Lecture Notes *Numerical Computation*

J.M. Melenk, M. Faustmann

Institute of Analysis und Scientific Computing
TU Wien
WS 2023/24

Contents

0 Introduction

The aim of this lecture is to give an overview of common elementary numerical methods. The goal of numerical methods is to approximately solve mathematical problems on computers (which oftentimes come from applications in physics, engineering, etc.) that have no known closed form solution.
Hereby, some key questions should be answered before implementing a method: Does the method produce a solution (i.e. convergence), is the solution the one I want (uniqueness), how accurate is my approximation and is the method efficient? In this lecture, these questions are mathematically discussed and answered for problems in interpolation, numerical integration, solution of linear systems and nonlinear equations, computation of eigenvalues and solution of differential equations.

The lecture is especially designed for the master studies Computational Science and Engineering (CSE) and Technical Informatics/Visual Computing at TU Wien.

The lecture notes assume that the reader is familiar with the following topics:

- basic calculus, convergence of sequences
- vector spaces, integration and differentiation in more variables,
- matrices, linear systems of equations, eigenvalues,
- ordinary differential equations.

For an overview of some of these topics, we refer the appendix of this document and to the lecture notes for the course *Applied Mathematics Foundations* by Markus Faustmann (downloadable at https://www.tuwien.at/mg/asc/faustmann/lehre/skripten).

This is version 2 of the lecture notes, written during the winter term 2023.

1 Polynomial Interpolation

The idea of polynomial interpolation is to find an easy function (here: a polynomial or a trigonometric polynomial) that matches some given data points exactly. As the data may come from real applications (e.g. measurements), the representation of it by an easy function can be used to make further calculations or predictions.

goal: given pairs of points x_i (also called knots) and values f_i, $i = 0, \dots, n$,

$$\text{find } p \in \mathcal{P}_n \text{ s.t. } p(x_i) = f_i,\ i = 0, \dots, n. \tag{1.1}$$

applications (examples):

- "Extrapolation": oftentimes the data is provided as function evaluations $f_i = f(x_i)$ for an (unknown) function f. Using polynomial interpolation gives a p such that $p(\overline{x})$ approximates $f(\overline{x})$ on the unknown values $\overline{x} \notin \{x_0, \dots, x_n\}$.
- "Dense output/plotting of f", if only the values $f_i = f(x_i)$ are given (or, e.g., function evaluations are too expensive).
- "Approximation of f": if the function f is available, but e.g. integration/differentiation is too expensive: measure f at data points $\rightarrow$ find the polynomial interpolation $p \rightarrow$ integrate or differentiate the interpolating polynomial p.

1.1 Existence and uniqueness of the polynomial interpolation problem

We first discuss, whether the problem (1.1) is uniquely soveable, i.e., if there is exactly one polynomial of degree $\leq n$ that satisfies the conditions. In order to do so, we introduce a useful basis for the space $\mathcal{P}_n$ of polynomials of degree $\leq n$.

Definition 1.1 (Lagrange polynomial) *Let the points x_i, $i = 0, \dots, n$, be distrinct. Then, the* Lagrange polynomials $(\ell_i)_{i=0}^n$ *w.r.t. the points $(x_i)_{i=0}^n$ are given by*

$$\ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

By definition, there holds

- $\ell_i \in \mathcal{P}_n$ for all $i = 0, \dots, n$;
- $\ell_i(x_j) = \delta_{ij}$, i.e., $\ell_i(x_i) = 1$ and $\ell_i(x_j) = 0$ for $j \neq i$.

This directly implies that the $(\ell_i)_{i=0}^n$ are $n+1$ linearly independent functions in $\mathcal{P}_n$. As $\dim \mathcal{P}_n = n + 1$, they also form a basis of $\mathcal{P}_n$. By definition of a basis, any $p \in \mathcal{P}_n$ can thus be written as

$$p(x) = \sum_{i=0}^{n} p_i \ell_i(x).$$

Now, if p should solve the interpolation problem, the coefficients p_i can be determined by the conditions $p(x_i) = f_i$. As $\ell_i(x_j) = \delta_{ij}$, we calculate

$$f_j = p(x_j) = \sum_{i=0}^{n} p_i \ell_i(x_j) = p_j.$$

We summarize this in the following theorem, that also takes care of uniqueness.

Theorem 1.2 (Lagrange interpolation) *Let the points x_i, $i = 0, \ldots, n$, be distinct. Then there exists, for all values $(f_i)_{i=0}^n \subset \mathbb{R}$, a unique interpolating polynomial $p \in \mathcal{P}_n$. It is given by*

$$p(x) = \sum_{i=0}^{n} f_i \ell_i(x), \qquad \ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}. \tag{1.2}$$

Proof: Uniqueness: Let p_1, $p_2 \in \mathcal{P}_n$ be two interpolating polynomials. Then, the difference $p := p_1 - p_2$ is a polynomial of degree n with (at least) $n + 1$ zeros. A mathematical theorem ("fundamental theorem of algebra") states that a non-zero polynomial of (exact) degree n has exactly n zeros (counting multiplicity). Hence, $p \equiv 0$, i.e., $p_1 = p_2$. □

Example 1.3 *The polynomial $p \in \mathcal{P}_2$ interpolating the data*

$$(0, 0), \quad \left(\frac{\pi}{4}, \frac{\sqrt{2}}{2}\right), \quad \left(\frac{\pi}{2}, 1\right)$$

is given by

$$\begin{aligned}
p(x) &= 0 \cdot \ell_0(x) + \frac{\sqrt{2}}{2} \cdot \ell_1(x) + 1 \cdot \ell_2(x), \\
\ell_0(x) &= \frac{(x - \pi/4)(x - \pi/2)}{(0 - \pi/4)(0 - \pi/2)} = 1 - (1.909...)x + (0.8105...)x^2, \\
\ell_1(x) &= \frac{(x - 0)(x - \pi/2)}{(\pi/4 - 0)(\pi/4 - \pi/2)} = (2.546...)x - (1.62...)x^2 \\
\ell_2(x) &= \frac{(x - 0)(x - \pi/4)}{(\pi/2 - 0)(\pi/2 - \pi/4)} = -(0.636...)x + (0.81...)x^2.
\end{aligned}$$

That is, $p(x) = (1.164...)x - (0.3357...)x^2$

In fact the given data points f_i are function evaolutions of the function $f(x) = \sin(x)$, i.e., $f_i = \sin(x_i)$. As we have computed the interpolation polynomial, we can now easily obtain approximations to

- *$f'(0) = 1$ by $f'(0) \approx p'(0) = 1.164...$;*
- *$\int_0^{\pi/2} f(x)\,dx = 1$ by $\int_0^{\pi/2} p(x)\,dx = 1.00232....$*

finis 1.DS

1.2 Neville scheme

It is not efficient to evaluate the interpolating polynomial $p(x)$ at a point x based on (1.2) since it involves many (redundant) multiplications when evaluating the ℓ_i. Traditionally, an interpolating polynomial is evaluated at a point x with the aid of the *Neville scheme*.

The main idea of the Neville scheme is that the interpolating polynomial p_n in $n+1$ knots can be constructed from certain interpolation polynomials of degree $n-1$. Denote by $p_{0,n-1}$ the interpolating polynomial for the pairs (x_i, f_i) with $i = 0, \ldots, n-1$ and by $p_{1,n-1}$ the interpolating polynomial for the pairs (x_i, f_i) with $i = 1, \ldots, n$. Then, the function

$$\pi(x) = \alpha(x - x_n)p_{0,n-1}(x) + \beta(x - x_0)p_{1,n-1}(x)$$

with $\alpha, \beta \in \mathbb{R}$ is a polynomial of degree n. In order for it to satisfy the interpolation condition, we calculate

$$\pi(x_0) = \alpha(x_0 - x_n)p_{0,n-1}(x_0) = \alpha(x_0 - x_n)f_0 \overset{!}{=} f_0$$
$$\pi(x_n) = \beta(x_n - x_0)p_{1,n-1}(x_n) = \beta(x_n - x_0)f_n \overset{!}{=} f_n$$

Thus, $\alpha = -\beta = -\frac{1}{x_n - x_0}$ and the polynomial

$$\pi(x) = \frac{(x - x_0)p_{1,n-1}(x) - (x - x_n)p_{0,n-1}(x)}{x_n - x_0}$$

satisfies for $j = 1, \ldots, n-1$

$$\pi(x_j) = \frac{(x_j - x_0)f_j - (x_j - x_n)f_j}{x_n - x_0} = f_j.$$

Thus, π solves the polynomial interpolation problem and by uniqueness is the solution we are looking for. Now, the same concepts can be applied for the polynomials $p_{0,n-1}$, $p_{1,n-1}$, i.e., they can be expressed by some interpolation polynomials of degree $n-1$ and so on. This motivates the following theorem, which is shown with the exact same calculations/ideas as above (exercise!).

Theorem 1.4 *Let $x_0, \ldots, x_n$, be distinct knots and let f_i, $i = 0, \ldots n$, be the corresponding values. Denote by $p_{j,m} \in \mathcal{P}_m$ the solution of*

$$\textit{find } p \in \mathcal{P}_m, \ \textit{s.t. } p(x_k) = f_k \textit{ for } k = j, j+1, \ldots, j+m. \tag{1.3}$$

Then, there hold the recursions:

$$p_{j,0} = f_j, \ \ j = 0, \ldots, n \tag{1.4}$$
$$p_{j,m}(x) = \frac{(x - x_j)p_{j+1,m-1}(x) - (x - x_{j+m})p_{j,m-1}(x)}{x_{j+m} - x_j} \qquad m \geq 1 \tag{1.5}$$

The solution p of (1.1) is $p(x) = p_{0,n}(x)$.

Theorem 1.4 shows that evaluating p at x can be realized with the following scheme:

$$
\begin{array}{c|cccccccccc}
x_0 & f_0 & =: & p_{0,0}(x) & \longrightarrow & p_{0,1}(x) & \longrightarrow & p_{0,2}(x) & \longrightarrow \ \ldots \ \longrightarrow & p_{0,n}(x) = p(x) \\
 & & & & \nearrow & & \nearrow & \vdots & & \nearrow \\
x_1 & f_1 & =: & p_{1,0}(x) & \longrightarrow & p_{1,1}(x) & & \vdots & & \\
 & & & & \nearrow & \vdots & & \vdots & & \\
x_2 & f_2 & =: & p_{2,0}(x) & & \vdots & & \vdots & & \\
\vdots & \vdots & & \vdots & & \vdots & & \vdots & & \\
\vdots & \vdots & & \vdots & & \vdots & \longrightarrow & p_{n-2,2}(x) & & \\
\vdots & \vdots & & \vdots & & \vdots & \nearrow & & & \\
\vdots & \vdots & & \vdots & \longrightarrow & p_{n-1,1}(x) & & & & \\
\vdots & \vdots & & \vdots & \nearrow & & & & & \\
x_n & f_n & =: & p_{n,0}(x) & & & & & &
\end{array}
$$

⟦ here, the operation " $\begin{array}{c}\longrightarrow\\ \nearrow\end{array}$ " is realized by formula (1.5) ⟧

slide 1 - Neville scheme example

Exercise 1.5 *Formulate explicitly the algorithm that computes (in a 2-dimensional array) the values $p_{i,j}$. How many multiplications (in dependence on n) are needed? (It suffices to state α in the complexity bound $O(n^\alpha)$.)*

The scheme computes the values "column by column". If *merely* the last value $p(x)$ is required, then one can be more memory efficient by overwriting the given vector of data:

Algorithm 1.6 (Aitken-Neville Scheme)
Input: knot vector $x \in \mathbb{R}^{n+1}$, vector $f \in \mathbb{R}^{n+1}$ of values, evaluation point $\overline{x} \in \mathbb{R}$
Output: $p(\overline{x})$, p solves (1.1)

for $m = 1 : n$ ***do***
  for $j = 0 : n - m$ ***do*** ▷ *array has triangular form*
    $f_j := \frac{(\overline{x}-x_j)\, f_{j+1} - (\overline{x}-x_{j+m})\, f_j}{x_{j+m}-x_j}$
  end for
end for
return f_0

Remark 1.7
- *Cost of Alg. 1.6: $O(n^2)$ arithmetic operations.*
- *The knots x_i need <u>not</u> be sorted.*
- *The Neville scheme, i.e., the algorithm formulated in Exercise 1.5 is particularly convenient, if additional data points are added at a later time: one merely appends one additional row at the bottom.*

1.3 Newton representation of the interpolating polynomial (CSE)

The cost of evaluating the interpolating polynomial p at *a single* point x is $O(n^2)$. If the interpolating polynomial has to be evaluated in *many* points x (e.g., for plotting), then it is of interest to reduce the cost (i.e., number of floating point operations) from $O(n^2)$ to $O(n)$ per evaluation point x. The "classical" way to achieve this is with the *Horner scheme*, which actually works in a different polynomial basis.

