution corresponding to $\mathcal{T}_0$ is a good initial guess for any iterative solver for the discrete problem corresponding to $\mathcal{T}_1$.

III.1.2. Nested iteration. The above considerations lead to the following *nested iteration*. Here $\mathcal{T}_0, \ldots, \mathcal{T}_R$ denotes a sequence of successively (globally or locally) refined meshes with corresponding finite element problems

$$L_k u_k = f_k \qquad 0 \le k \le R.$$

ALGORITHM III.1.1. (Nested iteration)

(1) *Compute*

$$\widetilde{u}_0 = u_0 = L_0^{-1} f_0.$$

(2) *For $k = 1, \ldots, R$ compute an approximate solution $\widetilde{u}_k$ for $u_k = L_k^{-1} f_k$ by applying m_k iterations of an iterative solver for*

the problem

$$L_k u_k = f_k$$

with starting value $I_{k-1,k}\widetilde{u}_{k-1}$, where $I_{k-1,k}$ is a suitable interpolation operator from the mesh $\mathcal{T}_{k-1}$ to the mesh $\mathcal{T}_k$.

Usually, the number m_k of iterations in algorithm III.1.1 is determined by the stopping criterion

$$\boxed{\|f_k - L_k\widetilde{u}_k\| \le \varepsilon\|f_k - L_k(I_{k-1,k}\widetilde{u}_{k-1})\|.}$$

That is, the residual of the starting value measured in an appropriate norm should be reduced by a factor ε. Typically, $\|\cdot\|$ is a weighted Euclidean norm and ε is in the realm 0.05 to 0.1. If the iterative solver has the convergence rate δ_k, the number m_k of iterations is given by

$$\boxed{m_k = \left\lceil \frac{\ln\varepsilon}{\ln\delta_k} \right\rceil.}$$

Table III.1.2 gives the number m_k of iterations that require the classical Gauß-Seidel algorithm, the conjugate gradient algorithm III.3.2 and the preconditioned conjugate gradient algorithm III.3.4 with SSOR-preconditioning III.3.5 for reducing an initial residual by the factor $\varepsilon = 0.1$. These algorithms need the following number of operations per unknown:

$$\begin{aligned} &2d+1 \quad \text{(Gauß-Seidel)},\\ &2d+6 \quad \text{(CG)},\\ &5d+8 \quad \text{(SSOR-PCG)}.\end{aligned}$$

Table III.1.2. Number of iterations required for reducing an initial residual by the factor 0.1

h	Gauß-Seidel	CG	SSOR-PCG
$\frac{1}{16}$	236	12	4
$\frac{1}{32}$	954	23	5
$\frac{1}{64}$	3820	47	7
$\frac{1}{128}$	15287	94	11

Table III.1.2 shows that the preconditioned conjugate gradient algorithm with SSOR-preconditioning yields satisfactory results for problems that are not too large. Nevertheless, its computational work is not proportional to the number of unknowns; for a fixed tolerance ε it approximately is of the order $N_h^{1+\frac{1}{2d}}$. The multigrid algorithm III.4.1 overcomes this drawback. Its convergence rate is independent of the mesh-size. Correspondingly, for a fixed tolerance ε, its computational work is proportional to the number of unknowns. The advantages of the multigrid algorithm are reflected by table III.1.3.

Table III.1.3. Arithmetic operations required by the preconditioned conjugate gradient algorithm with SSOR-preconditioning and the V-cycle multigrid algorithm with one Gauß-Seidel step for pre- and post-smoothing applied to the model problem in $(0,1)^d$

d	h	PCG-SSOR	multigrid
2	$\frac{1}{16}$	$16'200$	$11'700$
	$\frac{1}{32}$	$86'490$	$48'972$
	$\frac{1}{64}$	$500'094$	$206'988$
	$\frac{1}{128}$	$3'193'542$	$838'708$
3	$\frac{1}{16}$	$310'500$	$175'500$
	$\frac{1}{32}$	$3'425'965$	$1'549'132$
	$\frac{1}{64}$	$4.0 \cdot 10^7$	$1.3 \cdot 10^7$
	$\frac{1}{128}$	$5.2 \cdot 10^8$	$1.1 \cdot 10^8$

III.2. Classical Iterative Solvers

The setting of this and the following section is as follows: We want to solve a linear system of equations

$$Lu = f$$

with N unknowns and a symmetric positive definite matrix L. We denote by κ the *condition* of L, i.e. the ratio of its largest to its smallest eigenvalue. Moreover we assume that $\kappa \approx N^{\frac{2}{d}}$.

III.2.1. Stationary iterative solvers. All methods of this section are so-called *stationary iterative solvers* and have the following structure:

Algorithm III.2.1. (Stationary iterative solver)

(0) *Given: a matrix L, a right-hand side f, an initial guess u_0 and a tolerance ε.*
Sought: an approximate solution of the linear system of equations $Lu = f$.

(1) *Set* $i = 0$.
(2) *If*

$$\|Lu_i - f\| < \varepsilon,$$

return u_i *as approximate solution;* stop.
(3) *Compute*

$$u_{i+1} = F(u_i; L, f)$$

increase i *by* 1 *and return to step* (2).

Here, $u \mapsto F(u; L, f)$ is an affine mapping, the so-called *iteration method*, which characterizes the particular iterative solver. $\|\cdot\|$ is any norm on $\mathbb{R}^N$, e.g., the Euclidean norm.

III.2.2. Richardson iteration. The simplest stationary iteration method is the *Richardson iteration.* The iteration method is given by

$$\boxed{u \mapsto u + \frac{1}{\omega}(f - Lu).}$$

Here, ω is a *damping parameter*, which has to be of the same order as the largest eigenvalue of L. The convergence rate of the Richardson iteration is $\frac{\kappa-1}{\kappa+1} \approx 1 - N^{-\frac{2}{d}}$.

III.2.3. Jacobi iteration. The *Jacobi iteration* is closely related to the Richardson iteration. The iteration method is given by

$$\boxed{u \mapsto u + D^{-1}(f - Lu).}$$

Here, D is the diagonal of L. The convergence rate again is $\frac{\kappa-1}{\kappa+1} \approx 1 - N^{-\frac{2}{d}}$. Notice, the Jacobi iteration sweeps through all equations and exactly solves the current equation for the corresponding unknown without modifying subsequent equations.

III.2.4. Gauß-Seidel iteration. The *Gauß-Seidel iteration* is a modification of the Jacobi iteration: Now every update of an unknown is immediately transferred to all subsequent equations. This modification gives rise to the following iteration method:

$$\boxed{u \mapsto u + \mathcal{L}^{-1}(f - Lu).}$$

Here, $\mathcal{L}$ is the lower diagonal part of L, diagonal included. The convergence rate again is $\frac{\kappa-1}{\kappa+1} \approx 1 - N^{-\frac{2}{d}}$.

III.2.5. SSOR iteration. The *SSOR iteration* is a modification of the Gauß-Seidel iteration based on the following ideas:

- Sweep through the equations first in increasing order, then in decreasing order.
- Solve the i-th equation for the i-th unknown and write the result in the form “old value plus increment”.
- The new approximation for the i-th unknown then is the old one plus a factor, usually 1.5, times the increment.
- Immediately insert the new value of the i-th unknown in all subsequent equations.

III.2.6. Comparison. Figure III.2.1 shows the evolution of the convergence rate in the course of the iteration process for the Richardson, Jacobi, Gauß-Seidel and SSOR iterations for the model problem of section III.1.1 with mesh-size $h = \frac{1}{64}$. The slow-down of the convergence with increasing number of iterations is typical. The mean convergence rates for the four methods are 0.992, 0.837, 0.752 and 0.513. Thus the convergence rate of the SSOR iteration is acceptable for this simple example. Nevertheless, when further reducing the mesh-size h, the convergence rate of the SSOR iteration will also approach 1.

III.3. Conjugate Gradient Algorithm

III.3.1. The gradient algorithm. This algorithm is based on the following ideas:

- For symmetric positive definite stiffness matrices L the solution of the linear system of equations
$$Lu = f$$
is equivalent to the minimization of the quadratic functional
$$J(u) = \frac{1}{2} u \cdot (Lu) - f \cdot u.$$
- Given an approximation v to the solution u of the linear system, the negative gradient
$$-\nabla J(v) = f - Lv$$
of J at v gives the direction of the steepest descent.
- Given an approximation v and a search direction $d \neq 0$, J attains its minimum on the line $t \mapsto v + td$ at the point
$$t^* = \frac{d \cdot (f - Lv)}{d \cdot (Ld)}.$$

These lead to the following algorithm:

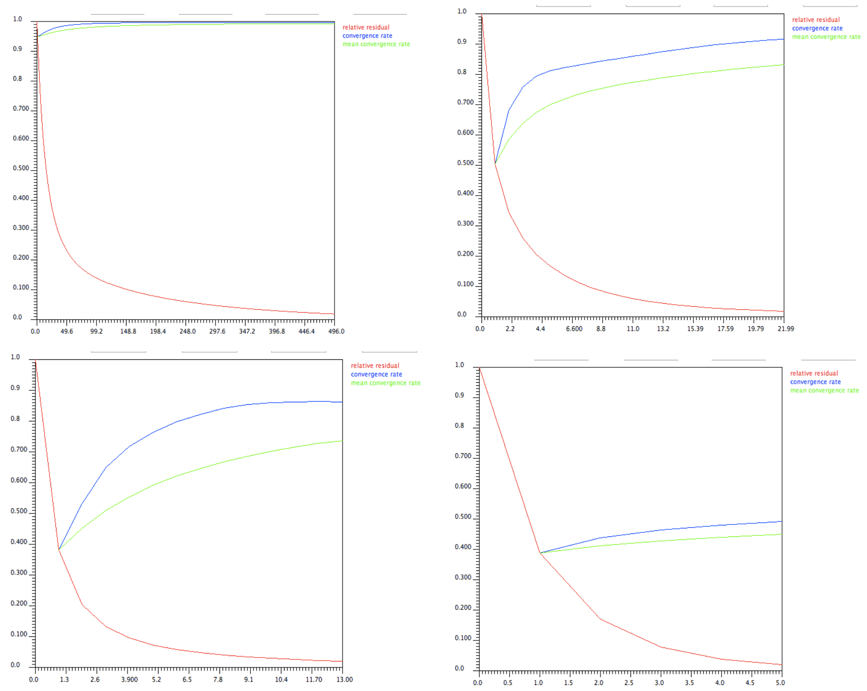

FIGURE III.2.1. Evolution of the convergence rate in the course of the iteration process for the Richardson, Jacobi, Gauß-Seidel and SSOR iterations (left to right, top to bottom) for the model problem of section III.1.1 with mesh-size $h = \frac{1}{64}$

ALGORITHM III.3.1. (Gradient algorithm)

(0) *Given: a linear system of equations $Lu = f$ with a symmetric, positive definite matrix L, an initial guess u_0 for the solution, and a tolerance $\varepsilon > 0$.*
Sought: an approximate solution of the linear system.

(1) *Compute*

$$r_0 = f - Lu_0, \quad \gamma_0 = r_0 \cdot r_0.$$

Set $i = 0$.

(2) *If $\gamma_i < \varepsilon^2$ return u_i as approximate solution;* stop. *Otherwise go to step 3.*

(3) *Compute*

$$\begin{aligned} s_i &= Ld_i, & \alpha_i &= \frac{\gamma_i}{d_i \cdot s_i}, \\ u_{i+1} &= u_i + \alpha_i d_i, & r_{i+1} &= r_i - \alpha_i s_i, \\ \gamma_{i+1} &= r_{i+1} \cdot r_{i+1}. \end{aligned}$$

Increase i by 1 and go to step 2.

The gradient algorithm corresponds to a Richardson iteration with an automatic and optimal choice of the relaxation parameter. The convergence rate, however, is the same as for the Richardson iteration and equals $\frac{\kappa-1}{\kappa+1} \approx 1 - N^{-\frac{2}{d}}$. Figure III.3.1 shows the evolution of the convergence rate in the course of the iteration process for the Richardson iteration and the gradient algorithm for the model problem of section III.1.1 with mesh-size $h = \frac{1}{64}$. The mean convergence rates are 0.992 for the Richardson iteration and 0.775 for the gradient algorithm. When the mesh-size is reduced the convergence rate of the gradient algorithm will further approach the rate of the Richardson iteration.