Definition 1.8 *The* **Newton polynomials** ω_j, $j = 0, \ldots, n$, *w.r.t. the knots* $x_0, x_1, \ldots, x_n$, *are defined by*

$$\omega_j(x) := \prod_{i=0}^{j-1} (x - x_i).$$

Note: an empty product is defined to be 1.

Written explicitly they read as

$$1, \ (x - x_0), \ (x - x_0)(x - x_1), \quad \ldots, \quad (x - x_0)(x - x_1) \cdots (x - x_{n-1}). \tag{1.6}$$

These polynomials form a basis of $\mathcal{P}_n$. That is, for every polynomial $p(x)$ of degree n there are coefficients $d_0, \ldots, d_n$, such that

$$\begin{aligned} p(x) = d_0 \cdot 1 + d_1(x - x_0) + d_2(x - x_0)(x - x_1) + d_3(x - x_0)(x - x_1)(x - x_2) \\ + \ \ldots \ + d_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned} \tag{1.7}$$

Example A particular case is $x_0 = x_1 = \cdots = x_{n-1}$. Then, the representation (1.7) of p is the Taylor polynomial (around x_0). □

Once the coefficients d_i are available, the polynomial $p(x)$ can be evaluated very efficiently by rearranging (1.7) as follows:

$$\begin{aligned} p(x) &= d_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1) + \ldots + d_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}) = \\ &= d_0 + (x - x_0)\Big[d_1 + (x - x_1)\Big[d_2 + (x - x_2)\Big[\ldots\big[d_{n-1} + (x - x_{n-1})[d_n]\big]\ldots\Big]\Big]\Big] \end{aligned}$$

This procedure is formalized in the following "Horner scheme":

Algorithm 1.9 (Horner scheme)
Input: *knots* x_i, *coefficients* d_i, *evaluation point* $\overline{x}$
Output: $p(\overline{x}) = \sum_{j=0}^{n} d_j \, \omega_j(\overline{x})$

$y := d_n$
for $j = n - 1 : -1 : 0$ ***do***
  $y = d_j + (\overline{x} - x_j)y$
end for
return y

Remark 1.10 *The computational cost of the method is:*

- $O(n^2)$ *to compute the coefficients* d_j *(→ see below);*
- $O(n)$ *to evaluate* $p(x)$ *using Alg. 1.9.*

$\Rightarrow$ *Horner scheme is useful, if* p *is evaluated at "many" points* x*.*
The Horner scheme is particularly economical on multiplications. Thus, the Horner scheme is useful in situations where multiplications are expensive. An example is the evaluation of matrix polynomials $p(\mathbf{A}) = \sum_{i=0}^{n} a_i \mathbf{A}^i$*, since the multiplication of two* $N \times N$ *matrices* $\mathbf{A}$, $\mathbf{B}$ *costs* $O(N^3)$ *floating point operations.*

Example 1.11 *(Conversion of binary numbers into decimal numbers) The binary number* 1011_{binary} *means* $1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3$*. With* $x = 2$*, we have to evaluate a polynomial at* $x = 2$*, which can be done efficiently with the Horner scheme.*

We now answer the question how to determine the coefficients d_i in (1.7) for given data

$$(x_0, f_0), \quad (x_1, f_1), \quad \ldots, \quad (x_n, f_n).$$

This is achieved by using successively the interpolation conditions:

$x = x_0$ in (1.7)

$$f_0 = p(x_0) = d_0 \tag{1.8}$$

$x = x_1$ in (1.7)

$$f_1 = p(x_1) = d_0 + d_1(x_1 - x_0) = f_0 + d_1(x_1 - x_0)$$
$$\Rightarrow d_1 = \frac{f_1 - f_0}{x_1 - x_0} \tag{1.9}$$

$x = x_2$ in (1.7)

$$\begin{aligned} f_2 = p(x_2) &= d_0 + d_1(x_2 - x_0) + d_2(x_2 - x_0)(x_2 - x_1) \\ &= f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x_2 - x_0) + d_2(x_2 - x_0)(x_2 - x_1) \end{aligned}$$

Rearranging yields

$$\begin{aligned}
f_2 - f_1 + f_1 - f_0 - \frac{f_1 - f_0}{x_1 - x_0}(x_2 - x_0) &= d_2(x_2 - x_0)(x_2 - x_1) \\
\iff \frac{f_2 - f_1}{x_2 - x_1} + \frac{(f_1 - f_0)(x_1 - x_0)}{(x_1 - x_0)(x_2 - x_1)} - \frac{(f_1 - f_0)(x_2 - x_0)}{(x_1 - x_0)(x_2 - x_1)} &= d_2(x_2 - x_0) \\
\iff \frac{f_2 - f_1}{x_2 - x_1} - \frac{(f_1 - f_0)(x_0 - x_1) + (f_1 - f_0)(x_2 - x_0)}{(x_1 - x_0)(x_2 - x_1)} &= d_2(x_2 - x_0) \\
\iff \frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0} &= d_2(x_2 - x_0)
\end{aligned}$$

and finally

$$\frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0} = d_2 \tag{1.10}$$

$$\vdots$$

(1.8), (1.9), and (1.10) suggest to define the so-called **divided differences** :

zeroth divided difference

$$f[x_0] := f(x_0) = f_0$$

first divided difference

$$f[x_0, x_1] := \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f_1 - f_0}{x_1 - x_0} = \frac{f[x_1] - f[x_0]}{x_1 - x_0}$$

second divided difference

$$f[x_0, x_1, x_2] := \frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0} = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

We recognize how the k-th divided difference should be defined:
The denominator is the difference $x_k - x_0$, the numerator is the difference between the $(k-1)$-th divided difference for the knots $x_1, \ldots, x_k$ and the $(k-1)$-th divided difference for the knots $x_0, x_1, \ldots, x_{k-1}$. Formally:

Definition 1.12 *The* **divided differences** *are given by the following recursion:*

$$f[x_i] = f(x_i) = f_i, \quad i = 0, 1, \ldots, n,$$

and

$$f[x_0, x_1, \ldots, x_k] := \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}. \tag{1.11}$$

The above discussion suggests that the coefficients d_i in (1.7) are given by the divided differences. This is indeed the case:

Theorem 1.13 *Let the knots $x_0, \ldots, x_n$ be distinct. Then, the interpolating polynomial p has the form*

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ + \ldots + f[x_0, x_1, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}). \tag{1.12}$$

Proof: For any polynomial $\pi \in \mathcal{P}_n$ of the form $\pi(x) = \sum_{i=0}^{n} a_i x^i$ we define its *leading coefficient* $lc(\pi) := a_n$. We show with the notation of Theorem 1.4 that, for any j, k,

$$lc(p_{j,k}) = f[x_j, \ldots, x_{j+k}]. \tag{1.13}$$

To see (1.13), we proceed by induction on k. By definition, we have $p_{j,0} = f[x_j]$ for all j. Let us assume that (1.13) holds true for all $k \leq K$. Then with the aid of Theorem 1.4

$$lc(p_{j,K+1}) \overset{\text{Thm. 1.4}}{=} \frac{lc(p_{j+1,K}) - lc(p_{j,K})}{x_{j+(K+1)} - x_j} \overset{\text{induction hyp.}}{=} \frac{f[x_{j+1}, \ldots, x_{j+1+K}] - f[x_j, \ldots, x_{j+K}]}{x_{j+(K+1)} - x_j} \\ \overset{\text{Def. 1.12}}{=} f[x_j, \ldots, x_{j+K+1}].$$

This shows (1.13). From (1.13) we obtain the claim of the theorem (why?). □

Remark 1.14 *Divided differences can be interpreted as approximations to derivatives.*

1. *Consider the specific knots $x_1 = x_0 + h$, $x_2 = x_0 + 2h$, $x_3 = x_0 + 3h, \ldots$ for small h. Then we have (the $\approx$ becomes an equality in the limit $h \to 0$):*

$$\begin{aligned} f[x_0, x_1] &= \frac{f_1 - f_0}{h} \approx f'(x_0) \\ f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{2h} \approx \frac{1}{2}\frac{f'(x_1) - f'(x_0)}{h} \approx \frac{1}{2} f''(x_0) \\ f[x_0, x_1, x_2, x_3] &= \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{3h} \approx \frac{1}{3}\frac{\frac{1}{2}f''(x_1) - \frac{1}{2}f''(x_0)}{h} \approx \frac{1}{2 \cdot 3} f'''(x_0). \end{aligned}$$

   In general, one has

$$f[x_0, x_1, \ldots, x_k] \approx \frac{1}{k!} f^{(k)}(x_0). \tag{1.14}$$

2. *This observation suggests to* define *for $x_0 = x_1 = \ldots = x_k$ the divided difference by*

$$f[x_0, x_1, \ldots, x_k] := \frac{1}{k!} f^{(k)}(x_0).$$

   This definition also allows one to generalize the definition of divided differences to the case when some knots coincide, for which the statement of Theorem 1.13 also holds.

3. *In general, for any knot sequence $x_0, \ldots, x_n$ there is an intermediate point*

$$\xi \in (\min\{x_0, \ldots, x_k\}, \max\{x_0, \ldots, x_k\})$$

   such that

$$f[x_0, \ldots, x_k] = \frac{1}{k!} f^{(k)}(\xi).$$

Exercise 1.15 *Formulate an algorithm similar to the Neville scheme to compute the divided differences $f[x_0], \ldots, f[x_0, \ldots, x_n]$. How expensive is the evaluation of an interpolating polynomial of degree n in M points?*

1.4 Extrapolation as a prime application of the Neville scheme

A typical application of the Neville scheme is the extrapolation of a function value that is not directly accessible. Given $n+1$ pairs (x_i, f_i), a approximation to a function value $f(\overline{x})$ can be found by evaluating the interpolation polynomial $p_n \in \mathcal{P}_n$, i.e.,

$$p_n(\overline{x}) \approx f(\overline{x}).$$

A prime application of this method is to compute derivatives of a function (here at 0), for which only function values are available. Define for $h \neq 0$ the difference quotient

$$D(h) = \frac{f(0+h) - f(0)}{h}.$$

If f is differentiable, then the limit $\lim_{h\to 0} D(h) = f'(0)$ exists and is the sought derivative. As the value is unknown, one can compute the values of $D(h_j)$ for $h_j > 0$ (small) and then evaluate the corresponding interpolation polynomial at $h = 0$ to obtain an approximation to $f'(0)$.

Exercise 1.16 *Let $f(x) = \exp(x)$. We seek an approximation to $u'(0)$. Define the function $h \mapsto D(h)$ as above.*
Compute the Neville scheme for $h = 2^{-j}$, $j = 0, 1, \ldots, 10$. Compute a second array containing the actual errors (recall: $f'(0) = \exp(0) = 1$). What do you observe in the first, second, and third column of the Neville scheme?

slide 2 - Extrapolation example

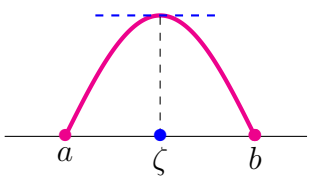
1.5 A simple error estimate

In this subsection, we now assume that the values f_i are point evaluations of a function f, i.e., $f_i = f(x_i)$ and that the knots x_i, $i = 0, \ldots, n$ are distinct.

Question: how big is the error $f(x) - p(x)$ for the interpolating polynomial p?

Our aim is to actually write the error in terms of an evaluation of polynomial depending only on the knots and some term that does depend on the function f.
Let $\omega_{n+1}(x) := \prod_{j=0}^{n}(x - x_j)$ be the Newton polynomial of degree $n+1$.

- In the knots x_j the error is zero by definition.
- In order to derive a formula for the error at $x \notin \{x_0, \ldots, x_n\}$, we employ the so called mean value theorem/Rolle's theorem: For functions $g \in C^1([a,b])$ for an interval $[a,b]$ with $g(a) = g(b)$, there exists $\xi \in (a,b)$ such that $g'(\xi) = 0$.



- Let $x \notin \{x_0, \ldots, x_n\}$ be fixed. Therefore, we consider the function

$$t \mapsto g(t) := f(t) - p(t) - K\omega_{n+1}(t), \qquad .$$

with some constant K depending on x. Choosing $K := \frac{f(x)-p(x)}{\omega_{n+1}(x)}$ gives $g(x) = 0$, which implies the formula for the error

$$f(x) - p(x) = K\omega_{n+1}(x).$$

- We now aim to find a formula for K that does not depend on p.

As p interpolates f and $\omega_{n+1}(x_j) = 0$, there also holds that $g(x_j) = 0$ for all knots x_j, $j = 0, \ldots, n$. Thus, g has at least $n + 2$ zeros. By Rolle's theorem, g' has at least $n + 1$ distinct zeros and therefore by Rolle's theorem again, g'' has n distinct zeros. Repeating these considerations one sees that $g^{(n+1)}$ has at least one zero ξ. Hence, (note: $p^{(n+1)} \equiv 0$ since $p \in \mathcal{P}_n$ and $\omega_{n+1}^{(n+1)}(x) = (n + 1)!$)

$$0 = g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - p^{(n+1)}(\xi) - K\omega_{n+1}^{(n+1)}(\xi) = f^{(n+1)}(\xi) - K(n + 1)!.$$

Hence, $K = \frac{f^{(n+1)}(\xi)}{(n+1)!}$.

We thus have shown the following theorem.

Theorem 1.17 *Let $[a, b] \subset \mathbb{R}$ and the knots $x_i \in [a, b]$, $i = 0, \ldots, n$, be distinct. Let $f \in C^{n+1}([a, b])$, and let p be the interpolating polynomial. Then for $x \in [a, b]$ there exists a $\xi \in (a, b)$ such that*

$$f(x) - p(x) = (x - x_0) \cdots (x - x_n) \frac{f^{(n+1)}(\xi)}{(n + 1)!} = \omega_{n+1}(x) \frac{f^{(n+1)}(\xi)}{(n + 1)!}. \tag{1.15}$$

The error formula (1.15) yields bounds for the interpolation error as seen in the following example.

Example 1.18 *(cf. Example 1.3) Let $f(x) = \sin x$ and $[a, b] = [0, \pi/2]$. Let $x_0 = 0$, $x_1 = \pi/4$, $x_2 = \pi/2$. Then the interpolating polynomial $p \in \mathcal{P}_2$ satisfies in view of $\max_{y\in\mathbb{R}} |f^{(3)}(y)| = \max_{y\in\mathbb{R}} |-\cos y| \leq 1$*

$$|f(x) - p(x)| \leq |\omega_3(x)| \frac{|f^{(3)}(\xi)|}{3!} \leq \frac{1}{6}|\omega_3(x)| = \frac{1}{6}|(x - 0)(x - \pi/4)(x - \pi/2)|.$$

Fig. 1.1 visualizes this estimate. The upper bound is pretty good in this example: it overestimates the error merely by a factor 1.5.

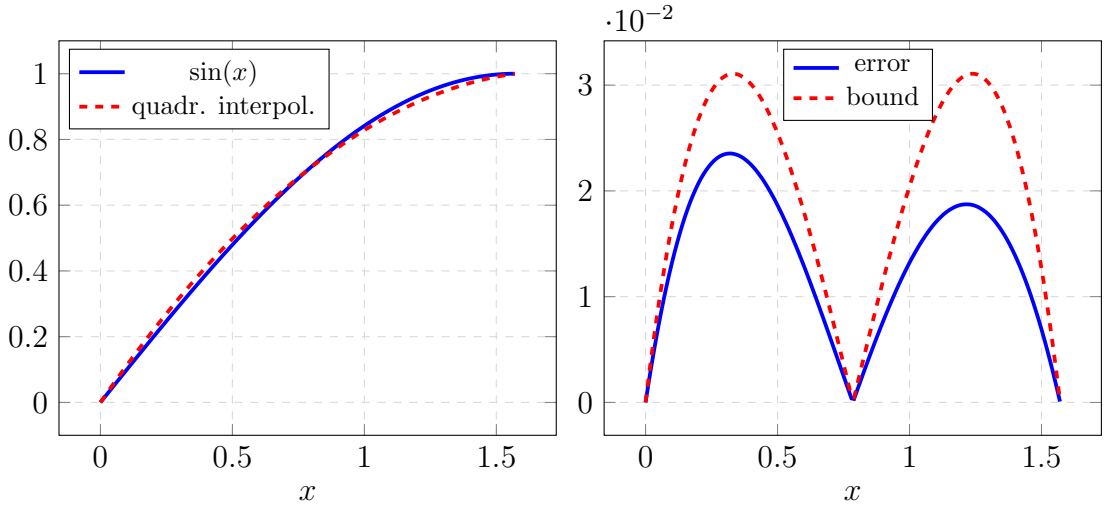


Figure 1.1: Left: $f(x)$ and the interpolating polynomial for Example 1.18. Right: absolute value of the error and upper bound.

Example 1.18 generalizes as follows:

Theorem 1.19 *Let $f \in C^{n+1}([a,b])$ and $h_i = q^i$, $i = 0, 1, \ldots$, for a chosen $0 < q < 1$. Let $x_0 \in [a,b]$. Denote by $p_{i,m} \in \mathcal{P}_m$ the polynomial that interpolates f in the points $x_0 + h_{i+j}$, $j = 0, \ldots, m$. Then there exists a constant $C > 0$ (which depends on f, m, and q), such that for $m \leq n+1$*

$$|f(x_0) - p_{i,m}(x_0)| \leq C h_i^{m+1} \tag{1.16}$$

Proof: From Theorem 1.17, we get (exercise!) for some $\xi \in (x_0, x_0 + h_i)$

$$|f(x_0) - p_{i,m}(x_0)| \leq \frac{1}{(m+1)!} |f^{(m+1)}(\xi)| \left| \prod_{j=0}^{m} (x_0 - (x_0 + h_{i+j})) \right| \leq C h_i^{m+1}.$$

□

If the function f is smooth, then the difference quotient $D(h) = \frac{f(x_0+h)-f(x_0)}{h}$ is a smooth function of h (Taylor expansion!). In that case, we may apply Theorem 1.19 to the function $h \mapsto D(h)$. Then, Theorem 1.19 explains the convergence behavior that was observed in Exercise 1.16 and silde 2 for the columns of the Neville scheme.

The assumption that f be smooth (i.e. n in Theorem 1.19 is fairly large), is essential for the rapid convergence behavior in the columns of the Neville scheme:

Example 1.20 **slide 2 - Extrapolation example**
Consider the Neville scheme as in Exercise 1.16 for the function $u(x) = |x|^{3/2}$, i.e., $D(h) = \sqrt{|h|}$. Then D is not smooth—it is not even differentiable at $h = 0$. Fig. 1.2 shows the errors $|D(0) - p_{i,m}(0)|$. We observe that increasing m does not lead to better results.

h	$m=0$	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$	$m=7$
2^0	1.00_0	4.14_{-1}	2.52_{-1}	1.68_{-1}	1.15_{-1}	8.06_{-2}	5.66_{-2}	3.99_{-2}
2^{-1}	7.07_{-1}	2.93_{-1}	1.79_{-1}	1.19_{-1}	8.17_{-2}	5.70_{-2}	4.00_{-2}	2.82_{-2}
2^{-2}	5.00_{-1}	2.07_{-1}	1.26_{-1}	8.40_{-2}	5.77_{-2}	4.03_{-2}	2.83_{-2}	
2^{-3}	3.54_{-1}	1.46_{-1}	8.93_{-2}	5.94_{-2}	4.08_{-2}	2.85_{-2}		
2^{-4}	2.50_{-1}	1.04_{-1}	6.31_{-2}	4.20_{-2}	2.89_{-2}			
2^{-5}	1.77_{-1}	7.32_{-2}	4.46_{-2}	2.97_{-2}				
2^{-6}	1.25_{-1}	5.18_{-2}	3.16_{-2}					
2^{-7}	8.84_{-2}	3.66_{-2}						
2^{-8}	6.25_{-2}							
Error	$\sqrt{h}$	$\sqrt{h}$	$\sqrt{h}$	$\sqrt{h}$	$\sqrt{h}$	$\sqrt{h}$		

Figure 1.2: (cf. Example 1.20) Extrapolation error at $h = 0$ for the function $h^{-1}(u(h) - u(0))$ with $u(x) = |x|^{3/2}$. The subscript numbers denote powers of 10.

Often the interpolation error is measured in a norm, e.g., the *maximum norm*. For an interval $[a, b]$, the maximum norm $\|g\|_{\infty,[a,b]}$ of a function $g \in C([a, b])$ is defined by

$$\|g\|_{\infty,[a,b]} := \max_{x\in[a,b]} |g(x)|. \tag{1.17}$$

Theorem 1.17 implies for the interpolation error

$$\|f - p\|_{\infty,[a,b]} \le \|\omega_{n+1}\|_{\infty,[a,b]} \frac{\|f^{(n+1)}\|_{\infty,[a,b]}}{(n+1)!} \le (b-a)^{n+1} \frac{\|f^{(n+1)}\|_{\infty,[a,b]}}{(n+1)!}.$$

Often, one approximates functions by *piecewise* polynomials as illustrated in the following exercise.

Exercise 1.21 *The goal is to approximate the function f on the interval $[a, b]$ by a piecewise polynomial of degree n. Proceed as follows: Partition $[a, b]$ in N subintervals $[t_j, t_{j+1}]$, $j = 0, \ldots, N-1$, of length $h = (b-a)/N$ with $t_j = a + jh$. In each subinterval $[t_j, t_{j+1}]$ select as the interpolation points $x_{i,j} := t_j + \frac{1}{n} ih$, $i = 0, \ldots, n$, and approximate f on $[t_j, t_{j+1}]$ by the polynomial that interpolates f in the points $x_{i,j}$, $i = 0, \ldots, n$. In this way, one obtains a function p that is a polynomial of degree n on each subinterval. Show:*

$$\|f - p\|_{\infty,[a,b]} \le \frac{1}{(n+1)!} h^{n+1} \|f^{(n+1)}\|_{\infty,[a,b]}.$$

Sketch the function p for the case $n = 1$.

finis 2.DS

1.6 Chebyshev interpolation

Question: If one is allowed to choose the interpolation points, which one should one choose?

For large n, the choice of the interpolation points may strongly impact the approximation quality of the interpolation process as we will see in the following.

1.6.1 Uniform point distribution

The easiest choice that comes to mind would be to distribute the knots uniformly in the considered interval $[a, b]$, i.e., take

$$x_j = a + \frac{b-a}{n} j \quad j = 0, \dots, n.$$

However, the following example illustrates that this might not be a good choice - even for fairly harmless functions f.