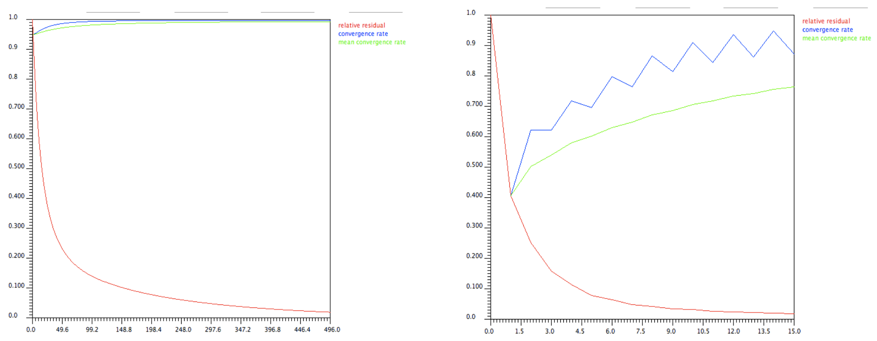

FIGURE III.3.1. Evolution of the convergence rate in the course of the iteration process for the Richardson iteration (left) and the gradient algorithm (right) for the model problem of section III.1.1 with mesh-size $h = \frac{1}{64}$

III.3.2. The conjugate gradient algorithm. The conjugate gradient algorithm is a modification of the gradient algorithm and is inspired by the following observations:

- The gradient algorithm slows down since the search directions become nearly parallel.
- The algorithm speeds up when choosing the successive search directions L-orthogonal, i.e.

$$d_i \cdot (Ld_{i-1}) = 0$$

  for the search directions of iterations $i-1$ and i.
- These L-orthogonal search directions can be computed during the algorithm by a suitable three-term recursion.

ALGORITHM III.3.2. (Conjugate gradient algorithm)

(0) *Given: a linear system of equations* $Lu = f$ *with a symmetric, positive definite matrix* L*, an initial guess* u_0 *for the solution, and a tolerance* $\varepsilon > 0$.
Sought: an approximate solution of the linear system.

(1) *Compute*

$$r_0 = f - Lu_0, \quad d_0 = r_0, \quad \gamma_0 = r_0 \cdot r_0.$$

Set $i = 0$.

(2) *If*

$$\gamma_i < \varepsilon^2$$

return u_i *as approximate solution;* stop. *Otherwise go to step 3.*

(3) *Compute*

$$\begin{aligned} s_i &= Ld_i, & \alpha_i &= \frac{\gamma_i}{d_i \cdot s_i}, \\ u_{i+1} &= u_i + \alpha_i d_i, & r_{i+1} &= r_i - \alpha_i s_i, \\ \gamma_{i+1} &= r_{i+1} \cdot r_{i+1}, & \beta_i &= \frac{\gamma_{i+1}}{\gamma_i}, \\ d_{i+1} &= r_{i+1} + \beta_i d_i. \end{aligned}$$

Increase i *by* 1 *and go to step 2.*

The convergence rate of the CG-algorithm equals $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \approx 1 - N^{-\frac{1}{d}}$. Notice:

> The CG-algorithm can only be applied to symmetric positive definite matrices, it breaks down for non-symmetric or indefinite matrices.

EXAMPLE III.3.3. Consider the linear system

$$\begin{pmatrix} 1 & -3 \\ -3 & 8 \end{pmatrix} x = \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

Since $\det A = -1$ the matrix A is not positive definite. If we nevertheless apply the conjugate gradient algorithm with starting value $x_0 = 0$ we obtain

$$r_0 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \quad \text{and} \quad (r_0, Ar_0) = 0.$$

Hence, the conjugate gradient algorithm breaks down.

III.3.3. The preconditioned conjugate gradient algorithm. The preconditioned conjugate gradient algorithm is based on the following ideas:

- Instead of the original system
$$Lu = f$$
solve the equivalent system
$$\widehat{L}\widehat{u} = \widehat{f}$$
with
$$\begin{aligned}\widehat{L} &= H^{-1}LH^{-t}\\ \widehat{f} &= H^{-1}f\\ \widehat{u} &= H^t u\end{aligned}$$
and an invertible square matrix H.
- Choose the matrix H such that:
  - The condition number of $\widehat{L}$ is much smaller than the one of L.
  - Systems of the form
$$Cv = d$$
with
$$C = HH^t$$
are much easier to solve than the original system $Lu = f$.
- Apply the conjugate gradient algorithm to the new system $\widehat{L}\widehat{u} = \widehat{f}$ and express everything in terms of the original quantities L, f, and u.

ALGORITHM III.3.4. (Preconditioned conjugate gradient algorithm)

(0) *Given: a linear system of equations $Lu = f$ with a symmetric, positive definite matrix L, an approximation C to L, an initial guess u_0 for the solution, and a tolerance $\varepsilon > 0$.*
Sought: an approximate solution of the linear system.

(1) *Compute*
$$r_0 = f - Lu_0,$$
solve
$$Cz_0 = r_0,$$
and compute
$$d_0 = z_0, \quad \gamma_0 = r_0 \cdot z_0.$$
Set $i = 0$.

(2) *If $\gamma_i < \varepsilon^2$ return u_i as approximate solution;* stop. *Otherwise go to step 3.*

(3) *Compute*

$$s_i = L d_i, \qquad \alpha_i = \frac{\gamma_i}{d_i \cdot s_i},$$
$$u_{i+1} = u_i + \alpha_i d_i, \qquad r_{i+1} = r_i - \alpha_i s_i,$$

solve

$$C z_{i+1} = r_{i+1},$$

and compute

$$\gamma_{i+1} = r_{i+1} \cdot z_{i+1}, \quad \beta_i = \frac{\gamma_{i+1}}{\gamma_i}, \quad d_{i+1} = z_{i+1} + \beta_i d_i.$$

Increase i *by* 1 *and go to step 2.*

For the trivial choice $C = I$, the identity matrix, algorithm III.3.4 reduces to the conjugate gradient algorithm III.3.2. For the non-realistic choice $C = A$, algorithm III.3.4 stops after one iteration and produces the exact solution.

The convergence rate of the PCG-algorithm equals $\frac{\sqrt{\widehat{\kappa}}-1}{\sqrt{\widehat{\kappa}}+1}$, where $\widehat{\kappa}$ is the condition number of $\widehat{L}$ and equals the ratio of the largest to the smallest eigenvalue of $\widehat{L}$.

III.3.4. SSOR-preconditioning. Obviously the efficiency of the PCG-algorithm hinges on a good choice of the preconditioning matrix C. It has to satisfy the contradictory goals that $\widehat{L}$ should have a small condition number and that problems of the form $Cz = d$ should be easy to solve. A good compromise is the SSOR-preconditioner. It corresponds to

$$C = \frac{1}{\omega(2-\omega)}(D - \omega U^t) D^{-1} (D - \omega U)$$

where D and U denote the diagonal of L and its strictly upper diagonal part, respectively and where $\omega \in (0,2)$ is a relaxation parameter.

The following algorithm realizes the SSOR-preconditioning.

Algorithm III.3.5. (SSOR-preconditioning)

(0) *Given:* r *and a relaxation parameter* $\omega \in (0,2)$.
Sought: $z = C^{-1} r$.

(1) *Set*

$$z = 0.$$

(2) *For* $i = 1, \ldots, N$ *compute*

$$z_i = z_i + \omega L_{ii}^{-1} \Big\{ r_i - \sum_{j=1}^{N} L_{ij} z_j \Big\}.$$

(3) *For* $i = N, \ldots, 1$ *compute*

$$z_i = z_i + \omega L_{ii}^{-1} \Big\{ r_i - \sum_{j=1}^{N} L_{ij} z_j \Big\}.$$

The SSOR-preconditioning yields $\widehat{\kappa} \approx N^{\frac{1}{d}}$. Hence, the PCG algorithm with SSOR-preconditioning has a convergence rate $\approx 1 - N^{-\frac{1}{2d}}$.

III.3.5. Comparison. Figure III.3.2 shows the evolution of the convergence rate in the course of the iteration process for the CG and SSOR-PCG algorithm for the model problem of section III.1.1 with mesh-sizes $h = \frac{1}{64}$ and $h = \frac{1}{128}$. The mean convergence rates for the two methods are 0.712 and 0.376 for $h = \frac{1}{64}$ and 0.723 and 0.377 for $h = \frac{1}{128}$. They underline that the SSOR-PCG-algorithm is an efficient solver for problems of medium size.

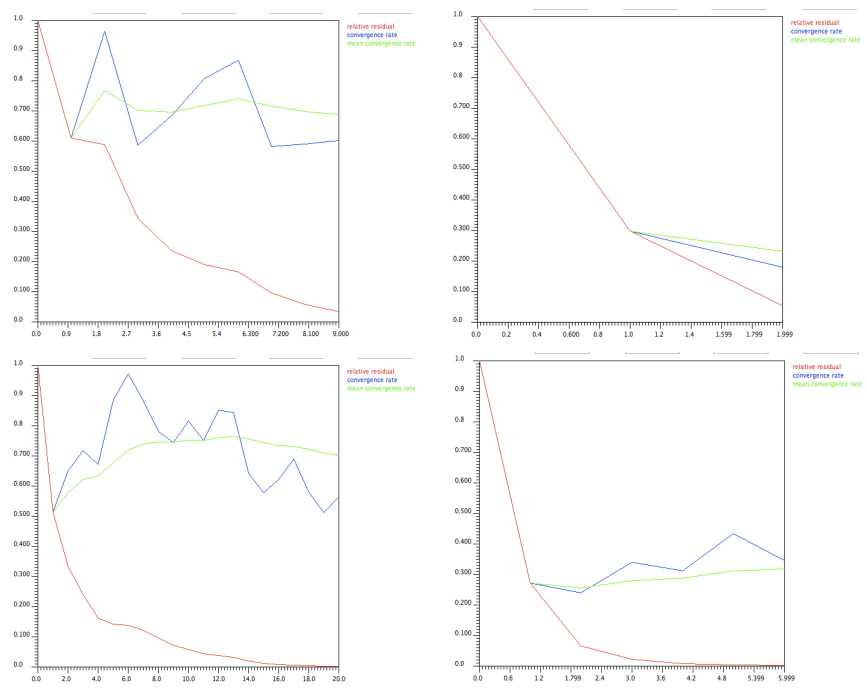

FIGURE III.3.2. Evolution of the convergence rate in the course of the iteration process for the CG (left) and SSOR-PCG (right) algorithm for the model problem of section III.1.1 with mesh-sizes $h = \frac{1}{64}$ (top) and $h = \frac{1}{128}$ (bottom)

III.4. Multigrid Algorithm

III.4.1. The basic idea. The multigrid algorithm is based on the following observations:

- Classical iterative methods such as the Gauß-Seidel algorithm quickly reduce highly oscillatory error components.
- Classical iterative methods such as the Gauß-Seidel algorithm on the other hand are very poor in reducing slowly oscillatory error components.

- Slowly oscillating error components can well be resolved on coarser meshes with fewer unknowns.

III.4.2. The multigrid algorithm. The multigrid algorithm is based on a sequence of meshes $\mathcal{T}_0, \ldots, \mathcal{T}_R$, which are obtained by successive local or global refinement, and associated discrete problems $L_k u_k = f_k$, $k = 0, \ldots, R$, corresponding to a partial differential equation. The finest mesh $\mathcal{T}_R$ corresponds to the problem that we actually want to solve.

The multigrid algorithm has three ingredients:

- a *smoothing operator* M_k, which should be easy to evaluate and which at the same time should give a reasonable approximation to L_k^{-1},
- a *restriction operator* $R_{k,k-1}$, which maps functions on a fine mesh $\mathcal{T}_k$ to the next coarser mesh $\mathcal{T}_{k-1}$,
- a *prolongation operator* $I_{k-1,k}$, which maps functions from a coarse mesh $\mathcal{T}_{k-1}$ to the next finer mesh $\mathcal{T}_k$.

For a concrete multigrid algorithm these ingredients must be specified. This will be done in the next sections. Here, we discuss the general form of the algorithm and its properties.

ALGORITHM III.4.1. (*MG* $(k, \mu, \nu_1, \nu_2, L_k, f_k, u_k)$ one iteration of the multigrid algorithm on mesh $\mathcal{T}_k$)

(0) *Given: the level number k of the actual mesh, parameters μ, ν_1, and ν_2, the stiffness matrix L_k of the actual discrete problem, the actual right-hand side f_k, and an initial guess u_k. Sought: an improved approximate solution u_k.*

(1) *If $k = 0$, compute*

$$u_0 = L_0^{-1} f_0;$$

stop. Otherwise go to step 2.