Example 1.22 (Runge example) *Consider $f(x) = (1 + 25x^2)^{-1}$ on the interval $[-1, 1]$. Fig. 1.5 shows the interpolation in equidistant points. We clearly observe failure for the interpolation in equidistant points as n grows.*

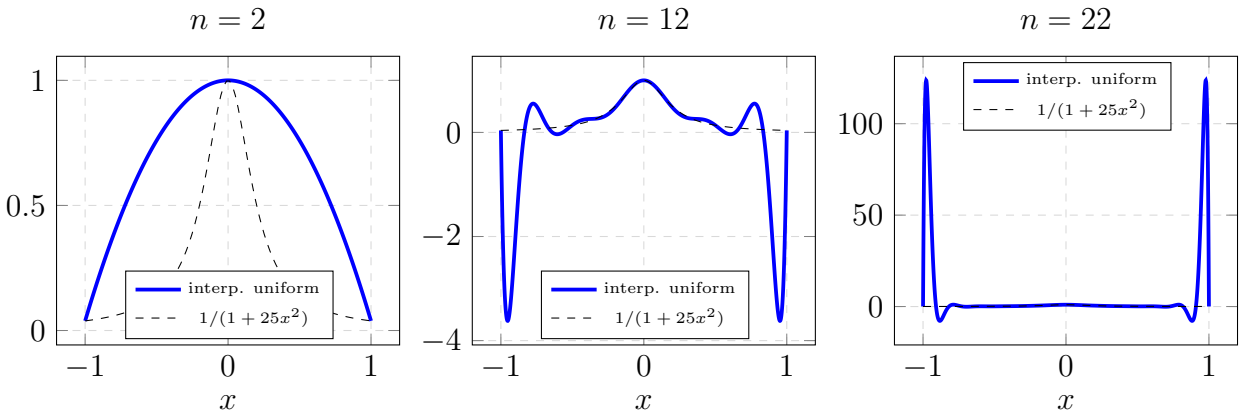


Figure 1.3: Interpolation of $(1 + 25x^2)^{-1}$ on $[-1, 1]$ using equidistant points ($n = 2$, 12, 22).

1.6.2 Chebyshev points

We now aim to present a better choice of interpolation points that leads to more satisfactory results for large polynomial degrees.

The representation of the interpolation error (1.15) has the advantage of being an *equality*. It has the disadvantage that the intermediate point ξ is not known and depends on the function f and the chosen knots x_i. Typically, one does not study the error in single points but studies the interpolation error in a norm. Here, we consider the maximum norm and estimate

$$\|f - p\|_{\infty,[a,b]} \leq \underbrace{\|\omega_{n+1}\|_{\infty,[a,b]}}_{\text{depends } \textit{solely} \text{ on the knots}} \underbrace{\frac{\|f^{(n+1)}\|_{\infty,[a,b]}}{(n+1)!}}_{\text{depends } \textit{solely} \text{ on } f \text{ and } n}$$

This shows that a sensible strategy to choose the knots x_i, $i = 0, \dots, n$, is to minimize $\|\omega_{n+1}\|_{\infty,[a,b]}$ (recall $\omega_{n+1}(x) = (x - x_0) \cdots (x - x_n)$):

$$\text{given } n, \text{ find } x_i \in [a, b] \text{ such that } \|\omega_{n+1}\|_{\infty,[a,b]} \text{ is minimal.} \tag{1.18}$$

This minimization problem has a solution, the so-called Chebyshev points:

Theorem 1.23 (Chebyshev points) *The minimization problem* (1.18) *has a solution given by*

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} x_{i,n}^{Cheb}, \qquad x_{i,n}^{Cheb} := \cos\left(\pi \frac{2i+1}{2n+2}\right), \qquad i = 0, \dots, n. \tag{1.19}$$

For this choice of interpolation points, there holds

$$\|\omega_{n+1}^{Cheb}\|_{\infty,[a,b]} = 2\left(\frac{b-a}{4}\right)^{n+1}.$$

In particular, for every choice of interpolation points x_i with corresponding polynomial ω_{n+1} there holds

$$\|\omega_{n+1}\|_{\infty,[a,b]} \geq \|\omega_{n+1}^{Cheb}\|_{\infty,[a,b]}.$$

Example 1.24 *The Chebyshev points $x_{i,n}^{Cheb}$, $i = 0, \dots, n$, for the interval $[-1, 1]$ are* not *uniformly distributed in the interval $[-1, 1]$ but more closely spaced near the endpoints ± 1. Fig. 1.4 illustrates this.*

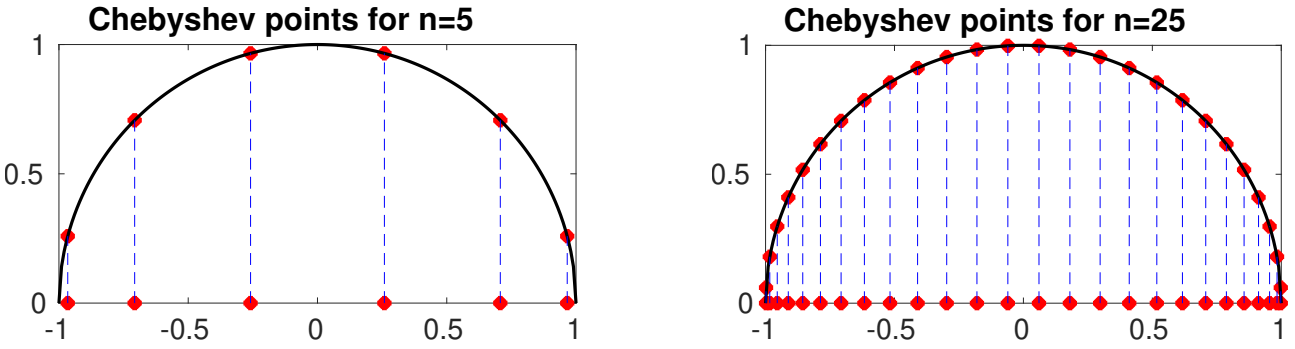


Figure 1.4: Chebyshev points $x_{i,n}^{Cheb}$, $i = 0, \dots, n$, for $n = 5$ (left) and $n = 25$ (right).

1.6.3 Error bounds for Lagrange interpolation

Question: How does the interpolation error compare to the best approximation error?

We fix the interval $[a, b] = [-1, 1]$ and denote by $I_n f$ the Lagrangian interpolation polynomial of degree n that interpolates f in some distinct knots x_i (e.g. Chebyshev points or uniformly distributed points).
By definition of Lagrangian interpolation, we can derive the following stability estimate

$$\begin{aligned}\|I_n f\|_{\infty,[-1,1]} &= \max_{x\in[-1,1]} |(I_n f)(x)| = \max_{x\in[-1,1]} |\sum_{i=0}^{n} f(x_i)\ell_i(x)| \\ &\leq \max_{i=0,\dots,n} |f(x_i)| \max_{x\in[-1,1]} \sum_{i=0}^{n} |\ell_i(x)| \leq \|f\|_{\infty,[-1,1]} \max_{x\in[-1,1]} \sum_{i=0}^{n} |\ell_i(x)|. \end{aligned} \tag{1.20}$$

Thus, the maximum of the interpolation polynomial is bounded by the maximum of the given function times an amplification factor, the so called *Lebesgue constant* Λ_n defined by

$$\Lambda_n := \max_{x\in[-1,1]} \sum_{i=0}^{n} |\ell_i(x)|. \tag{1.21}$$

The following exercise shows that – as a mapping – Lagrangian interpolation is linear and reproduces polynomials of degree n.

Exercise 1.25 *Show that:*

- *The mapping $f \mapsto I_n f$ is a linear map, i.e., for continuous functions f, g and $\lambda \in \mathbb{R}$ there holds $I_n(f + g) = (I_n f) + (I_n g)$ as well as $I_n(\lambda f) = \lambda I_n f$.*
- *$I_n f = f$ for all polynomials $f \in \mathcal{P}_n$.*

  Hint: *Uniqueness of polynomial interpolation, Theorem 1.2.*

Using these two properties, together with the stability estimate (1.20), we can now estimate the best approximation error. For arbitrary $q \in \mathcal{P}_n$, there holds

$$\begin{aligned}\|f - I_n f\|_{\infty,[-1,1]} &\overset{\text{Exer. 1.25}}{=} \|f - q - I_n(f - q)\|_{\infty,[-1,1]} \\ &\leq \|f - q\|_{\infty,[-1,1]} + \|I_n(f - q)\|_{\infty,[-1,1]} \\ &\overset{(1.20)}{\leq} \|f - q\|_{\infty,[-1,1]} + \Lambda_n \|(f - q)\|_{\infty,[-1,1]} = (1 + \Lambda_n)\|f - q\|_{\infty,[-1,1]}.\end{aligned}$$

We summarize the findings in the following theorem, which also gives bounds on the Lebesgue constants for Chebyshev and uniformly distributed interpolation (see literature for that).

Theorem 1.26 *Let I_n be Lagrangian interpolation operator for some distinct knots.*

(i) There hold the stability and quasi-best approximation:

$$\begin{aligned}\|I_n f\|_{\infty,[-1,1]} &\leq \Lambda_n \|f\|_{\infty,[-1,1]} \\ \|f - I_n f\|_{\infty,[-1,1]} &\leq (1 + \Lambda_n) \min_{q\in\mathcal{P}_n} \|f - q\|_{\infty,[-1,1]}\end{aligned}$$

(ii) For the Lebesgue constants there holds:

$$\text{Uniform points:} \qquad \Lambda_n^{unif} \sim \frac{2^n}{en\ln n} \quad \text{(for large } n\text{)}$$

$$\text{Chebyshev points:} \qquad \Lambda_n^{Cheb} \leq \frac{2}{\pi}\ln(n+1)+1$$

Remark 1.27 (Interpretation of Λ_n) *1. The factor $1+\Lambda_n$ measures how much worse the approximation of f by the interpolation is compared to the* best possible *polynomial approximation (in the norm $\|\cdot\|_{\infty,[-1,1]}$).*

The logarithmic growth of Λ_n^{Cheb} is very slow so that Chebyshev interpolation is typically very good: for example, for (the already rather high polynomial degree) $n=20$ one has $\Lambda_n^{Cheb}\approx 2.9$ and thus $1+\Lambda_{20}^{Cheb}\leq 4$.

2. Λ_n can also be understood as an amplification factor: If, instead of the exact function values $f(x_i)$, perturbed values $\widetilde{f}_i$ with $|\widetilde{f}_i - f(x_i)|\leq\delta$ are employed, then the "perturbed" interpolation polynomial $\sum_i \widetilde{f}_i\ell_i$ satisfies (Exercise!)

$$\|(\sum_{i=0}^{n}\widetilde{f}_i\ell_i) - I_nf\|_{\infty,[-1,1]} \leq \Lambda_n\delta.$$

In other words: Since Λ_n^{Cheb} of Chebyshev interpolation is moderate, perturbations or errors in the values $f(x_{i,n}^{Cheb})$ have a rather small impact on the error in the interpolating polynomial.

In general, computing best-approximation errors to given functions by polynomials of fixed degree, e.g., $\min_{q\in\mathcal{P}_n}\|f-q\|_{\infty,[-1,1]}$ exactly is very hard/impossible. Numerically, one could employ the so called *Remez algorithm* for that (see literature).
However, estimates for the best-approximation error can be easily found by inserting special polynomials such as the Taylor polynomial or the Chebyshev interpolation polynomial, which we illustrate in the following example.

Example 1.28 *Let $f(x)=\exp(x)$ on $[-1,1]$. Computing the Taylor polynomial of degree 1 around $x_0=0$ gives $T_1(x)=1+x$. Then, the function $|\exp(x)-(1+x)|$ takes its maximal value at $x=1$. Thus, we have*

$$\min_{q\in\mathcal{P}_1}\|f-q\|_{\infty,[-1,1]} \leq \|f-T_1\|_{\infty,[-1,1]} = \exp(1)-2\approx 0.7183.$$

Note that the Chebyshev points on $[-1,1]$ are given by $x_1^{Cheb}=-\frac{1}{\sqrt{2}}$ and $x_2^{Cheb}=\frac{1}{\sqrt{2}}$ and the interpolation polynomial reads as $I_1^{Cheb}f\approx 1.0854x+1.2606$, which gives a bound

$$\min_{q\in\mathcal{P}_1}\|f-q\|_{\infty,[-1,1]} \leq \|f-I_1^{Cheb}f\|_{\infty,[-1,1]}\approx 0.3723.$$

Employing the Remez algorithm provides the polynomial $q(x)=1.1752x+1.2643$ such that

$$\min_{q\in\mathcal{P}_1}\|f-q\|_{\infty,[-1,1]}\approx 0.2788.$$

Thus, the Chebyshev interpolation is reasonably close to the best-approximation error and provides a much better bound for the error as the Taylor polynomial.

Chebyshev interpolation converges very rapidly for *smooth* functions, which is the topic of the following exercise.