(2) (Pre-smoothing) *Perform ν_1 steps of the iterative procedure*

$$u_k \mapsto u_k + M_k(f_k - L_k u_k).$$

(3) (Coarse grid correction)

  (a) *Compute*

$$f_{k-1} = R_{k,k-1}(f_k - L_k u_k)$$

  and set

$$u_{k-1} = 0.$$

  (b) *Perform μ iterations of $MG(k-1, \mu, \nu_1, \nu_2, L_{k-1}, f_{k-1}, u_{k-1})$ and denote the result by u_{k-1}.*

  (c) *Update*

$$u_k \mapsto u_k + I_{k-1,k} u_{k-1}.$$

(4) (Post-smoothing) *Perform ν_2 steps of the iterative procedure*

$$u_k \mapsto u_k + M_k(f_k - L_k u_k).$$

The following `Java` method implements the multigrid method with $\mu = 1$. Note that the recursion has been resolved.

```
// one MG cycle
// non-recursive version
    public void mgcycle( int lv, SMatrix a ) {
        int level = lv;
        count[level] = 1;
        while( count[level] > 0 ) {
            while( level > 1 ) {
                smooth( a, 1 );               // presmoothing
                a.mgResidue( res, x, bb );    // compute residue
                restrict( a );                // restrict
                level--;
                count[level] = cycle;
            }
            smooth( a, 0 );                   // solve on coarsest grid
            count[level]--;
            boolean prolongate = true;
            while( level < lv && prolongate ) {
                level++;
                prolongateAndAdd( a );        // prolongate to finer grid
                smooth( a, -1 );              // postsmoothing
                count[level]--;
                if( count[level] > 0 )
                    prolongate = false;
            }
        }
    } // end of mgcycle
```

The used variables and methods have the following meaning:

- `lv` number of actual level, corresponds to k,
- `a` stiffness matrix on actual level, corresponds to L_k,
- `x` actual approximate solution, corresponds to u_k,
- `bb` actual right-hand side, corresponds to f_k,
- `res` actual residual, corresponds to $f_k - L_k u_k$,
- `smooth` perform the smoothing,
- `mgResidue` compute the residual,
- `restrict` perform the restriction,
- `prolongateAndAdd` compute the prolongation and add the result to the current approximation.

III.4.3. Computational cost. The parameter μ determines the complexity of the algorithm. Popular choices are $\mu = 1$ called *V-cycle* and $\mu = 2$ called *W-cycle*. Figure III.4.1 gives a schematic presentation of the multigrid algorithm for the case $\mu = 1$ and $R = 2$ (three meshes). Here, $\mathcal{S}$ denotes smoothing, $\mathcal{R}$ restriction, $\mathcal{P}$ prolongation, and $\mathcal{E}$ exact solution.

The number of smoothing steps per multigrid iteration, i.e. the parameters ν_1 and ν_2, should not be chosen too large. A good choice for

$$\begin{tikzcd}
{} \arrow[r, "\mathcal{S}"] & {} \arrow[d, "\mathcal{R}"'] & & & {} \arrow[r, "\mathcal{S}"] & {} \\
& {} \arrow[r, "\mathcal{S}"] & {} \arrow[d, "\mathcal{R}"'] & {} \arrow[r, "\mathcal{S}"] & {} \arrow[u, "\mathcal{P}"'] & \\
& & {} \arrow[r, "\mathcal{E}"] & {} \arrow[u, "\mathcal{P}"'] & &
\end{tikzcd}$$

FIGURE III.4.1. Schematic presentation of a multigrid algorithm with V-cycle and three grids. The labels have the following meaning: $\mathcal{S}$ smoothing, $\mathcal{R}$ restriction, $\mathcal{P}$ prolongation, $\mathcal{E}$ exact solution.

positive definite problems such as the Poisson equation is $\nu_1 = \nu_2 = 1$. For indefinite problems such as mixed formulations of the equations of linearized elasticity, a good choice is $\nu_1 = \nu_2 = 2$.

If $\mu \leq 2$, one can prove that the computational work of one multigrid iteration is proportional to the number of unknowns of the actual discrete problem.

III.4.4. Convergence rate. Under suitable conditions on the smoothing algorithm, which is determined by the matrix M_k, one can prove that the convergence rate of the multigrid algorithm is independent of the mesh-size, i.e., it does not deteriorate when refining the mesh. In practice one observes convergence rates of about 0.1 for positive definite problems such as the Poisson equation and of about 0.3 for indefinite problems such as mixed formulations of the equations of linearized elasticity.

III.4.5. Smoothing. The symmetric *Gauss-Seidel algorithm* of section III.2.4 is the most popular smoothing algorithm for positive definite problems such as the Poisson equation. It corresponds to the choice

$$M_k = (D_k - U_k^t) D_k^{-1} (D_k - U_k),$$

where D_k and U_k denote the diagonal and the strictly upper diagonal part of L_k respectively.

III.4.6. Prolongation. Since the partition $\mathcal{T}_k$ of level k always is a refinement of the partition $\mathcal{T}_{k-1}$ of level $k-1$, the corresponding finite element spaces are nested, i.e., finite element functions corresponding to level $k-1$ are contained in the finite element space corresponding to level k. Therefore, the values of a coarse-grid function corresponding to level $k-1$ at the nodal points corresponding to level k are obtained by evaluating the nodal basis functions corresponding to $\mathcal{T}_{k-1}$ at the requested points. This defines the interpolation operator $I_{k-1,k}$.

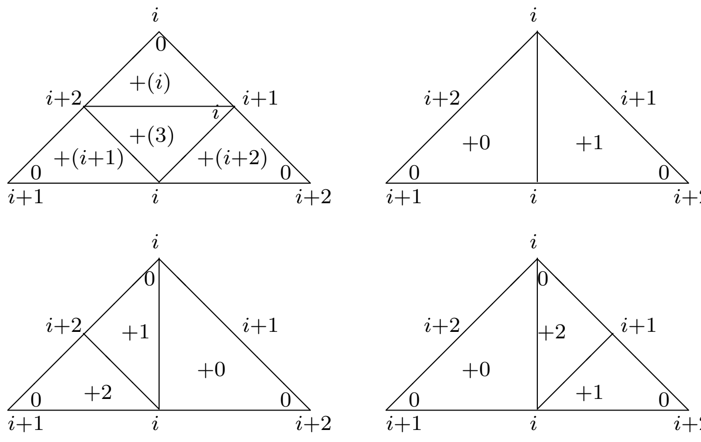

Figure III.4.2. Partitions of a triangle; expressions of the form $i + 1$ have to be taken modulo 3

Figures III.4.2 and III.4.3 show various partitions of a triangle and of a square, respectively. The numbers outside the element indicate the enumeration of the element vertices and edges. Thus, e.g., edge 2 of the triangle has the vertices 0 and 1 as its endpoints. The numbers +0, +1 etc. inside the elements indicate the enumeration of the children. The remaining numbers inside the elements give the enumeration of the vertices of the children.

Example III.4.2. Consider a piecewise constant approximation, i.e. $S^{0,-1}(\mathcal{T})$. The nodal points are the barycentres of the elements. Every element in $\mathcal{T}_{k-1}$ is subdivided into several smaller elements in $\mathcal{T}_k$. The nodal value of a coarse-grid function at the barycentre of a children in $\mathcal{T}_k$ then is its nodal value at the barycentre of the parent element in $\mathcal{T}_k$.

Example III.4.3. Consider a piecewise linear approximation, i.e. $S^{1,0}(\mathcal{T})$. The nodal points are the vertices of the elements. The refinement introduces new vertices at the mid-points of some edges of the parent element and possibly – when using quadrilaterals – at the barycentre of the parent element. The nodal value at the mid-point of an edge is the average of the nodal values at the endpoints of the edge. Thus, e.g., the value at vertex 1 of child +0 is the average of the values at vertices 0 and 1 of the parent element. Similarly, the nodal value at the barycentre of the parent element is the average of the nodal values at the four element vertices.

III.4.7. Restriction. The restriction is computed by expressing the nodal basis functions corresponding to the coarse partition $\mathcal{T}_{k-1}$ in terms of the nodal basis functions corresponding to the fine partition $\mathcal{T}_k$ and inserting this expression in the variational formulation. This

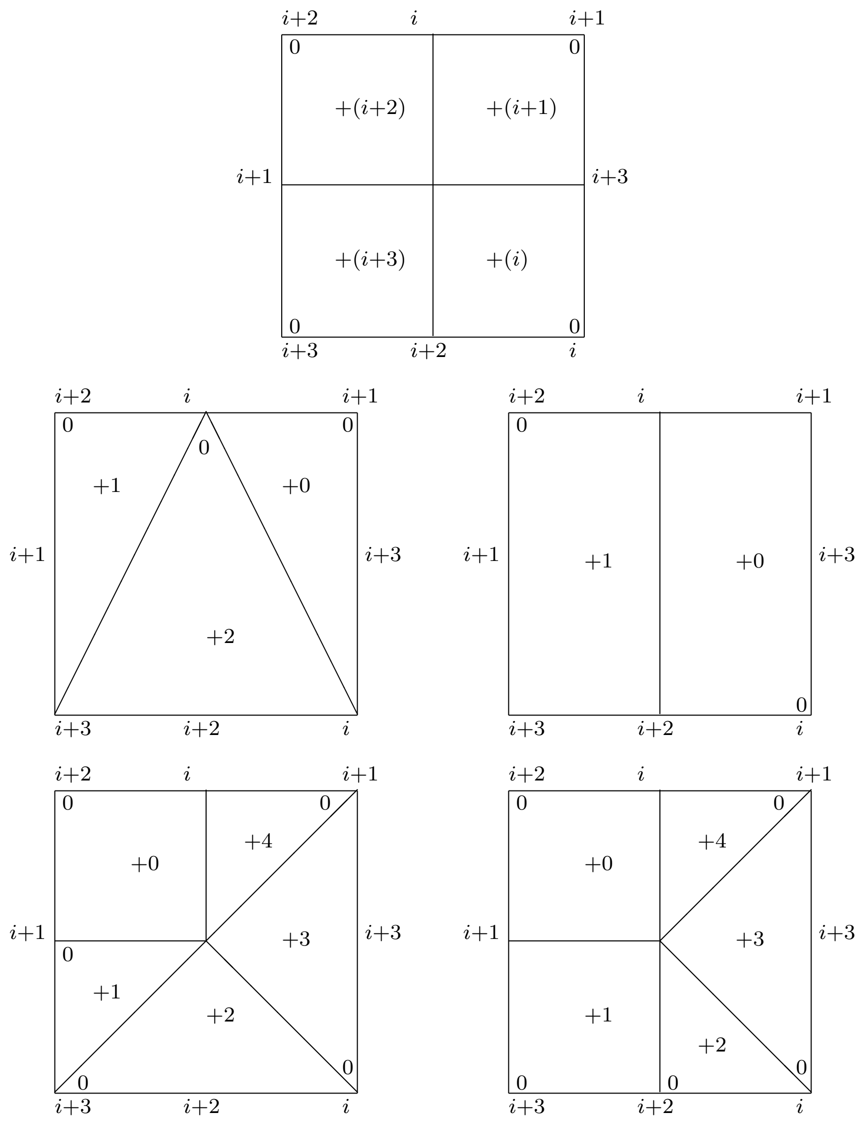

FIGURE III.4.3. Partitions of a square; expressions of the form $i + 1$ have to be taken modulo 4

results in a lumping of the right-hand side vector which, in a certain sense, is the transpose of the interpolation.

EXAMPLE III.4.4. Consider a piecewise constant approximation, i.e. $S^{0,-1}(\mathcal{T})$. The nodal shape function of a parent element is the sum of the nodal shape functions of the children. Correspondingly, the components of the right-hind side vector corresponding to the children are all added and associated with the parent element.

EXAMPLE III.4.5. Consider a piecewise linear approximation, i.e. $S^{1,0}(\mathcal{T})$. The nodal shape function corresponding to a vertex of a parent *triangle* takes the value 1 at this vertex, the value $\frac{1}{2}$ at the mid-points of the two edges sharing the given vertex and the value 0 on the remaining edges. If we label the current vertex by a and the mid-points of the two edges emanating form a by m_1 and m_2, this results in the following formula for the restriction on a *triangle*

$$R_{k,k-1}\psi(a) = \psi(a) + \frac{1}{2}\{\psi(m_1) + \psi(m_2)\}.$$

When considering a *quadrilateral*, we must take into account that the nodal shape functions take the value $\frac{1}{4}$ at the barycentre b of the parent quadrilateral. Therefore the restriction on a *quadrilateral* is given by the formula

$$R_{k,k-1}\psi(a) = \psi(a) + \frac{1}{2}\{\psi(m_1) + \psi(m_2)\} + \frac{1}{4}\psi(b).$$

An efficient implementation of the prolongation and restrictions loops through all elements and performs the prolongation or restriction element-wise. This process is similar to the usual element-wise assembly of the stiffness matrix and the load vector.

III.5. Indefinite Problems

III.5.1. Conjugate gradient type algorithms. The CG- and the PCG-algorithms III.3.2 and III.3.4 can only be applied to problems with a symmetric positive definite stiffness matrix, i.e., to scalar linear elliptic equations without convection and the displacement formulation of the equations of linearized elasticity. Scalar linear elliptic equations with convection – though possibly being small – and mixed formulations of the equations of linearized elasticity lead to non-symmetric or indefinite stiffness matrices. For these problems algorithms III.3.2 and III.3.4 break down.

There are several possible remedies to this difficulty. An obvious one is to consider the equivalent normal equations

$$L^t Lu = L^t f$$

which have a symmetric positive matrix. This simple device, however, cannot be recommended, since passing to the normal equations squares the condition number and thus doubles the number of iterations. A much better alternative is the bi-conjugate gradient algorithm. It tries to solve simultaneously the original problem $Lu = f$ and its adjoint or conjugate problem $L^t v = L^t f$.

Algorithm III.5.1. (Stabilized bi-conjugate gradient algorithm Bi-CG-stab)

(0) *Given: a linear system of equations $Lx = b$, an initial guess x_0 for the solution, and a tolerance ε.*
Sought: an approximate solution of the linear system.

(1) *Compute*

$$r_0 = b - Lx_0,$$

and set

$$\overline{r}_0 = r_0, \qquad v_{-1} = 0, \qquad p_{-1} = 0\ ,$$
$$\alpha_{-1} = 1, \qquad \rho_{-1} = 1, \qquad \omega_{-1} = 1\ .$$

Set $i = 0$.

(2) *If $r_i \cdot r_i < \varepsilon^2$ return x_i as approximate solution;* stop. *Otherwise go to step 3.*

(3) *Compute*

$$\rho_i = \overline{r}_i \cdot r_i, \quad \beta_{i-1} = \frac{\rho_i \alpha_{i-1}}{\rho_{i-1}\omega_{i-1}}.$$

If

$$|\beta_{i-1}| < \varepsilon$$

there is a possible break-down; stop. *Otherwise compute*

$$p_i = r_i + \beta_{i-1}\{p_{i-1} - \omega_{i-1}v_{i-1}\}, \qquad v_i = Lp_i,$$
$$\alpha_i = \frac{\rho_i}{\overline{r}_0 \cdot v_i}.$$

If

$$|\alpha_i| < \varepsilon$$

there is a possible break-down; stop. *Otherwise compute*

$$s_i = r_i - \alpha_i v_i, \qquad t_i = Ls_i,$$
$$\omega_i = \frac{t_i \cdot s_i}{t_i \cdot t_i}, \qquad x_{i+1} = x_i + \alpha_i p_i + \omega_i s_i,$$
$$r_{i+1} = s_i - \omega_i t_i.$$

Increase i by 1 and go to step 2.

III.5.2. Multigrid algorithm. The multigrid algorithm III.4.1 can directly be applied to non-symmetric and indefinite problems. One only has to take special care for an appropriate smoothing method.

For non-symmetric or indefinite problems such as scalar linear elliptic equations with convection or mixed formulations of the equations of linearized elasticity, the most popular smoothing algorithm is the squared Jacobi iteration. This is the Jacobi iteration applied to the squared system $L_k^t L_k u_k = L_k^t f_k$ and corresponds to the choice

$$M_k = \omega^{-2} L_k^t$$

with a suitable damping parameter satisfying $\omega > 0$ and $\omega = O(h_k^{-2})$.

Alternatively, one may use an incomplete LU-decompostion $\mathcal{LU} \approx \frac{1}{2}(L_k + L_k^t)$ of the symmetric part of L_k. Here, incomplete means that every fill-in is suppressed in the standard LU-decomposition. This approach corresponds to the choice $M_k = \mathcal{LU}$.

CHAPTER IV

Linear and Non-Linear Optimization Problems

IV.1. Linear Optimization Problems

IV.1.1. Motivation. Optimization problems play a crucial role in science and engineering. They consist in finding a minimum or maximum of an objective function subject to constraints. Since the maximum of a given function is the negative of the minimum of the negative function, it suffices to consider minimization problems. Usually both the objective function and the constraints are non-linear functions. These non-linear problems are solved by some Newton-type process which requires the solution of auxiliary linear problems with linear objective and constraints. Thus the solution of linear optimization problems is a fundamental tool. Linear optimization problems, however, are of their own interest since many practical problems already have this simpler form.

We start with a simple example.

Example IV.1.1. A small company produces two models of shoes. The net profit is 16 \$ and 32 \$, resp. per shoe. The required material is 6dm^2 and 15dm^2, resp. per shoe; there are 4500dm^2 available per month. The required machine-time is 4h and 5h, resp. per shoe; the available total time is 2000h per month. The required man-time is 20h and 10h, resp. per shoe; the available total time is 8000h per month. The company wants to maximize its profit, this lead to the optimization problem:

$$\text{maximize the } \textit{objective function} \quad 16x + 32y$$

subject to the *constraints*

$$\begin{gathered} 6x + 15y \le 4500, \quad 4x + 5y \le 2000, \quad 20x + 10y \le 8000, \\ x \ge 0, \qquad\qquad y \ge 0. \end{gathered}$$

Figure IV.1.1 shows the domain of the admissible x and y values in blue and the level lines of the objective function in red. Since level lines farther to the top right corner correspond to larger profits we conclude from figure IV.1.1 that the maximum profit is 10400 \$ and is attained when producing 250 shoes of the first kind and 200 ones of the second kind.

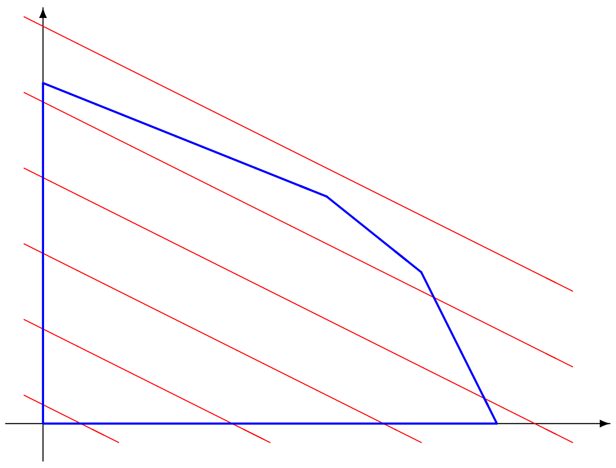

Figure IV.1.1. Domain of admissible x and y values (blue) and level lines of the objective function (red) for example IV.1.1

IV.1.2. General linear optimization problems. We say that two vectors u and v in $\mathbb{R}^k$ satisfy the inequality $u \leq v$ if and only if the inequality $u_i \leq v_i$ holds for *all* components of these vectors.

Given two integers $1 \leq m < n$, a vector $c \in \mathbb{R}^n$, a matrix $A \in \mathbb{R}^{m \times n}$, vectors $\underline{b}, \overline{b} \in [\mathbb{R} \cup \{-\infty, \infty\}]^m$ and vectors $\ell, u \in [\mathbb{R} \cup \{-\infty, \infty\}]^n$ the *general form of a linear optimization problem* is given by:

> Find a *minimum* of the linear function
> $$\mathbb{R}^n \ni x \mapsto c^t x \in \mathbb{R}$$
> subject to the linear *constraints*
> $$\underline{b} \leq Ax \leq \overline{b}, \quad \ell \leq x \leq u.$$

Notice that inequalities of the form $u_i \leq \infty$ or $u_i \geq -\infty$ can be ignored in practice and are only incorporated into the above problem to obtain a convenient short-hand form.

Example IV.1.2. Since the maximum of a given function is the negative of the minimum of the negative function, example IV.1.1 corresponds to the following data:

$$n = 2, \qquad m = 3$$

$$A = \begin{pmatrix} 6 & 15 \\ 4 & 5 \\ 20 & 10 \end{pmatrix},$$

$$\underline{b} = \begin{pmatrix} -\infty \\ -\infty \\ -\infty \end{pmatrix}, \qquad \overline{b} = \begin{pmatrix} 4500 \\ 2000 \\ 8000 \end{pmatrix},$$

$$\ell = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad u = \begin{pmatrix} \infty \\ \infty \end{pmatrix}$$
$$c = \begin{pmatrix} -16 \\ -32 \end{pmatrix}.$$

IV.1.3. Linear optimization problems in standard form. The so-called *standard form of a linear optimization problem* is given by:

$$\boxed{\begin{array}{l} \text{Find a } \textit{minimum} \text{ of the linear function} \\ \qquad\qquad \mathbb{R}^n \ni x \mapsto c^t x \in \mathbb{R} \\ \text{subject to the linear } \textit{constraints} \\ \qquad\qquad Ax = b, \quad x \geq 0. \end{array}}$$

The set

$$\boxed{\mathcal{P} = \{x \in \mathbb{R}^n : Ax = b\ ,\ x \geq 0\}}$$

is called the *set of admissible vectors* or *admissible set* in short associated with the optimization problem.

Example IV.1.3. Example IV.1.1 corresponds to the following data:

$$n = 5, \qquad m = 3,$$
$$A = \begin{pmatrix} 6 & 15 & 1 & 0 & 0 \\ 4 & 5 & 0 & 1 & 0 \\ 20 & 10 & 0 & 0 & 1 \end{pmatrix},$$
$$b = \begin{pmatrix} 4500 \\ 2000 \\ 8000 \end{pmatrix}, \qquad c = \begin{pmatrix} -16 \\ -32 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

When comparing examples IV.1.2 and IV.1.3 we observe that the number of unknowns has increased. This is due to the introduction of *slack variables* as described in section IV.1.5 below.

IV.1.4. Linear optimization problems in simplex form. The so-called *simplex form of a linear optimization problem* is particularly convenient for the *simplex algorithm* described below. It takes the form:

$$\boxed{\begin{array}{l} \text{Find a } \textit{maximum} \text{ of the special linear function} \\ \qquad\qquad \mathbb{R} \ni z \mapsto z \in \mathbb{R} \end{array}}$$

subject to the *constraints*

$$Ax = b, \quad c^t x + z = 0, \quad x \geq 0.$$

Setting

$$\widehat{A} = \begin{pmatrix} A & 0 \\ c^t & 1 \end{pmatrix}, \quad \widehat{x} = \begin{pmatrix} x \\ z \end{pmatrix}, \quad \widehat{b} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

the above constraints can be written in the short-hand form

$$\widehat{A}\widehat{x} = \widehat{b}, \quad x \geq 0.$$

EXAMPLE IV.1.4. Example IV.1.1 corresponds to the data

$$\widehat{A} = \begin{pmatrix} 6 & 15 & 1 & 0 & 0 & 0 \\ 4 & 5 & 0 & 1 & 0 & 0 \\ 20 & 10 & 0 & 0 & 1 & 0 \\ -16 & -32 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \widehat{b} = \begin{pmatrix} 4500 \\ 2000 \\ 8000 \\ 0 \end{pmatrix}.$$

IV.1.5. Equivalence of linear optimization problems. The different forms of optimization problems described above are all equivalent in the sense that each problem can be transformed into one of the other forms by introducing suitable new variables or writing equalities as inequalities. In particular observe that:

- The function $x \mapsto c^t x$ is minimal, if and only if the function $x \mapsto (-c)^t x$ is maximal. Hence, it is sufficient to consider minimization problems.
- The equality $y = b$ is equivalent to the two inequalities $y \leq b$ and $y \geq b$.
- An inequality $y \leq b$ is equivalent to equality $y + z = b$ plus the inequality $z \geq 0$. The vector z is called *slack vector* or *slack variable*.