Exercise 1.29 *Consider the function $f(x) = (4-x^2)^{-1}$. Give an upper bound for $\min_{q\in\mathcal{P}_n} \|f-q\|_{\infty,[-1,1]}$ by selecting q as the Taylor polynomial of f about a suitable point.*
Determine the interpolating polynomials $I_n^{Cheb} f$ for $n = 1, \ldots, 10$. Plot the error semilogarithmically (`semilogy` *in* `matlab` *or* `matplotlib.pyplot.semilogy` *in* `python`*) versus n. To that end, approximate the error $\|f - I_n^{Cheb} f\|_{\infty,[-1,1]}$ by simply computing the error in 100 points that are uniformly distributed over $[-1, 1]$.*

We now come back to the Runge example from before.

Example 1.30 (Runge example, cont.) **slide 3 - Chebyshev interpolation**
Consider again $f(x) = (1 + 25x^2)^{-1}$ on the interval $[-1, 1]$. Fig. 1.5 now compares the interpolation in Chebyshev and equidistant points. Whereas Chebyshev interpolation works well, we observe failure for the interpolation in equidistant points.

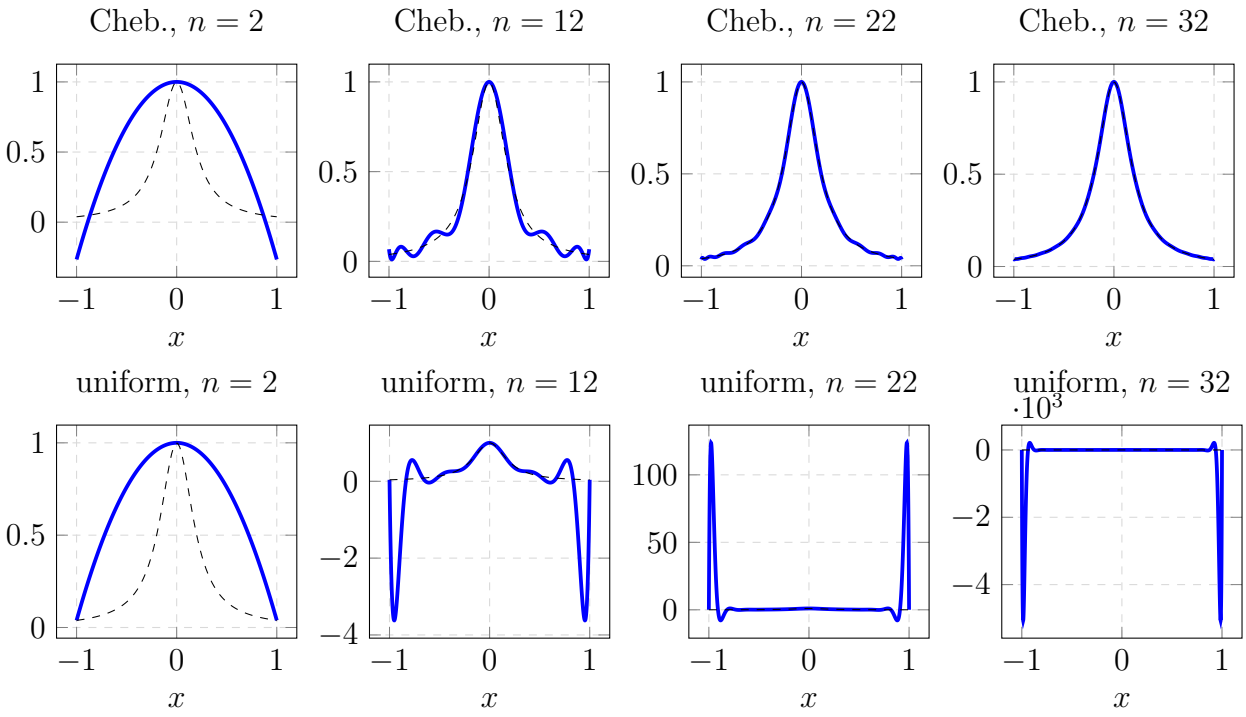


Figure 1.5: Interpolation of $(1+25x^2)^{-1}$ on $[-1, 1]$. Top row: interpolation in Chebyshev points ($n = 2$, 12, 22, 32). bottom row: Interpolation in equidistant points ($n = 2$, 12, 22, 32).

Example 1.30 shows that one should not use equidistant points for interpolation by polynomials of high degree. If the data set is based on (more or less) equidistant points, then one typically approximates by splines, i.e., *piecewise* polynomials of a *fixed* degree (e.g., $n \in \{1, 2, 3\}$) as illustrated in Exercise 1.21. An important representative of of this class is the "cubic spline" (see Section 1.8.2.)

1.7 Remarks on Hermite interpolation

A generalization of polynomial interpolation is Hermite interpolation, where not only nodal values are reproduced exactly, but also derivatives. Its most general form is as follows: Let $x_0, \ldots, x_n$ be $n+1$ distinct knots, and let $d_i \in \mathbb{N}_0$ be given for each i. Then, given values f_i^j, $i = 0, \ldots, n$, $j = 0, \ldots, d_i$, the *Hermite interpolant* is given by: Find $p \in \mathcal{P}_{n+\sum_{i=0}^n d_i}$ s.t.

$$p^{(j)}(x_i) = f_i^j, \quad i = 0, \ldots, n, \quad j = 0, \ldots, d_i. \tag{1.22}$$

Remark 1.31 *Hermite interpolation generalizes the polynomial interpolation problem (1.1): the choice $d_0 = d_1 = \cdots = d_n = 0$ reproduces (1.1). Another extreme case is $n = 0$ and $d_0 = N$. Then $p(x) = \sum_{j=0}^N \frac{f_0^j}{j!}(x - x_0)^j$. In particular, for $f_0^j = f^{(j)}(x_0)$, we obtain the Taylor polynomial of f of degree N.*

One can show that problem (1.22) is uniquely solvable. One can also show that, if $f_i^j = f^{(j)}(x_i)$ for a sufficiently smooth f, then an error bound analogous to that of Theorem 1.17 holds true.

finis 3.DS

1.8 Splines (CSE)

Question: Can we find a good localized approximation on a uniform grid?

slide 2a - Splines

Splines are *piecewise* polynomials on a partition Δ of an interval $[a, b]$.

Definition 1.32 (Spline spaces) *A* partition Δ *is described by knots* $a = x_0 < x_1 < \cdots x_n = b$. *We denote the* elements *by* $I_i = (x_i, x_{i+1})$, $i = 0, \ldots, n-1$ *and set* $h_i := x_{i+1} - x_i$. *We also set* $h := \max_i h_i$ *as the maximal element width.*

For a partition Δ *and* p *(polynomial degree),* $r \in \mathbb{N}_0$ *(regularity) the spline space* $S^{p,r}(\Delta)$ *is defined as*

$$S^{p,r}(\Delta) := \{u \in C^r([a,b]) \,|\, u|_{I_i} \in \mathcal{P}_p \quad \forall i\}. \tag{1.23}$$

Given values f_i, $i = 0, \ldots, n$, we say that $s \in S^{p,r}(\Delta)$ is an *interpolating spline*, if

$$s(x_i) = f_i, \qquad i = 0, \ldots, n. \tag{1.24}$$

Splines are widely used to fit given data or to describe curves or surfaces, e.g., in CAD systems[1].

1.8.1 Piecewise linear approximation

The simplest case is $p = 1$ and $r = 0$ is shown in Figure 1.6.

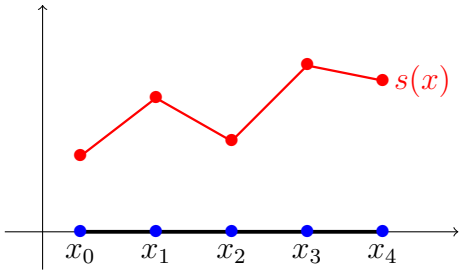


Figure 1.6: A piecewise linear spline ($p = 1$, $r = 0$).

The interpolation problem reads as: Given knots $a = x_0 < x_1 < \cdots < x_n = b$ and the corresponding partition,

$$\text{find } s \in S^{1,0}(\Delta) \text{ s.t.} \quad s(x_i) = f_i, \qquad i = 0, \ldots, n. \tag{1.25}$$

It is uniquely solvable and has as the solution

$$s(x) = \sum_{i=0}^{n} f_i \varphi_i(x),$$

where the φ_i continuous, piecewise linear functions defined by the condition $\varphi_i(x_j) = \delta_{ij}$ (Exercise: sketch the φ_i!). Concerning error estimates, one has from a generalization of Exercise 1.21

$$\|f - s\|_{\infty,[a,b]} \leq Ch^2 \|f''\|_{\infty,[a,b]}.$$

[1] *key words:* Bézier curves. Extensions of the idea of splines are NURBS (= nonuniform rational B-splines)

1.8.2 The classical cubic spline

The classical cubic spline space is given by the choices $p = 3$ and $r = 2$. The interpolation problem is:

$$\text{find } s \in S^{3,2}(\Delta) \text{ s.t.} \quad s(x_i) = f_i, \qquad i = 0, \dots, n. \tag{1.26}$$

Obviously, (1.26) represents a system of $n + 1$ equations.

Now, the question is how many free parameters (also called degrees of freedom) a function in $S^{3,2}(\Delta)$ has. We answer this more generally for the spline spaces $S^{p,r}(\Delta)$ in the following.

We count degrees of freedom needed to describe a spline: We have $\dim \mathcal{P}_p = p + 1$ so that the space of *discontinuous* piecewise polynomials of degree p is $(p + 1)n$. The condition of C^r continuity at the $n - 1$ interior knots $x_1, \dots, x_{n-1}$ imposes $(n - 1)(r + 1)$ conditions. Thus, we expect $\dim S^{p,r}(\Delta) = n(p + 1) - (n - 1)(r + 1)$, which in fact, is the statement of the following lemma.

Lemma 1.33 *Let Δ be a partition given by $n + 1$ (distinct) knots $x_0, \dots, x_n$. Then,*

$$\dim S^{p,r}(\Delta) = n(p + 1) - (n - 1)(r + 1). \tag{1.27}$$

For the case $p = 3$, $r = 2$, we get $\dim S^{3,2}(\Delta) = 4n - 3(n - 1) = n + 3$. The interpolation conditions (1.26) yield $n+1$ conditions. Hence, two more conditions have to be imposed. These two extra conditions are selected depending on the application. Typically, one of the following four choices is made:

1. *Complete/clamped spline:* The user provides two additional values f'_0, $f'_n \in \mathbb{R}$ and imposes the following two additional conditions:

$$s'(x_0) = f'_0, \qquad s'(x_n) = f'_n. \tag{1.28}$$

2. *Periodic spline:* one assumes $f_0 = f_n$ and imposes additionally

$$s'(x_0) = s'(x_n), \qquad s''(x_0) = s''(x_n). \tag{1.29}$$

3. *Natural spline:* one imposes

$$s''(x_0) = 0, \qquad s''(x_n) = 0. \tag{1.30}$$

4. *"not-a-knot condition":* one requires that the jump of s''' at the knots x_1 and x_{n-1} be zero:

$$\lim_{x \to x_1 -} s'''(x) = \lim_{x \to x_1 +} s'''(x), \qquad \lim_{x \to x_{n-1} -} s'''(x) = \lim_{x \to x_{n-1} +} s'''(x). \tag{1.31}$$

Concerning the accuracy of the interpolation method, we have:

Theorem 1.34 *Let $f \in C^4([a, b])$ and $h := \max_i h_i$. Let $f_i = f(x_i)$, $i = 0, \dots, n$. Then, the estimates*

$$\|f - s\|_{\infty,[a,b]} \le Ch^4 \|f^{(4)}\|_{\infty,[a,b]}, \qquad \|(f - s)'\|_{\infty,[a,b]} \le Ch^3 \|f^{(4)}\|_{\infty,[a,b]}$$

hold in the following cases:

(i) *s is the complete spline and $f_0' = f'(x_0)$ and $f_n' = f'(x_n)$.*

(ii) *s is the periodic spline and f is additionally periodic, i.e., $f \in C^4(\mathbb{R})$ and $f(x+(b-a)) = f(x)$ for all $x \in \mathbb{R}$.*

(iii) *s is the not-a-knot spline.*

In particular, in each of these cases, the spline interpolation problem is uniquely solvable.