IV.1.6. Properties of linear optimization problems. Linear optimization problems have the following properties:

The set $\mathcal{P}$ of admissible vectors is a *simplex*.
If the set $\mathcal{P}$ is empty, the optimization problem has no solution.
If the function $x \mapsto c^t x$ is not bounded from below on $\mathcal{P}$, the optimization problem has no solution.
If the set $\mathcal{P}$ is not empty and bounded, the optimization problem admits a solution.
The solution may not be unique.
Every solution is attained at a *vertex* of the set $\mathcal{P}$.

IV.1.7. Basic idea of the simplex algorithm. The *simplex algorithm* is the most commonly used algorithm for solving linear optimization problems. Its basic ideas can be described as follows:

> Given a vertex of $\mathcal{P}$ find a neighbouring vertex with a smaller value for $c^t x$.
> If such a neighbour does not exist, the current vertex solves the optimization problem.
> A vector $x \in \mathbb{R}^n$ is a vertex of $\mathcal{P}$, if it has m non-negative components and $n - m$ vanishing components and solves the system $Ax = b$.
> When freezing $n - m$ components of x to zero, the system $Ax = b$ reduces to a linear system of m equations and m unknowns involving only those columns of A which correspond to the unfrozen components of x.

In order to make these ideas operative we will discuss in the following sections how to solve the following tasks:

- Find a vertex of $\mathcal{P}$.
- Decide whether a given vertex is optimal.
- Find a neighbouring vertex with a smaller value of $c^t x$.

IV.1.8. Finding a vertex of the admissible set. Vertices of the admissible set $\mathcal{P}$ correspond to sets $J = \{j_1, \ldots, j_m\} \subset \{1, \ldots, n\}$ of m indices such that the corresponding columns of the matrix A are linearly independent. Given such a set we proceed as follows:

> Set $\overline{x}_k = 0$ for all $k \notin J$.
> Denote by A_J the $m \times m$ matrix which is obtained by discarding all columns of A corresponding to indices not contained in J.
> Solve the linear system of equations $A_J y = b$.
> Set $\overline{x}_{j_i} = y_i$ for $i = 1, \ldots, m$.

If $\overline{x}_j \geq 0$ for all $j \in J$, $\overline{x}$ is a vertex of $\mathcal{P}$. In this case we set:

$$\boxed{\begin{aligned} &\overline{A} = A_J^{-1} A, & & \overline{b} = A_J^{-1} b \\ &\widetilde{c}_i = c_{j_i} \qquad \text{for } 1 \leq i \leq m, & & \overline{c}^t = -\widetilde{c}^t \overline{A} + c^t, \\ &\beta = -\overline{c}^t \overline{x}. & & \end{aligned}}$$

EXAMPLE IV.1.5. For example IV.1.3 and $J = \{3, 4, 1\}$ we obtain

$$\overline{x} = \begin{pmatrix} 400 \\ 0 \\ 2100 \\ 400 \\ 0 \end{pmatrix}, \qquad \overline{A} = \begin{pmatrix} 0 & 12 & 1 & 0 & -\frac{6}{20} \\ 0 & 3 & 0 & 1 & -\frac{4}{20} \\ 1 & \frac{1}{2} & 0 & 0 & -\frac{1}{20} \end{pmatrix},$$

$$\overline{b} = \begin{pmatrix} 2100 \\ 400 \\ 400 \end{pmatrix}, \qquad \overline{c} = \begin{pmatrix} 0 \\ -24 \\ 0 \\ 0 \\ \frac{16}{20} \end{pmatrix},$$

$$\beta = 6400.$$

IV.1.9. Checking for optimality and solvability. Suppose that $\overline{x}$ is a vertex of the admissible set $\mathcal{P}$. Then we have:

> If $\overline{c}_k \geq 0$ for all $k \notin J$, $\overline{x}$ solves the optimization problem.
> If, for all $s \notin J$ with $\overline{c}_s < 0$, all entries in the corresponding columns of $\overline{A}$ are non-positive, the optimization problem has no solution.

EXAMPLE IV.1.6. We continue with example IV.1.5. Since $\overline{c}_2 = -24 < 0$ and $2 \notin J$, $\overline{x}$ does not solve the optimization problem. Since the entries in the second column of $\overline{A}$ are positive, the optimization problem admits a solution and we must find a better vertex.

IV.1.10. Finding a better neighbour. Suppose that $\overline{x}$ is a vertex of the admissible set $\mathcal{P}$ which is not optimal and which guarantees the solvability of the optimization problem. Then we determine a new vertex with a better value of $c^t x$ as follows:

> Choose an index $s \notin J$ such that $\overline{c}_s < 0$ and such that $\overline{a}$, the s-th column of $\overline{A}$, has a positive entry.
> Find an index $r \in \{1, \ldots, m\}$ such that $\overline{a}_r > 0$ and such that $\frac{\overline{b}_r}{\overline{a}_r}$ is minimal among all fractions $\frac{\overline{b}_j}{\overline{a}_j}$ with positive denominator.
> Remove the r-th entry from the index set J and put s into J.
> Update $\overline{x}$, $\overline{A}$, $\overline{b}$, β and $\overline{c}$.

Notice that the above update can be performed by dividing the r-th row of a suitable matrix by $\overline{a}_r$ and subtracting the result from the other rows of that matrix.

EXAMPLE IV.1.7. We continue with examples IV.1.5 and IV.1.6 and choose $s = 2$. Then we have

$$\frac{\overline{b}_1}{\overline{a}_1} = \frac{2100}{12} = 175, \quad \frac{\overline{b}_2}{\overline{a}_2} = \frac{400}{3}, \quad \frac{\overline{b}_3}{\overline{a}_3} = \frac{400}{\frac{1}{2}} = 800$$

and conclude that $r = 2$. Hence, the new set J is $\{3, 2, 1\}$ and we obtain the updated values

$$\overline{x} = \begin{pmatrix} \frac{1000}{3} \\ \frac{400}{3} \\ 500 \\ 0 \\ 0 \end{pmatrix}, \qquad \overline{A} = \begin{pmatrix} 0 & 0 & 1 & -4 & \frac{1}{2} \\ 0 & 1 & 0 & \frac{1}{3} & -\frac{2}{30} \\ 1 & 0 & 0 & -\frac{1}{6} & \frac{5}{60} \end{pmatrix},$$

$$\overline{b} = \begin{pmatrix} 500 \\ \frac{400}{3} \\ \frac{1000}{3} \end{pmatrix}, \qquad \overline{c} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 8 \\ -\frac{16}{20} \end{pmatrix},$$

$$\beta = 9600.$$

Since $\overline{c}_5 = -\frac{16}{20} < 0$ and $5 \notin J$, $\overline{x}$ is not the solution of the optimization problem. Since the fifth column of $\overline{A}$ contains a positive entry. The optimization problem admits a solution and we must find a better vertex. We now have

$$\frac{\overline{b}_1}{\overline{a}_1} = \frac{500}{\frac{1}{2}} = 1000, \quad \frac{\overline{b}_3}{\overline{a}_3} = \frac{\frac{1000}{3}}{\frac{5}{60}} = 4000.$$

Hence, we have $s = 5$ and $r = 1$ and obtain the new set $J = \{5, 2, 1\}$. We obtain the updated values

$$\overline{x} = \begin{pmatrix} 250 \\ 200 \\ 0 \\ 0 \\ 1000 \end{pmatrix}, \qquad \overline{A} = \begin{pmatrix} 0 & 0 & 2 & -8 & 1 \\ 0 & 1 & \frac{4}{30} & \frac{1}{5} & 0 \\ 1 & 0 & -\frac{1}{6} & \frac{1}{2} & 0 \end{pmatrix},$$

$$\overline{b} = \begin{pmatrix} 1000 \\ 200 \\ 250 \end{pmatrix}, \qquad \overline{c} = \begin{pmatrix} 0 \\ 0 \\ \frac{8}{5} \\ \frac{8}{5} \\ 0 \end{pmatrix},$$

$$\beta = 10400.$$

Since now all values $\overline{c}_s$ with $s \notin J$ are positive, this is the solution of the optimization problem, $\beta = 10400$ is the maximal profit.

IV.1.11. Modifications of the simplex algorithm. The simplex algorithm may run into a *cycle* since different index sets J may lead to the same value of $c^t x$. This cycling can be avoided by introducing a suitable *ordering* of the vectors x.

The first index set J needed to start the simplex algorithm can be determined by applying the simplex algorithm to a suitable auxiliary optimization problem which has unit vectors as vertices.

IV.1.12. Complexity of the simplex algorithm. Every step of the simplex algorithm requires $O((m+1)(n+1-m))$ operations. Since the admissible set has at most $\binom{n}{m}$ vertices, the algorithm stops after at most $\binom{n}{m}$ iterations with a solution or the information that the optimization problem has no solution. Thus, in the worst case the overall complexity of the simplex algorithm is $O(2^{\frac{n}{2}}\left(\frac{n}{2}\right)^2)$ operations. Notice, that this is an *exponential complexity.* Klee and Minty have constructed optimization problems where this pessimistic number of iterations really is attained.

IV.1.13. Dual problems. Every vertex of the admissible set $\mathcal{P}$ yields an upper bound for the function $c^t x$. To obtain a lower bound for $c^t x$ one has to consider the so-called *dual optimization problem*:

Find a *maximum* of the function

$$\mathbb{R}^m \ni y \mapsto b^t y \in \mathbb{R}$$

subject to the *constraint*

$$A^t y \leq c.$$

Example IV.1.8. The dual problem associated with example IV.1.1 consists in finding the maximum of the function

$$4500y_1 + 2000y_2 + 8000y_3$$

subject to the constraints

$$\begin{aligned} 6y_1 + 4y_2 + 20y_3 &\leq -16 \\ 15y_1 + 5y_2 + 10y_3 &\leq -32 \\ y_1 &\leq 0 \\ y_2 &\leq 0 \\ y_3 &\leq 0. \end{aligned}$$

One can prove that the minimal value of $c^t x$ and the maximal value of $b^t y$ are identical. Therefore, every vertex of the admissible set of the dual problem yields a lower bound for $b^t y$ and thus for the original function $c^t x$.

The dual problem can be solved with a variant of the simplex algorithm which works with the original data A, b and c.

IV.1.14. Idea of interior point methods. The simplex algorithm sweeps through the *boundary* of the admissible set $\mathcal{P}$. Interior point methods instead sweep through the *interior* of $\mathcal{P}$ and aim at a simultaneous solution of both the original and the dual optimization problem. To this end both problems are reformulated as a system of algebraic equations to which Newton's method is applied.

Interior point methods yield an approximation to the solutions of both optimization problems with an error ε with a complexity of $O(\sqrt{n}\ln(\frac{n}{\varepsilon}))$ operations. Notice that this is an *algebraic complexity*.

After stopping the interior point method, the obtained approximation for the original optimization problem is projected to a close-by vertex of $\mathcal{P}$ and a few steps of the simplex algorithm then yield the exact solution.

IV.1.15. Basic form of interior point methods. Given a vector x denote by X the diagonal matrix which has the components of x as its diagonal entries.

We consider the optimization problem

$$\text{minimize } c^tx \text{ subject to the constraints } Ax = b,\ x \geq 0$$

and the corresponding dual problem

$$\text{maximize } b^ty \text{ subject to the constraints } A^ty + s = c,\ s \geq 0.$$

Then the vector (x^*, y^*, s^*) solves both problems if and only if

$$\Psi_0(x^*, y^*, s^*) = 0$$

where

$$\Psi_0(x,y,s) = \begin{pmatrix} Ax - b \\ A^ty + s - c \\ Xs \end{pmatrix}$$

This set of non-linear equations is then solved by applying Newton's method.

Example IV.1.9. For example IV.1.1 the function Ψ_0 is defined on $\mathbb{R}^5 \times \mathbb{R}^3 \times \mathbb{R}^5$ and is given by

$$\Psi_0(x,y,s) = \begin{pmatrix} 6x_1+15x_2+x_3-4500 \\ 4x_1+5x_2+x_4-2000 \\ 20x_1+10x_2+x_5-8000 \\ 6y_1+4y_2+20y_3+s_1+16 \\ 15y_1+5y_2+10y_3+s_2+32 \\ y_1+s_3 \\ y_2+s_4 \\ y_3+s_5 \\ x_1s_1 \\ x_2s_2 \\ x_3s_3 \\ x_4s_4 \\ x_5s_5 \end{pmatrix}.$$

IV.1.16. Improved form of interior point methods. The interior point method in its primitive form as described above suffers from the difficulty that the derivative $D\Psi_0(x, y, s)$ becomes nearly singular when (x, y, s) approaches the solution (x^*, y^*, s^*) so that Newton's method tends to fail when approaching (x^*, y^*, s^*). To overcome this difficulty one applies Newton's method to

$$\Psi_\mu(x, y, s) = \begin{pmatrix} Ax - b \\ A^t y + s - c \\ Xs - \mu \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \end{pmatrix}$$

and lets μ tend to 0 in a judicious way.

IV.2. Unconstrained Non-Linear Optimization Problems

IV.2.1. Problem setting. Unconstrained non-linear optimization problems often appear as auxiliary problems in solving constrained non-linear optimization problems (see section IV.3 below). Still, they are also of their own interest.

To describe the general form of an unconstrained non-linear optimization problem, consider a non-empty set D in $\mathbb{R}^n$ and a function $f : D \to \mathbb{R}$. Then we are looking for a *minimizer* of f, i.e. a point $x \in D$ with

$$\boxed{f(x) \le f(y) \quad \text{for all } y \in D.}$$

or in short-hand notation

$$\boxed{\min\{f(x) : x \in D\}.}$$

Ideally, we are looking for a *global minimum* but in most cases we have to be satisfied with a *local minimum* (see figure IV.2.1). The methods of this and the following section usually only yield local minima. Methods for finding global minima will be discussed in section IV.4.

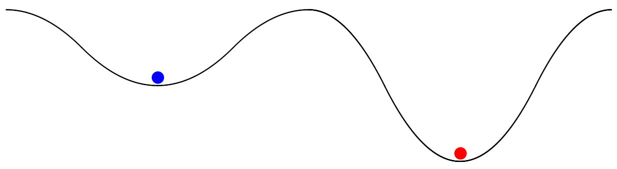
FIGURE IV.2.1. Local (blue) versus global (red) minima

IV.2.2. Optimality conditions. For differentiable functions f there are the following necessary and sufficient *optimality conditions* for local minima:

> If f is differentiable, every local minimum is a *critical point*, i.e. it satisfies $Df(x) = 0$.
> If f is twice differentiable, x is a critical point and the Hessian $D^2 f$ is positive definite, then x is a local minimum.

IV.2.3. Newton's method. The above optimality conditions suggest to apply Newton's method to Df in order to find a local minimum. This idea gives rise to the following algorithm.

ALGORITHM IV.2.1. (Newton's method for finding a local minimum)

(0) *Given: an initial guess x_0 and a tolerance ε.*
Set $n = 0$.

(1) *If*
$$\|Df(x_n)\| \le \varepsilon,$$
go to step (3).

(2) *Solve the linear system*
$$D^2 f(x_n) z_n = -Df(x_n)$$
set
$$x_{n+1} = x_n + z_n,$$
increase n by 1 *and got to step* (1).

(3) *Check whether $D^2 f(x_n)$ is positive definite.*

This naive approach has the following drawbacks:

- Newton's method at best yields a critical point, its result may be a maximum or a saddle-point.
- The algorithm requires second order derivatives.
- Checking the positive definiteness of a matrix is expensive.
- A critical point may be a local minimum although $D^2 f$ is only positive semi-definite, e.g. $f(x) = x^4$.

In what follows we therefore strive at attaining the following goals:

- Develop algorithms which at least find a local minimum.
- Develop algorithms which need as few derivatives as possible.
- Embed Newton's method into a larger class of algorithms to gain more flexibility and insight.
- In view of future applications, develop efficient algorithms for *line search*, i.e. for the minimization of functions of one variable.

IV.2.4. One-dimensional minimization by bisection. Algorithm IV.2.2 below is based on the following ideas:

- Assume that the function $f : [a, b] \to \mathbb{R}$ is continuous and that there is a point $x \in (a, b)$ with
$$f(x) \le \min\{f(a), f(b)\}.$$
- Then f admits a local minimum $\eta \in (a, b)$ and $f'(\eta) = 0$ if f is differentiable (see figure IV.2.2).
- Determine the mid-point u of the smaller one of the two intervals $[a, x]$ and $[x, b]$ and suitably choose three points out of $\{a, x, u, b\}$.

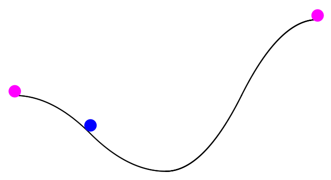

FIGURE IV.2.2. Situation of algorithm IV.2.2

ALGORITHM IV.2.2. (One-dimensional minimization by bisection)

(0) *Given: three points* $a_0 < x_0 < b_0$ *with*
$$f(x_0) \le \min\{f(a_0), f(b_0)\}$$
and a tolerance ε.
Set $k = 0$.

(1) *Compute*
$$u_k = \begin{cases} \frac{1}{2}(b_k + x_k) & \text{if } x_k \le \frac{1}{2}(a_k + b_k), \\ \frac{1}{2}(a_k + x_k) & \text{if } x_k > \frac{1}{2}(a_k + b_k). \end{cases}$$
If
$$f(x_k) \le f(u_k),$$
set
$$x_{k+1} = x_k$$
and
$$a_{k+1} = \begin{cases} a_k & \text{if } x_k \le \frac{1}{2}(a_k + b_k), \\ u_k & \text{if } x_k > \frac{1}{2}(a_k + b_k), \end{cases}$$
$$b_{k+1} = \begin{cases} u_k & \text{if } x_k \le \frac{1}{2}(a_k + b_k), \\ b_k & \text{if } x_k > \frac{1}{2}(a_k + b_k). \end{cases}$$
If
$$f(u_k) < f(x_k),$$

set

$$x_{k+1} = u_k$$

and

$$a_{k+1} = \begin{cases} x_k & \text{if } x_k \le \frac{1}{2}(a_k + b_k), \\ a_k & \text{if } x_k > \frac{1}{2}(a_k + b_k), \end{cases}$$

$$b_{k+1} = \begin{cases} b_k & \text{if } x_k \le \frac{1}{2}(a_k + b_k), \\ x_k & \text{if } x_k > \frac{1}{2}(a_k + b_k). \end{cases}$$

(2) *Increase k by 1. If $b_k - a_k < \varepsilon$* stop. *Otherwise return to step* (1).

Algorithm IV.2.2 has the following properties:

$a_k < x_k < b_k$ for all k.
$f(x_k) \le \min\{f(a_k), f(b_k)\}$ for all k.
$b_k - a_k \le (\frac{3}{4})^{k-1}(b_0 - a_0)$ for all k.
For every prescribed tolerance, the algorithm yields an interval with length less than the tolerance which contains a local minimum of f.
If f is differentiable, the common limit point η of the sequences a_k, b_k and x_k is a critical point of f, i.e. $f'(\eta) = 0$. If f is twice differentiable $f''(\eta) \ge 0$.

IV.2.5. General descent algorithm. We now embed Newton's method into a broader class of algorithms for the minimization of functions.

ALGORITHM IV.2.3. (General descent algorithm)

(0) *Given: parameters $0 < c_1 \le c_2 < 1$, $0 < \gamma \le 1$ and an initial guess $x_0 \in \mathbb{R}^n$.*
Set $k = 0$.

(1) *If*

$$Df(x_k) = 0$$

stop, *otherwise proceed with step* (2).

(2) *Choose a* search direction $s_k \in \mathbb{R}^n$ *with*

$$\|s_k\| = 1 \quad \text{and} \quad -Df(x_k)s_k \ge \gamma\|Df(x_k)\|.$$

(3) *Choose a* step size $\lambda_k > 0$ *such that*

$$f(x_k + \lambda_k s_k) \le f(x_k) + \lambda_k c_1 Df(x_k)s_k$$

and

$$Df(x_k + \lambda_k s_k)s_k \ge c_2 Df(x_k)s_k.$$

(4) *Set*

$$x_{k+1} = x_k + \lambda_k s_k,$$

increase k by 1 and return to step (1).

Algorithm IV.2.3 has the following properties:

> The sequence $f(x_k)$ is monotonically decreasing.
> The sequence x_k admits at least one accumulation point.
> Every accumulation point of the sequence x_k is a critical point of f.

In order to make algorithm IV.2.3 operative we will next present feasible choices of the search direction and of the step size.

IV.2.6. Choice of the search direction. Smaller values of γ give more flexibility in the choice of the search direction. In the limiting case $\gamma \to 0$, the only condition is that the search direction must not be orthogonal to the negative gradient $-Df(x_k)$.

The choice

$$s_k = -\frac{1}{\|Df(x_k)\|} Df(x_k)$$

is feasible for all values of γ and corresponds to Newton's method with damping.

When applied to

$$f(x) = \frac{1}{2}x^t Ax - b^t x$$

with a symmetric positive definite matrix A, the general descent algorithm with a suitable choice of search directions covers the gradient algorithm III.3.1 and (preconditioned) conjugate gradient algorithms III.3.2 and III.3.4.

IV.2.7. Choice of the step size. There are two major possibilities for determining the step size:

- *Exact line search*: The step size λ_k is chosen such that it minimizes the function $t \mapsto f(x_k + ts_k)$ on the positive real line.
- *Armijo line search*: Fix a constant $\sigma > 0$, determine $\lambda_{k,0}^*$ such that
$$\lambda_{k,0}^* \geq \sigma \|Df(x_k)\|$$
and determine the smallest integer j_k satisfying
$$f(x_k + 2^{-j_k}\lambda_{k,0}^* s_k) \leq f(x_k) + 2^{-j_k} c_1 Df(x_k) s_k.$$
Set
$$\lambda_k = 2^{-j_k}\lambda_{k,0}^*$$
or
$$\lambda_k = 2^{-i^*}\lambda_{k,0}^*$$
with
$$f(x_k + 2^{-i^*}\lambda_{k,0}^* s_k) = \min_i f(x_k + 2^{-i}\lambda_{k,0}^* s_k).$$

IV.3. Constrained Non-Linear Optimization Problems

IV.3.1. Convex sets and functions. Convex optimization problems are a particularly important subclass of non-linear optimization problems. Before formulating these problems and corresponding optimality conditions, we recall the definitions of convex sets and of convex and affine functions.

A set $C \subset \mathbb{R}^n$ is called *convex*, if for all $x, y \in C$ and all $\lambda \in [0, 1]$ the point $\lambda x + (1 - \lambda)y$ is contained in C too (see figure IV.3.1).

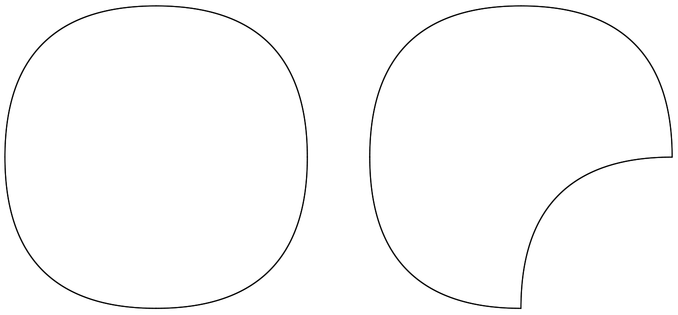

Figure IV.3.1. Examples of convex (left) and non-convex (right) sets

A function $f : C \to \mathbb{R}^n$ on a convex set is called *convex*, if for all $x, y \in C$ and all $\lambda \in [0, 1]$ the inequality

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

is valid (see figure IV.3.2). A function is convex if and only if the set $\{(x, z) \in C \times \mathbb{R} : z \geq f(x)\}$ is convex.

A function $g : C \to \mathbb{R}$ on a convex set $C \subset \mathbb{R}^n$ is called *affine* if it takes the form $g(x) = \mathbf{a} \cdot x + b$ with a given vector $\mathbf{a} \in \mathbb{R}^n$ and a given number $b \in \mathbb{R}$.