Remark 1.35 *If only the values $f_i = f(x_i)$ are available and a good spline approximation to f is sought, then typically the not-a-knot interpolation is chosen. This is the default choice of the* `spline` *command in* `matlab` *and in* `scipy.interpolate.CubicSpline`*. However, both* `matlab` *and* `python` *also allow for other endpoint conditions.*

Minimization property of cubic splines

By Theorem 1.34, the cubic spline interpolation problems with any of the above 4 extra conditions is uniquely solvable. In the three cases "complete spline", "natural spline", and "periodic spline" the interpolating spline has an optimality property:

Theorem 1.36 ("energy minimization" of cubic splines) *Let $I = [a, b]$ and Δ be a partition given by $a = x_0 < x_1 < \cdots x_n = b$. Let f_i, $i = 0, \ldots, n$, be given values.*

(i) *(complete spline) Let f_0', $f_n' \in \mathbb{R}$ be additionally be given. Then, the complete spline $s \in S^{3,2}(\Delta)$ satisfies*

$$\|s''\|_{L^2(I)} \leq \|y''\|_{L^2(I)} \qquad \forall y \in \mathcal{C}_{complete},$$

where $\mathcal{C}_{complete}$ is given by

$$\mathcal{C}_{complete} = \{v \in C^2(I) \,|\, v(x_i) = f_i \text{ for } i = 0, \ldots, n \text{ and } v'(x_0) = f_0',\ v'(x_n) = f_n'\}.$$

(ii) *(natural spline) The natural spline $s \in S^{3,2}(\Delta)$ satisfies*

$$\|s''\|_{L^2(I)} \leq \|y''\|_{L^2(I)} \qquad \forall y \in \mathcal{C}_{nat},$$

where $\mathcal{C}_{nat}$ is given by

$$\mathcal{C}_{nat} = \{v \in C^2(I) \,|\, v(x_i) = f_i \text{ for } i = 0, \ldots, n \text{ and } v''(x_0) = v''(x_n) = 0\}.$$

(iii) *(periodic spline) Assume $f_0 = f_n$. Then, the periodic spline $s \in S^{3,2}(\Delta)$ satisfies*

$$\|s''\|_{L^2(I)} \leq \|y''\|_{L^2(I)} \qquad \forall y \in \mathcal{C}_{per},$$

where $\mathcal{C}_{per}$ is given by

$$\mathcal{C}_{per} = \{v \in C^2(I) \,|\, v(x_i) = f_i \text{ for } i = 0, \ldots, n \text{ and } v'(x_0) = v'(x_n) \text{ and } v''(x_0) = v''(x_n)\}.$$

Remark 1.37 *The minimization property explains the name "spline". If one studies the deflection of an elastic "spline", then the theory of linear elasticity states that the deflection is such that the spline's elastic energy is minimized. If y describes the deflection of this spline, then in good approximation, the elastic energy of a spline is given by (ignoring physical units) $\frac{1}{2}\|y''\|^2_{L^2(I)}$. Hence, if the spline is forced to pass through points (x_i, f_i), $i = 0, \ldots, n$, then the sought deflection s is the minimizer of the problem:*

$$\begin{aligned} &\textit{minimize} \quad \frac{1}{2}\|y''\|^2_{L^2(I)} \\ &\textit{under the constraint } y(x_i) = f_i,\ i = 0, \ldots, n \textit{ (plus possibly further conditions).} \end{aligned}$$

Theorem 1.36 states that the minimizer is the interpolating cubic spline, if the additional constraints are that the spline is the "complete", "natural", or "periodic" one.

Computation of the cubic spline

The computation of the interpolating spline can be reduced to the solution of a linear system of equations. In principle, one could make the ansatz that s is a cubic polynomial on each element $I_i = (x_i, x_{i+1})$. The interpolation conditions $s(x_i) = f_i$, the continuity conditions

$$\lim_{x \to x_i -} s^{(j)}(x) = \lim_{x \to x_i +} s^{(j)}(x), \qquad i = 1, \ldots, n-1, \quad j = 0, 1, 2$$

and the two additional conditions for complete/natural/periodic/not-a-knot splines describe a linear system of equations that can be solved.

1.8.3 Remarks on splines

Exercise 1.38 *Show: for $r \geq p$, one has $S^{p,r}(\Delta) = \mathcal{P}_p$ irrespective of the partition Δ.*

Remark 1.39 *For fixed, (small) r the spaces $S^{p,r}$ are much more local than the spaces $\mathcal{P}_p$. In polynomial interpolation, changing one data value f_i affects the interpolant* everywhere. *For splines (with small r), the effect is much more local, i.e., a value only affects the spline interpolant in the neighborhood of the data point. This is of interest, e.g., in the following situations:*

1. *some data values have large errors (e.g., measurement errors): then the spline is only wrong near the corresponding knot. In contrast, in polynomial interpolation, the approximation is affected everywhere.*

2. *point evaluation: if a spline is truely local (e.g., in the case $r = 0$), then the evaluation of a spline at a point x requires only the data points near x, i.e., a local calculation.*

Example 1.40 *Fig. 1.7 shows polynomial interpolation and the (complete) cubic spline interpolation of the Runge example (cf. Example 1.30) on* $[-1, 1]$. *For* $n = 8$, *the* $n+1 = 9$ *knots are uniformly distributed in* $[-1, 1]$. *We observe that, while the polynomial interpolation is rather poor, the cubic spline is very good.*

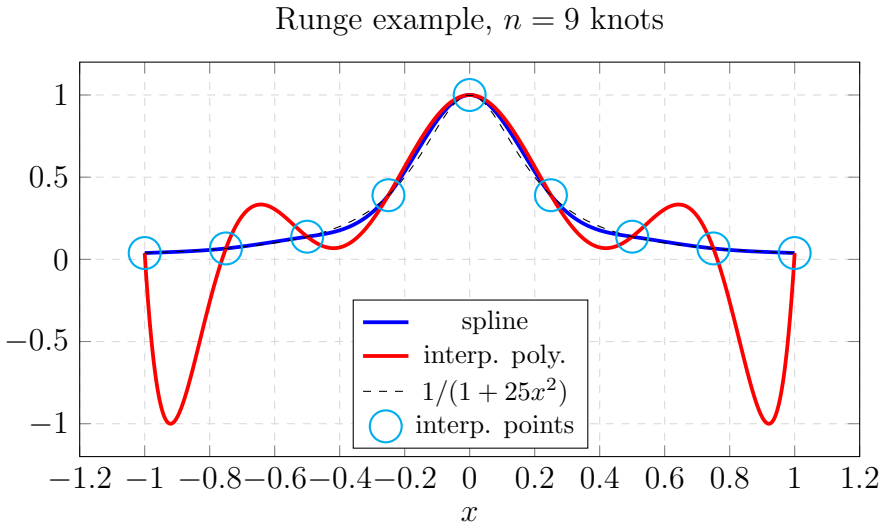


Figure 1.7: Polynomial interpolation and cubic spline interpolation for uniform knot distribution; Runge example.

1.9 Trigonometric interpolation and FFT (CSE)

convention: In this chapter, $\mathbf{i} = \sqrt{-1}$ with $\mathbf{i}^2 = -1$ (complex unit), that is, *not* an index. Numbering of indices of vectors starts at 0.

1.9.1 Trigonometric interpolation

Question: Can we use other simple functions for interpolation?

Motivation

Classical trigonometric polynomials are of the form

$$p(x) = a_0 + \sum_{j=1}^{m} a_j \cos(jx) + b_j \sin(jx). \tag{1.32}$$

A meaningful interpolation problem is: given $2m+1$ knots x_k, $k = 0, \ldots, 2m$ and values y_k find the coefficients a_j, b_j such that

$$p(x_k) = y_k, \qquad k = 0, \ldots, 2m. \tag{1.33}$$

Using the Euler formula $e^{\mathbf{i}x} = \cos x + \mathbf{i} \sin x$, one can rewrite trigonometric polynomials also in the form

$$p(x) = \sum_{j=-m}^{m} c_j e^{\mathbf{i}jx}, \quad \text{where } c_j = \tfrac{1}{2}(a_j - \mathbf{i}b_j) \text{ for } j \geq 1 \text{ and } c_j = \tfrac{1}{2}(a_j + \mathbf{i}b_j) \text{ for } j < 0,\ c_0 = a_0. \tag{1.34}$$

Hence, the interpolation problem (1.33) of finding the coefficients a_j and b_j can equivalently be posed as finding the coeffcients c_j of p in the form (1.34) such that (1.33) holds.

Remark 1.41 *(i) The trigonometric polynomial $x \mapsto p(x)$ is a 2π-periodic function. It therefore is natural to assume that the knots $x_k \in [0, 2\pi)$.*

(ii) The (continuous) Fourier transform is an important tool in signal processing, e.g., when analyzing audio signals. In the simplest setting, a signal is assumed to be periodic (over a given time interval $(0, T)$) and writing it as a Fourier series decomposes the signal into different frequency components. These components are then analyzed or modified (e.g., with low pass or high pass filters).

For $T = 2\pi$, the Fourier series is simply the representation

$$f(x) = \sum_{j=-\infty}^{\infty} f_j e^{\mathbf{i}xj}, \qquad f_j = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-\mathbf{i}xj}\, dx, \tag{1.35}$$

and f_j are the Fourier coefficients. In order to avoid evaluating the integrals, one could proceed as follows: 1) sample the signal in the points x_j; 2) approximate f by its trigonometric interpolant p; 3) interpret the Fourier coefficients of p as (good) approximations to the Fourier coefficients of f.

A simplification of the trigonometric interpolation problem

Multiplying the polynomial $p(x)$ in (1.34) by $e^{\mathbf{i}mx}$, one arrives at

$$e^{\mathbf{i}mx}p(x) = e^{\mathbf{i}mx}\sum_{j=-m}^{m} c_j e^{\mathbf{i}jx} = \sum_{j'=0}^{2m} c_{j'-m}e^{\mathbf{i}j'x}$$

so that the interpolation problem (1.33) can be rephrased as finding a trigonometric polynmomial $\widetilde{p}(x)$ of the form $\sum_{j'=0}^{2m} c_{j'-m}e^{\mathbf{i}j'x}$ such that $\widetilde{p}(x_k) = \widetilde{y}_k := y_k e^{\mathbf{i}mx_k}$.
These considerations motivate us to introduce the following definition:

Definition 1.42 *The polynomials* $p:\ \mathbb{R}\to\mathbb{C},\ p(x) = \sum_{j=0}^{n-1} c_j e^{\mathbf{i}jx},\ c_j\in\mathbb{C}$ *are called* **modified trigonometric polynomials** *of* **degree** $n-1$.

The interpolation problem now reads: given distinct knots $x_j\in[0,2\pi),\ \ j=0,\dots,n-1$ and values $y_j,\ \ j=0,\dots,n-1$ solve:

$$\text{find modified trigonometric polyomial } p \text{ of degree } n-1 \text{ s.t. } p(x_j)=y_j,\quad j=0,\dots,n-1 \tag{1.36}$$

Remark 1.43 *The interpolation problem (1.33) for a polynomial of the form (1.34) can also be solved with the same techniques as the problem (1.36). In particular, the FFT-techniques that we develop below can be applied. See Remark 1.58 below for more details.*
Reasons for introducing the modified trigonometric polynomials and interpolation problem (1.36) are mostly due to the fact that the DFT-matrix (and subsequently the FFT) take a form that is more common in the literature and can be found in the `matlab` *and* `numpy` *implementations.*

Remark 1.44 *The modified trigonometric polynomial* $x\mapsto p(x)$ *is a* 2π*-periodic function. The coefficients* c_j *are its Fourier coefficients.*

The interpolation problem (1.36) can be written as a linear system

$$\mathbf{V}\mathbf{c} = \mathbf{y}$$

with a so-called Vandermonde matrix $\mathbf{V}$, which leads to the following existence result.

Theorem 1.45 *Let* $x_j\in[0,2\pi),\ \ j=0,\dots,n-1$ *be distinct. Then, (1.36) is uniquely solvable for each sequence* $(y_j)_{j=0}^{n-1}\in\mathbb{C}^n$.