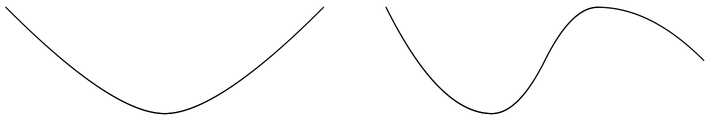

Figure IV.3.2. Examples of convex (left) and non-convex (right) functions

IV.3.2. Convex optimization problems. Suppose that we are given two integers $m \geq 1$ and p with $0 \leq p \leq m$, a convex set $C \subset \mathbb{R}^n$, a convex function $f : C \to \mathbb{R}$, convex functions $f_1, \ldots, f_p : C \to$

$\mathbb{R}$, and affine functions $f_{p+1}, \dots, f_m : C \to \mathbb{R}$. A *convex constrained optimization problem* then takes the form:

> Find a *minimum* of f under the constraints
> $$\begin{aligned} f_i(x) &\le 0 \quad \text{for } 1 \le i \le p, \\ f_j(x) &= 0 \quad \text{for } p+1 \le j \le m. \end{aligned}$$

Notice that the particular cases $p = 0$, no inequality constraints, and $p = m$, no equality constraints, are explicitly admitted.

IV.3.3. Optimality conditions for convex constrained optimization problems. Assume that $C = \mathbb{R}^n$ and that the functions f and $f_1, \dots, f_m$ are differentiable. Then $x^* \in \mathbb{R}^n$ solves the above convex constrained optimization problem, if and only if there is a $y^* \in \mathbb{R}^m$ such that (x^*, y^*) satisfies the following *Karush-Kuhn-Tucker conditions* or *KKT conditions* in short:

$$\boxed{\begin{aligned} Df(x^*) + \sum_{i=1}^{m} y_i^* Df_i(x^*) &= 0, \\ f_i(x^*) y_i^* &= 0, \quad 1 \le i \le p, \\ f_i(x^*) &\le 0, \quad 1 \le i \le p, \\ y_i^* &\ge 0, \quad 1 \le i \le p, \\ f_j(x^*) &= 0, \quad p+1 \le j \le m. \end{aligned}}$$

IV.3.4. Lagrange function. For abbreviation set

$$D = \{ y \in \mathbb{R}^m : y_i \ge 0 \text{ for } 1 \le i \le p \}.$$

Then the function $\mathcal{L} : C \times D \to \mathbb{R}$ with

$$\boxed{\mathcal{L}(x, y) = f(x) + \sum_{j=1}^{m} y_j f_j(x)}$$

is called the *Lagrange function* of the convex constrained optimization problem of section IV.3.2.

With the help of the Lagrange function, the above optimality conditions can be formulated in the following compact form:

> A vector $x^* \in C$ is a solution of the convex constrained optimization problem if and only if there is $y^* \in D$ such

that (x^*, y^*) is a *saddle point* of $\mathcal{L}$, i.e.

$$\mathcal{L}(x, y^*) \geq \mathcal{L}(x^*, y^*) \geq \mathcal{L}(x^*, y)$$

for all $(x, y) \in C \times D$.

IV.3.5. General non-linear optimization problems. Suppose that we are given two integers $m \geq 1$ and p with $0 \leq p \leq m$, a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ and differentiable functions $f_1, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$. A *non-linear constrained optimization problem* then takes the form:

Find a *minimum* of f under the constraints

$$\begin{aligned} f_i(x) &\leq 0 \quad \text{for } 1 \leq i \leq p, \\ f_j(x) &= 0 \quad \text{for } p+1 \leq j \leq m. \end{aligned}$$

Notice that again the particular cases $p = 0$, no inequality constraints, and $p = m$, no equality constraints, are explicitly admitted.

IV.3.6. Tangent cones. Sharp optimality conditions for general non-linear constrained optimization problems are based on the concept of *tangent cones* which generalize the concept of a *tangent space*.

The *tangent cone* $T(S; x)$ of a set $S \subset \mathbb{R}^n$ at a point $x \in S$ is the collection of all vectors $v \in \mathbb{R}^n$ for which there is a sequence λ_k of non-negative real numbers and a sequence x_k of points in S such that $x_k \to x$ and $\lambda_k(x_k - x) \to v$.

Figure IV.3.3 shows two examples of tangent cones. Note that $T(S; x) = \mathbb{R}^n$ if x is an interior point of S and that $T(S; x)$ is the classical tangent space if x is a boundary point of S and if the boundary of S is smooth at x.

IV.3.7. Cone condition. The sharpest optimality condition for a general non-linear constrained optimization problem is the following *cone condition*:

Assume that $x^* \in S$ is a local minimum of f and that f is differentiable at x^*, then $Df(x^*)v \geq 0$ holds for all $v \in T(S; x)$.

The cone condition is of limited practical use since in general the computation of the tangent cone is too expensive. Hence it is replaced by weaker more practical conditions obtained by some sort of linearization.

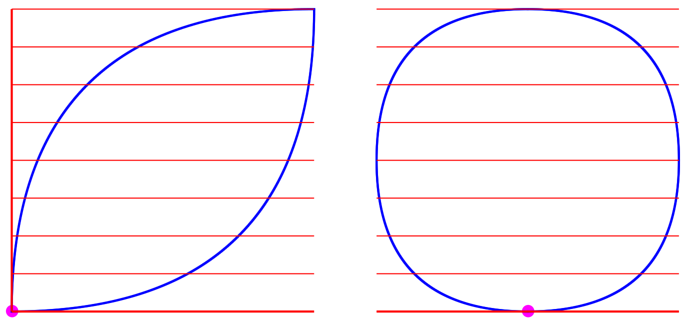

FIGURE IV.3.3. Tangent cone $T(S;x)$ (red) for a set S (blue) at a point x (magenta)

IV.3.8. Optimality conditions for non-linear constrained optimization problems. Assume that x^* is a local minimum of f, the gradients $Df_{p+1}(x^*), \ldots, Df_m(x^*)$ are linearly independent and there is a vector $s \in \mathbb{R}^n$ with $Df_j(x^*)s = 0$ for all $p+1 \le j \le m$ and $Df_i(x^*)s < 0$ for all those i with $1 \le i \le m$ and $f_i(x^*) = 0$. Then there is a vector $y^* \in \mathbb{R}^m$ such that (x^*, y^*) is a saddle point of the Lagrange function

$$\boxed{\mathcal{L}(x,y) = f(x) + \sum_{j=1}^{m} y_j f_j(x)}$$

and satisfies the following *Karush-Kuhn-Tucker conditions* or *KKT conditions* in short:

$$\boxed{\begin{aligned} Df(x^*) + \sum_{i=1}^{m} y_i^* Df_i(x^*) &= 0, \\ f_i(x^*)y_i^* &= 0, \quad 1 \le i \le p, \\ f_i(x^*) &\le 0, \quad 1 \le i \le p, \\ y_i^* &\ge 0, \quad 1 \le i \le p, \\ f_j(x^*) &= 0, \quad p+1 \le j \le m. \end{aligned}}$$

IV.3.9. Overview of algorithms for non-linear constrained optimization problems. The algorithms of sections IV.3.10 – IV.3.12 below aim at satisfying the KKT conditions of sections IV.3.3 and IV.3.8. The algorithm of section IV.3.13 on the other hand is completely different. In contrast to the other algorithms it in particular does not require any derivatives.

IV.3.10. Projection methods. Projection methods can be regarded as variants of the general descent algorithm IV.2.3 combined with an orthogonal projection onto the set S of admissible vectors satisfying the constraints.

For every convex set $S \subset \mathbb{R}^n$ and every point $x \in \mathbb{R}^n$ there is a unique point $P_S(x) \in S$, its *projection*, which is closest to x, i.e.

$$\|x - P_S(x)\| \le \|x - y\|$$

for all $y \in S$ (see figure IV.3.4).

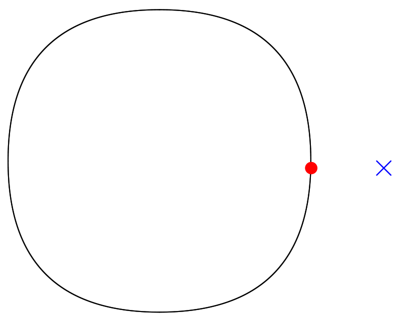

FIGURE IV.3.4. Projection $P_S(x)$ (red) of a point x (blue) onto a convex set S (black)

The projection P_S has the following properties

$$(P_S(y) - P_S(x))^t (y - x) \ge \|P_S(y) - P_S(x)\|^2$$
$$\|P_S(y) - P_S(x)\| \le \|x - y\|$$

for all $x, y \in \mathbb{R}^n$.

ALGORITHM IV.3.1. (Projection method)

(0) *Given: a convex set $S \subset \mathbb{R}^n$, an initial guess $x_0 \in S$ and parameters $\beta, \mu \in (0, 1)$ and $\gamma > 0$.*
Set $k = 0$.

(1) *Compute $Df(x_k)$.*

(2) *If*

$$Df(x_k)v \ge 0 \quad \textit{for all } v \in T(S; x_k)$$

stop, *otherwise proceed with step* (3).

(3) *Find the smallest integer m_k such that*

$$z_k = P_S(x_k - \beta^{m_k} \gamma Df(x_k))$$

satisfies

$$f(z_k) \le f(x_k) + \mu Df(x_k)(z_k - x_k).$$

Set

$$x_{k+1} = z_k,$$

increase k by 1 and return to step (1).

Algorithm IV.3.1 has the following properties:

It is a damped Newton's method combined with a projection onto the set S.
Its practicability hinges on the computability of the tangent cones and the ability to check the cone condition $Df(x_k)v \geq 0$.
Every accumulation point x^* of the generated sequence x_k satisfies the cone condition $Df(x^*)v \geq 0$ for all $v \in T(S;x^*)$.

IV.3.11. Penalty methods. The basic ideas of penalty methods can be summarized as follows:

- 'Penalize' the constraints.
- Solve unconstrained optimization problems incorporating the 'penalization'.
- If the penalty vanishes for the solution of the auxiliary unconstrained problem we have found a solution of the original constrained problem.
- Successively increase the penalty and hope that the solutions of the auxiliary problems converge to a solution of the original constrained problem.
- Either all constraints are penalized by a *penalty function* or only inequality constraints are penalized by a *barrier function*.

A function $\ell : \mathbb{R}^n \to \mathbb{R}$ is called a *penalty function* for the non-empty set $S \subset \mathbb{R}^n$ if

$$\ell(x) > 0 \quad \text{for all } x \notin S$$

and

$$\ell(x) = 0 \quad \text{for all } x \in S.$$

Example IV.3.2. The function

$$\ell(x) = \sum_{i=1}^{p} (f_i(x)^+)^\alpha + \sum_{j=p+1}^{m} |f_j(x)|^\alpha$$

with $\alpha > 0$ and

$$z^+ = \max\{z, 0\}$$

is a penalty function for the set

$$S = \{x \in \mathbb{R}^n : f_i(x) \leq 0, 1 \leq i \leq p, f_j(x) = 0, p+1 \leq j \leq m\}$$

associated with a general non-linear optimization problem.

Algorithm IV.3.3. (Penalty algorithm with general penalty function)

(0) *Given: initial guesses $x_0 \in \mathbb{R}^n$ and $r_0 > 0$, a continuous function $f : \mathbb{R}^n \to \mathbb{R}$, a non-empty closed set $S \subset \mathbb{R}^n$ and a penalty function ℓ for S.*
Set $k = 0$.

(1) *Compute an approximation x_k for a local minimum of*

$$p(x, r_k) = f(x) + r_k \ell(x).$$

(2) *If $x_k \in S$* stop. *Otherwise set*

$$r_{k+1} = 2r_k,$$

increase k by 1 and return to step (1).

Algorithm IV.3.3 has the following properties:

For sufficiently large r the function $p(x, r)$ admits a local minimum.
The sequence x_k converges to a local minimum $x^* \in S$ of the function f.

In order to take into account the particular structure of the non-linear constrained optimization problem, we introduce the so-called *augmented Lagrange function* associated with this problem:

$$\begin{aligned}\Lambda(x, y, r) = f(x) &+ \sum_{i=1}^{p} \frac{1}{2} r_i \left[\left(f_i(x) + \frac{y_i}{r_i} \right)^+ \right]^2 \\ &+ \sum_{j=p+1}^{m} \frac{1}{2} r_j \left[f_j(x) + \frac{y_j}{r_j} \right]^2 \\ &- \sum_{k=1}^{m} \frac{1}{2} \frac{y_k^2}{r_k}.\end{aligned}$$

Using this function algorithm IV.3.3 takes the following particular form:

Algorithm IV.3.4. (Penalty algorithm with augmented Lagrange function)

(0) *Given: a vector $r \in (\mathbb{R}_+^*)^m$ and an initial guess $y_0 \in (\mathbb{R}_+)^p \times \mathbb{R}^{m-p}$.*
Set $k = 0$.

(1) *Determine a local minimum x_k of the augmented Lagrange function $x \mapsto \Lambda(x, y_k, r)$.*

(2) *If (x_k, y_k) satisfies the Karush-Kuhn-Tucker conditions IV.3.8* stop. *Otherwise proceed with step (3).*

(3) *Set*

$$\begin{aligned} y_{k+1,i} &= (r_i f_i(x_k) + y_{k,i})^+ &&\textit{for } 1 \le i \le p, \\ y_{k+1,j} &= r_j f_j(x_k) + y_{k,j} &&\textit{for } p+1 \le j \le m. \end{aligned}$$

Increase k by 1 and return to step (1).

Algorithm IV.3.4 has the following properties:

> If $r = (\rho, \ldots, \rho)^t$ with a sufficiently large ρ, the algorithm converges to a saddle point of the Lagrange function $\mathcal{L}$.
> The convergence is linear.
> The convergence speed improves with increasing ρ.

A function $\mathcal{B} : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$ is called *barrier function* if it has the following properties:

> $\mathcal{B}(t) = \infty$ for all $t \le 0$.
> $\mathcal{B}$ is monotonically decreasing.
> $\mathcal{B}$ is convex.
> $\mathcal{B}$ is continuously differentiable on $\mathbb{R}_+^*$.
> $\lim\limits_{t \to 0+} \mathcal{B}(t) = \infty$.
> $\lim\limits_{t \to 0+} \mathcal{B}'(t) = -\infty$.

Example IV.3.5. The functions

$$\mathcal{B}(t) = \begin{cases} -\ln t & \text{for } t > 0 \\ \infty & \text{for } t \le 0 \end{cases}$$

$$\mathcal{B}(t) = \begin{cases} t^{-\alpha} & \text{for } t > 0 \\ \infty & \text{for } t \le 0 \end{cases}$$

with $\alpha > 0$ are barrier functions.

Algorithm IV.3.6. (Barrier algorithm for convex optimization)

(0) *Given: convex functions f and $f_1, \ldots, f_p$ and affine functions $f_{p+1}, \ldots, f_m$, a barrier function $\mathcal{B}$ and an initial guess $x_0 \in \mathbb{R}^n$ with $f_j(x_0) = 0$ for $p+1 \le j \le m$.*
Choose $\mu_0 > 0$ and $d_0 \in (\mathbb{R}_+^*)^p$ with $f_i(x_0) < d_{i,0}$ for $1 \le i \le p$.
Set $k = 0$.

(1) *Choose $\lambda_k \in (0, 1)$ with $f_i(x_k) < \lambda_k d_{i,k}$ for $1 \le i \le p$. Set*

$$\mu_{k+1} = \lambda_k \mu_k, \quad d_{k+1} = \lambda_k d_k.$$

(2) *Starting with x_k apply a line search to the problem*

$$\min_x \Big\{ f(x) + \mu \sum_{i=1}^{p} \mathcal{B}(d_i - f_i(x)) : f_j(x) = 0 \;,\; p+1 \le j \le m \Big\}$$

with result x_{k+1}. Increase k by 1 and return to step (1).

IV.3.12. SQP method. The basic idea of the *sequential quadratic programming method* or *SQP method* in short can be described as follows:

- Replace the Lagrange Function $\mathcal{L}$ by a second order approximation.
- Linearize the constraints.
- Successively solve constrained optimization problems with a quadratic object function and affine constraints.

This leads to the following algorithm.

Algorithm IV.3.7. (SQP algorithm)

(0) *Given: initial guesses $x_0 \in \mathbb{R}^n$ and $y_0 \in (\mathbb{R}_+^*)^p \times \mathbb{R}^{m-p}$. Compute*

$$B_0 = D^2 f(x_0) + \sum_{i=1}^m y_{0,i} D^2 f_i(x_0)$$

and set $k = 0$.

(1) *Find a solution (s, y) for the Karush-Kuhn-Tucker conditions of the auxiliary problem*

$$\min_s \Big\{ Df(x_k)s + \frac{1}{2} s^t B_k s \ : \\ f_i(x_k) + Df_i(x_k)s \le 0,\ 1 \le i \le p, \\ f_j(x_k) + Df_j(x_k)s = 0,\ p+1 \le j \le m \Big\}.$$

(2) *Set*

$$x_{k+1} = x_k + s, \quad y_{k+1} = y_k + y.$$

(3) *Compute*

$$B_{k+1} = D^2 f(x_{k+1}) + \sum_{i=1}^m y_{k+1,i} D^2 f_i(x_{k+1}),$$

increase k by 1 and return to step (1).

Algorithm IV.3.7 has the following properties:

It is locally quadratically convergent.
If the B_k are replaced by approximations in a suitable quasi Newton type, the convergence still is linear.

IV.3.13. The simplex method of Nelder and Mead. The basic idea of the *simplex method of Nelder and Mead* can be described as follows:

- Minimize a function f over $\mathbb{R}^n$.
- Take into account eventual constraints by setting $f(x) = \infty$ if x violates the constraints.
- Choose $n + 1$ points $x_0, \dots, x_n$ generating $\mathbb{R}^n$.
- Sort these points by increasing size of f.
- Reflect x_n at the barycentre of $x_0, \dots, x_{n-1}$ and eventually expand or contract the image x' depending on the values $f(x_0)$, …, $f(x_n)$ and $f(x')$.
- Replace an appropriate member of the list x_0, …, x_n by x'.

Note that the label 'simplex' for this algorithm does not imply any relation to the simplex algorithm for linear optimization problems. Further notice that $n+1$ points $x_0, \dots, x_n$ generate $\mathbb{R}^n$ if and only if the n vectors $x_1 - x_0, \dots, x_n - x_0$ are linearly independent.

ALGORITHM IV.3.8. (Simplex method of Nelder and Mead)

(0) *Given: $n + 1$ points $x_0, \dots, x_n \in \mathbb{R}^n$ generating $\mathbb{R}^n$ sorted by increasing size of f and a tolerance ε.*

(1) *Compute the mean value*

$$f = \frac{1}{n+1} \sum_{i=0}^{n} f(x_i)$$

and the standard deviation

$$d = \frac{1}{n+1} \sum_{i=0}^{n} \big(f(x_i) - \overline{f}\big)^2.$$

If $d \le \varepsilon$ stop.

(2) *Compute*

$$c = \frac{1}{n} \sum_{i=0}^{n-1} x_i, \quad x_r = 2c - x_n$$

and $f(x_r)$.

(3) *Distinguish the following cases:*

  (a) *If $f(x_0) \le f(x_r) \le f(x_{n-1})$*
  replace x_n by x_r (reflection).

  (b) *If $f(x_r) < f(x_0)$*
  compute

$$x_e = 2x_r - c$$

  and $f(x_e)$.
  If $f(x_e) < f(x_r)$
  replace x_r by x_e.
  Replace x_n by x_r (expansion).

(c) *If* $f(x_r) > f(x_{n-1})$
compute

$$x_c = \begin{cases} c + \frac{1}{2}(x_n - c) & \textit{if } f(x_r) \geq f(x_n) \\ c + \frac{1}{2}(x_r - c) & \textit{if } f(x_r) < f(x_n) \end{cases}$$

and $f(x_c)$.
If $f(x_c) < \min\{f(x_n), f(x_r)\}$
replace x_n *by* x_c,
otherwise compute

$$x_i = \frac{1}{2}(x_0 + x_i)$$

for $1 \leq i \leq n$ (contraction).

(4) *Re-sort* $x_0, \ldots, x_n$ *by increasing size of* f *and return to step* (1).

Algorithm IV.3.8 has the following properties:

It is very cheap since it does not require the computation of any derivative.
It is very slow.
It is very robust.
It may yield suitable initial guesses for the algorithms presented previously.
There is no convergence proof.

IV.4. Global Optimization Problems

All algorithms considered so far at best yield a *local minimum*. In some situations, however, we must find a *global minimum* or even all of them. Note, that this difficulty only arises for non-convex optimization problems since a convex function has at most one local minimum which is the global minimum.

IV.4.1. Structure of global optimization algorithms. All algorithms for global optimization have the following basic structures in common:

- Try several candidates for a global minimum.
- Eventually replace candidates by the result of a *local search*, i.e. apply one of the previously described algorithms with a given candidate as initial guess.
- Eventually iterate on lists of candidates.
- Eventually perturb candidates.

The algorithms differ by

- the *initial choice* of candidates,
- the method for *updating* the list of candidates,

- the form of *perturbation*,
- the amount of *randomness*,
- the work invested in local searches.

In what follows we will shortly address these topics.

IV.4.2. Initial choice of candidates. There two variants for choosing initial candidates: *deterministic* and *random*.

In the *deterministic* variant one covers the domain $S \subset \mathbb{R}^n$ of admissible points x by a uniform mesh (see the left part of figure IV.4.1).

In the *random* variant one covers the domain $S \subset \mathbb{R}^n$ of admissible points x by a random mesh according to a chosen probability measure, e.g. uniform distribution (see the right part of figure IV.4.1).

In both approaches one eventually constructs several lists of candidates by iteratively reducing the mesh size.

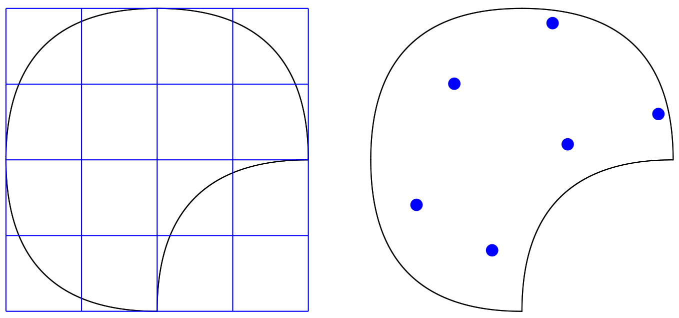

FIGURE IV.4.1. Deterministic (left) and random (right) mesh for an admissible domain S

IV.4.3. Updating lists of candidates. There are several possibilities for updating lists of candidates:

- Replace candidates by the result of a local search using one of the algorithms described in section IV.3.
- Replace candidates by a perturbation as described below.
- With a small probability also accept candidates with a larger value of f. In the method of *simulated annealing*, e.g., a point x' with $f(x') > f(x)$ is allowed to replace x with probability $e^{\frac{f(x)-f(x')}{T}}$ where the so-called *cooling time* T is chosen adaptively based on various heuristic criteria.
- Update lists by *branch and bound* techniques.

IV.4.4. Perturbation of candidates. We first normalize all points such that all their co-ordinates can be represented by an N-bit string. Then for a given candidate we pick one of its components by random and flip one of its bits by random. This creates perturbations which may be far off with respect to the standard Euclidean distance.

EXAMPLE IV.4.1. Choose $N = 4$ and

$$x = 15 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0.$$

Then

$$x' = 11 = 1 \cdot 2^3 + \mathbf{0} \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

and

$$x'' = 7 = \mathbf{0} \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

are perturbations of x whereas

$$\widetilde{x} = 9 = 1 \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{0} \cdot 2^1 + 1 \cdot 2^0$$

is not a perturbation of x.

IV.4.5. Concluding remarks. One always should keep in mind that each algorithm has its own benefits and drawbacks. The choice of an efficient algorithm requires knowledge of the particular structure of the given optimization problem. There is no efficient black-box algorithm solving all kind of problems.

Bibliography

[1] Dietrich Braess, *Finite Elements*, second ed., Cambridge University Press, Cambridge, 2001, Theory, fast solvers, and applications in solid mechanics, Translated from the 1992 German edition by Larry L. Schumaker.

[2] Eligius M. T. Hendrix and Boglárka G.-Tóth, *Introduction to Nonlinear and Global Optimization*, Springer Optimization and Its Applications, vol. 37, Springer, New York, 2010.

[3] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri, *Numerical Mathematics*, second ed., Texts in Applied Mathematics, vol. 37, Springer-Verlag, Berlin, 2007.

[4] Rüdiger Verfürth, *Computational Fluid Dynamics*, Lecture Notes, Ruhr-Universität Bochum, Bochum, May 2007, 127 pages, `www.rub.de/num1/files/lectures/CFD.pdf`.

[5] ———, *Adaptive Finite Element Methods*, Lecture Notes, Ruhr-Universität Bochum, Bochum, September 2010, 144 pages, `www.rub.de/num1/files/lectures/AdaptiveFEM.pdf`.

Index