Proof: Set $z_j := e^{\mathbf{i}x_j},\ \ j=0,\dots,n-1$. Then the z_j are distinct. The ansatz $p(x) = \sum_{k=0}^{n-1} c_k\, e^{\mathbf{i}kx}$ yields the linear system of equations:

$$\underbrace{\begin{pmatrix} z_0^0 & z_0^1 & \dots & z_0^{n-1}\\ z_1^0 & z_1^1 & \dots & z_1^{n-1}\\ \vdots & \vdots & & \vdots\\ z_{n-1}^0 & z_{n-1}^1 & \dots & z_{n-1}^{n-1}\end{pmatrix}}_{=:\mathbf{V}}\underbrace{\begin{pmatrix} c_0\\ c_1\\ \vdots\\ c_{n-1}\end{pmatrix}}_{=:\mathbf{c}} = \underbrace{\begin{pmatrix} y_0\\ y_1\\ \vdots\\ y_{n-1}\end{pmatrix}}_{=:\mathbf{y}}$$

The system matrix $\mathbf{V}$ satisfies $\det\mathbf{V} = \prod_{0\le j<k\le n-1}(z_k - z_j)\neq 0$ and is therefore invertible, which implies the unique solvability of the interpolation problem. □

Properties of the system matrix

In the remainder of the chapter, we consider the uniform knot distribution

$$x_j = \frac{2\pi j}{n}, \quad j = 0, \dots, n-1. \tag{1.37}$$

In the following we introduce some useful notation.

Definition 1.46 *Let $n \in \mathbb{N}$. Define the complex* root

$$\omega_n := e^{-\frac{2\pi \mathbf{i}}{n}} \tag{1.38}$$

and the so called ***DFT matrix***

$$\mathbf{V}_n := \left(\omega_n^{j\cdot k}\right)_{j,k=0}^{n-1}. \tag{1.39}$$

Note that there holds $\omega_n^n = e^{-2\pi\mathbf{i}} = 1$ as well as $\omega_n^j = e^{-\mathbf{i}x_j}$.

The matrix $\mathbf{V}_n$ of (1.39) is easily inverted, which gives a solution to the trigonometric interpolation problem.

Theorem 1.47 *Assume (1.37). Let $\mathbf{y} := (y_0, \dots, y_{n-1})^\top \in \mathbb{C}^n$ be given, $p(x) = \sum_{j=0}^{n-1} c_j\, e^{\mathbf{i}jx}$ be the solution to (1.36) and $\mathbf{V}_n$ the DFT-matrix from* (1.39)*. Then:*

(i) $\frac{1}{n}\mathbf{V}_n\,\mathbf{y} = \mathbf{c}$ $\qquad [\![\ \textit{i.e.,}\quad c_k = \frac{1}{n}\sum_{j=0}^{n-1} \omega_n^{j\cdot k} y_j\ ]\!]$

(ii) $\frac{1}{\sqrt{n}}\mathbf{V}_n$ *symmetric and unitary* $\qquad \left(\ \textit{i.e.,}\ \left(\frac{1}{\sqrt{n}}\mathbf{V}_n\right)^{-1} = \frac{1}{\sqrt{n}}\mathbf{V}_n^H = \frac{1}{\sqrt{n}}\overline{\mathbf{V}_n}\right)$

(iii) $\overline{\mathbf{V}_n} = \left(\overline{\omega_n}^{jk}\right)_{j,k=0}^{n-1} = \left(\omega_n^{-jk}\right)_{j,k=0}^{n-1}$

Proof: ad (iii): ✓

ad (ii): Let $v_j,\ j = 0, \dots, n-1$ be the columns of $\frac{1}{\sqrt{n}}\mathbf{V}_n$. Then:

- $v_k^H v_k = \frac{1}{n}\sum_{j=0}^{n-1} \omega_n^{-jk}\omega_n^{jk} = 1$
- $k \neq l$:

$$v_k^H v_l = \frac{1}{n}\sum_{j=0}^{n-1} \omega_n^{-jk}\omega_n^{lj} = \frac{1}{n}\sum_{j=0}^{n-1} \left(\omega_n^{l-k}\right)^j \overset{\text{geometr. series}}{=} \frac{1}{n}\frac{1-\left(\omega_n^{l-k}\right)^n}{1-\omega_n^{l-k}} = \frac{1}{n}\frac{1-\left(\omega_n^n\right)^{l-k}}{1-\omega_n^{l-k}} = 0 \qquad \text{since } \omega_n^n \overset{(1.38)}{=} 1$$

ad(i): For the equidistant points $x_j,\ j = 0, \dots, n-1$, given by (1.37), the linear system of equations (1.39) has the form $\overline{\mathbf{V}_n}\mathbf{c} = \mathbf{y} \overset{(ii)}{\Rightarrow} \mathbf{c} = \overline{\mathbf{V}_n}^{-1}\mathbf{y} = \frac{1}{\sqrt{n}}\left(\frac{1}{\sqrt{n}}\overline{\mathbf{V}_n}\right)^{-1}\mathbf{y} = \frac{1}{\sqrt{n}}\frac{1}{\sqrt{n}}\mathbf{V}_n\mathbf{y} = \frac{1}{n}\mathbf{V}_n\mathbf{y}$. ❐

The DFT-matrix induces a linear mapping that is a discrete version of the continuous Fourier transform.

Definition 1.48 *The linear map*

$$\mathcal{F}_n : \mathbb{C}^n \to \mathbb{C}^n, \quad \mathbf{y} = \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix} \mapsto \mathbf{V}_n \mathbf{y}$$

is called the **discrete Fourier transform** (DFT) *of length n.*
The inverse $\mathcal{F}_n^{-1}$ is called **IDFT (inverse discrete Fourier transform)**.

Remark 1.49 *Theorem 1.47 yields*

$$\mathcal{F}_n^{-1}\mathbf{y} \;=\; \frac{1}{n}\overline{\mathbf{V}_n}\mathbf{y} \;=\; \frac{1}{n}\overline{\mathbf{V}_n\overline{\mathbf{y}}} \;=\; \frac{1}{n}\overline{\mathcal{F}_n(\overline{\mathbf{y}})}. \tag{1.40}$$

Thus, the IDFT can be realized in the same fashion as the DFT.

1.9.2 Fast Fourier transform (FFT)

observation: The matrix $\mathbf{V}_n$ is fully populated. A naive realization of the DFT therefore requires $O(n^2)$ arithmetic operation, which is inefficient.

However, the matrix $\mathbf{V}_n$ has special structure, which can be exploited. This leads to the ***Fast Fourier transform*** (FFT), which only needs $O(n \log n)$ arithmetic operations. The FFT is a prime example of a *divide and conquer algorithm.*

Let $n = 2m$ with $m \in \mathbb{N}$. We aim to reduce the application of the DFT for some vector $\mathbf{y} \in \mathbb{C}^n$, i.e., the evaluation of

$$(\mathbf{V}_n\mathbf{y})_k = \sum_{j=0}^{n-1} \omega_n^{j\cdot k} y_j =: \alpha_k,$$

to an evaluation of two DFTs for vectors of the length $m = n/2$.
For even indices $k = 2\ell$, we use $\omega_n^{n\ell} = 1$ and obtain

$$\begin{aligned} \alpha_{2\ell} \;&=\; \sum_{j=0}^{n-1} \omega_n^{2\ell j} y_j \;=\; \sum_{j=0}^{m-1} \omega_n^{2\ell j} y_j + \omega_n^{2\ell(j+\frac{n}{2})} y_{j+m} \;= \\ &=\; \sum_{j=0}^{m-1} \omega_n^{2\ell j}(y_j + \omega_n^{\ell n} y_{j+\frac{n}{2}}) = \sum_{j=0}^{m-1} \omega^{2\ell j}(y_j + y_{j+\frac{n}{2}}). \end{aligned}$$

For odd indices $k = 2\ell + 1$, we use $\omega_n^m = e^{-i\pi} = -1$ and obtain

$$\begin{aligned} \alpha_{2\ell+1} \;&=\; \sum_{j=0}^{n-1} \omega_n^{(2\ell+1)j} y_j \;=\; \sum_{j=0}^{m-1} \omega_n^{(2\ell+1)j} y_j + \omega_n^{(2\ell+1)(j+m)} y_{j+m} \;= \\ &=\; \sum_{j=0}^{m-1} \omega_n^{2\ell j} \left(\omega_n^j\, y_j + \omega_n^j \omega_n^{2\ell m} \omega_n^m\, y_{j+m}\right) \;= \\ &=\; \sum_{j=0}^{m-1} \omega_n^{2\ell j}(y_j - y_{j+m})\, \omega_n^j. \end{aligned}$$

The same calculation can also be made for the inverse DFT by changing the sign in the exponent in ω_n. We summarize everything in the following lemma.

Lemma 1.50 *Let $n = 2m$, $\omega = e^{\pm \frac{2\pi \mathbf{i}}{n}}$. Let $(y_0, \ldots, y_{n-1}) \in \mathbb{C}^n$. Then, the terms*

$$\alpha_k := \sum_{j=0}^{n-1} y_j \, \omega^{kj} \qquad k = 0, \ldots, n-1$$

can be, defining $\xi := \omega^2$, computed for $\ell = 0, \ldots, m-1$ as follows:

$$\begin{aligned} \alpha_{2\ell} &= \sum_{j=0}^{m-1} g_j \, \xi^{j\ell} \qquad \text{with} \quad g_j := y_j + y_{j+m} \\ \alpha_{2\ell+1} &= \sum_{j=0}^{m-1} h_j \xi^{j\ell} \qquad \text{with} \quad h_j := (y_j - y_{j+m}) \, \omega^j. \end{aligned}$$

Lemma 1.50 shows that provided $n = 2m$ the computation of $\hat{\mathbf{y}} = (\hat{y}_0, \ldots, \hat{y}_{n-1})^\top := \mathcal{F}_n(\mathbf{y})$ can be reduced to the computation of $\mathcal{F}_{\frac{n}{2}}(\mathbf{g})$ and $\mathcal{F}_{\frac{n}{2}}(\mathbf{h})$, where

$$\begin{aligned} (\hat{y}_0, \hat{y}_2, \ldots, \hat{y}_{n-2})^\top &= \mathcal{F}_m(\mathbf{g}) \quad , \quad \mathbf{g} = (y_j + y_{j+m})_{j=0}^{m-1} \\ (\hat{y}_1, \hat{y}_3, \ldots, \hat{y}_{n-1})^\top &= \mathcal{F}_m(\mathbf{h}) \quad , \quad \mathbf{h} = ((y_j - y_{j+m}) \, \omega_n^j)_{j=0}^{m-1}. \end{aligned}$$

If now m is again an even number, the same idea can be employed to compute $\mathcal{F}_m(\mathbf{g}), \mathcal{F}_m(\mathbf{h})$ with applications of DFTs for vectors of lengths $m/2$. Proceeding further in this fashion gives the following algorithm called **Fast Fourier Transform (FFT)**.

Algorithm 1.51 (FFT)
Input: $n = 2^p$, $p \in \mathbb{N}_0$, $\mathbf{y} = (y_0, \ldots, y_{n-1})^\top \in \mathbb{C}^n$
Output: $\hat{\mathbf{y}} = (\hat{y}_0, \ldots, \hat{y}_{n-1}) = \mathcal{F}_n(\mathbf{y})$

if $n = 1$ ***then***
  $\hat{y}_0 := y_0$
else
  $\omega := e^{\frac{-2\pi \mathbf{i}}{n}}$
  $m := \frac{n}{2}$
  $(g_j)_{j=0}^{m-1} := (y_j + y_{j+m})_{j=0}^{m-1}$
  $(h_j)_{j=0}^{m-1} := ((y_j - y_{j+m}) \, \omega^j \,)_{j=0}^{m-1}$
  $(\hat{y}_0, \hat{y}_2, \ldots, \hat{y}_{n-2}) := FFT(m, \mathbf{g})$
  $(\hat{y}_1, \hat{y}_3, \ldots, \hat{y}_{n-1}) := FFT(m, \mathbf{h})$
end if
return $(\hat{\mathbf{y}})$

By (1.40), the same idea can also be used to compute the Inverse Discrete Fourier Transform: The computation of $(\check{y}_0, \ldots, \check{y}_{n-1}) := \mathcal{F}_n^{-1}(\mathbf{y})$ reduces to computation of (cf. first equation in (1.40)):

$$(\check{y}_0, \check{y}_2, \ldots, \check{y}_{n-2})^\top = \tfrac{1}{2}\mathcal{F}_{\frac{n}{2}}^{-1}(\mathbf{g}) \quad , \quad \mathbf{g} = (y_j + y_{j+m})_{j=0}^{m-1}$$
$$(\check{y}_1, \check{y}_3, \ldots, \check{y}_{n-1})^\top = \tfrac{1}{2}\mathcal{F}_{\frac{n}{2}}^{-1}(\mathbf{h}) \quad , \quad \mathbf{h} = \big((y_j - y_{j+m})\overline{\omega_n}^j\big)_{j=0}^{m-1}$$

Algorithm 1.52 (IFFT)
Input: $n = 2^p,\ p \in \mathbb{N}_0,\ \mathbf{y} = (y_0, \ldots, y_{n-1})^\top \in \mathbb{C}^n$
Output: $\check{\mathbf{y}} = \mathcal{F}_n^{-1}(\mathbf{y})$

if $n = 1$ ***then***
  $\check{y}_0 := y_0$
else
  $\omega := e^{\frac{2\pi \mathbf{i}}{n}}$
  $m := \frac{n}{2}$
  $(g_j)_{j=0}^{m-1} := \frac{1}{2}\,(y_j + y_{j+m})_{j=0}^{m-1}$
  $(h_j)_{j=0}^{m-1} := \frac{1}{2}\,((y_j - y_{j+m})\,\omega^j\,)_{j=0}^{m-1}$
  $(\check{y}_0, \check{y}_2, \ldots, \check{y}_{n-2}) := IFFT(m, \mathbf{g})$
  $(\check{y}_1, \check{y}_3, \ldots, \check{y}_{n-1}) := IFFT(m, \mathbf{h})$
end if
return $(\hat{\mathbf{y}})$

It reamains to justify the name Fast Fourier Transform, i.e., we show that the computational cost of the FFT is significantly lower than the cost of computing the DFT directly.

Cost of the FFT: Denote by $A(n)$ the cost of the call of $FFT(n, \mathbf{y})$ and let $n = 2^p,\ p \in \mathbb{N}_0$. Then:

$$A(n) \le 2A(n/2) + \underbrace{C}_{\text{computation of } g,\, h} n \tag{1.41}$$

and thus:

$$\begin{aligned}
A(n) &\overset{(1.41)}{\le} 2\,A\left(\frac{n}{2}\right) + C\,n \;= \\
&= 2\,A\left(2^{p-1}\right) + C\,2^p \overset{(1.41)}{\le} \\
&\overset{(1.41)}{\le} 2\left(2\,A\left(2^{p-2}\right) + C\,2^{p-1}\right) + C\,2^p \;= \\
&= 2^2 A\left(2^{p-2}\right) + 2\,C\,2^p \overset{(1.41)}{\le} \\
&\overset{(1.41)}{\le} 2^2\left(2A\left(2^{p-3}\right) + C\,2^{p-2}\right) + 2\,C\,2^p \;= \\
&= 2^3 A\left(2^{p-3}\right) + 3\,C\,2^p \;\le\; \ldots \;\le \\
&\le 2^p\,A\left(2^0\right) + p\,C\,2^p \;= \\
&= n\,A(1) + (\log_2 n)\,C\,n \;\le \\
&\le n \cdot \log_2 n \cdot C' \qquad \text{mit} \quad C' = C + A(1)
\end{aligned}$$

Example 1.53 *In the following, we compare the computational times when employing the naive DFT implementation with the FFT implementation of Matlab. As a test case, we take a signal* $f(t) = 3\sin(100\pi t) + \sin(240\pi t)$ *sampled at* N *points uniformly distributed in* $[0, 1]$.
Figure 1.8 shows that the FFT indeed scales like $\mathcal{O}(N\log(N))$ *for growing* N*, while the DFT performs significantly worse.*

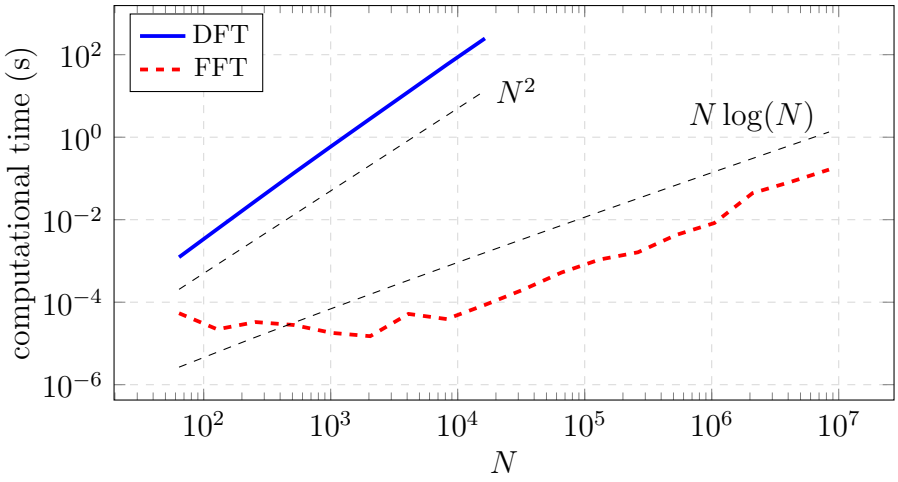


Figure 1.8: Computational times of DFT vs. FFT.

1.9.3 Properties of the DFT

The DFT appears very prominently when one is trying to compute efficiently the *convolution* of two sequences (with various applications in signal processing), which is defined in the following definition.

Definition 1.54 *(i) A sequence* $\mathfrak{f} = (f_j)_{j\in\mathbb{Z}}$ *is called* n**-periodic**, *if* $f_{j+n} = f_j \ \forall j \in \mathbb{Z}$. $\mathbb{C}^n_{per}$ *denotes the space of the* n*-periodic sequences.*

(ii) The DFT $\mathcal{F}_n$ *is defined by:*

$$\begin{aligned} \mathcal{F}_n : \quad \mathbb{C}^n_{per} &\to \mathbb{C}^n_{per} \\ (f_j)_{j\in\mathbb{Z}} &\mapsto \left(\sum_{j=0}^{n-1} \omega_n^{jk} f_j\right)_{k\in\mathbb{Z}} \end{aligned}$$

Since $\omega_n^n = 1$ *the DFT* $\mathcal{F}_n$ *is well-defined;* ⟦ *i.e.,* $\mathcal{F}_n((f_j)_{j\in\mathbb{Z}})$ *is again an* n*-periodic sequence* ⟧

(iii) the convolution $*$ *is defined by:*

$$\begin{aligned} * : \ \mathbb{C}^n_{per} \times \mathbb{C}^n_{per} &\to \mathbb{C}^n_{per} \\ (\mathfrak{f}, \mathfrak{g}) &\mapsto (\mathfrak{f} * \mathfrak{g})_k := \left(\sum_{j=0}^{n-1} f_{k-j}\, g_j\right) \qquad \forall k \in \mathbb{Z} \end{aligned}$$

(iv) the pointwise multiplication $\cdot$ is defined by:

$$\begin{array}{rccl} \cdot: & \mathbb{C}^n_{per} \times \mathbb{C}^n_{per} & \to & \mathbb{C}^n_{per} \\ & (\mathfrak{f}, \mathfrak{g}) & \mapsto & (\mathfrak{f} \cdot \mathfrak{g})_k := f_k \cdot g_k \qquad \forall k \in \mathbb{Z} \end{array}$$

Remark 1.55 *The DFT of Def. 1.48 coincides with the definition of the DFT of Def. 1.54, if one extends the finite sequence $(f_j)_{j=0}^{n-1}$ n-periodically.*

The following theorem (the proof is an exercise for the reader) motivates, why the DFT is very useful when calculating convolutions, as convolutions are turned into multiplications.

Theorem 1.56 *For $\mathfrak{f}$, $\mathfrak{g} \in \mathbb{C}^n_{per}$ let $\hat{\mathfrak{f}} := \mathcal{F}_n(\mathfrak{f}), \hat{\mathfrak{g}} := \mathcal{F}_n(\mathfrak{g})$ be the Fourier transformations. Then:*

(i) $\mathcal{F}_n: \ \mathbb{C}^n_{per} \to \mathbb{C}^n_{per}$ *is linear.*

(ii) $\mathcal{F}_n^{-1}(\mathfrak{f}) = \frac{1}{n}\left(\sum_{j=0}^{n-1} \overline{\omega_n}^{jk} f_j\right)_{k\in\mathbb{Z}}$

(iii) **(convolution theorem)**

$$\widehat{\mathfrak{f} * \mathfrak{g}} = \mathcal{F}_n(\mathfrak{f} * \mathfrak{g}) = \hat{\mathfrak{f}} \cdot \hat{\mathfrak{g}}$$

Remark 1.57 *In the context of periodic sequences, the DFT can alternatively be defined by*

$$(\mathcal{F}_n \mathfrak{f})_k := \sum_{j=-m}^{n-m-1} \omega_n^{jk} f_j \tag{1.42}$$

for any $m \in \mathbb{Z}$, i.e., it is only essential that the summation in j extends over one period but not where it starts.
The DFT of Def. 1.48 is (up to scaling) the definition employed in `matlab` *or* `numpy`*. Often in the literature, however, the DFT is defined differently from Def. 1.48, for example, as in (1.42) with $m = n/2 + 1$. In the setting of periodic sequences these definitions coincide, which corresponds to rearranging the input data if necessary.*

Remark 1.58 *With Remark 1.57 we can easily solve the interpolation problem of finding the vector $(c_j)_{j=-m}^m$ that solves the interpolation problem*

$$\sum_{j=-m}^{m} c_j e^{\mathbf{i}jx_k} = y_k, \qquad k = -m, \ldots, m.$$

(Note that we conveniently posed the interpolation problem in the points x_k, $k = -m, \ldots, m$.) In matrix-vector notation, this is

$$\mathbf{y} = \mathbf{V}\mathbf{c}, \qquad \mathbf{V} = (\omega_n^{jk})_{j,k=-m}^m,$$

where we adopted the notation to index the matrix $\mathbf{V}$ for j, $k = -m, \ldots, m$ rather than from 0 to $2m-1$. Inspection of the proof of Theorem 1.47 shows that $(2m)^{-1/2}\mathbf{V}$ is unitary and that hence

$$\mathbf{c} = \frac{1}{2m}\mathbf{V}\mathbf{y}.$$

That is:

$$c_k = \frac{1}{2m} \sum_{j=-m}^{m} \omega_n^{jk} y_j, \qquad k = -m, \ldots, m,$$

which is the same formula as for the standard DFT—only the range of the summation has changed. In view of Remark 1.57, this is the standard DFT (after suitably periodizing $\mathbf{y}$ *and* $\mathbf{c}$*). In particular, the FFT techniques are applicable.*

1.9.4 Application: fast convolution of sequence

Example 1.59 *Let* $\mathfrak{f}$, $\mathfrak{g} \in \mathbb{C}^n_{per}$*. The naive evaluation of the convolution* $\mathfrak{h} := \mathfrak{f} * \mathfrak{g}$ *costs* $O(n^2)$ *operations. It is more efficient to proceed with Theorem 1.56:*

1.) compute $\hat{\mathfrak{f}}$ *and* $\hat{\mathfrak{g}}$ *using FFT*	*cost:* $O(n \log n)$
2.) compute $\hat{\mathfrak{h}} := \hat{\mathfrak{f}} \cdot \hat{\mathfrak{g}}$	*cost:* $O(n)$
3.) compute $\mathfrak{h} = \mathcal{F}_n^{-1}\left(\hat{\mathfrak{h}}\right)$ *using IFFT*	*cost:* $O(n \log n)$

The convolution of finite (non-periodic) sequences is defined slightly differently, namely, for two finite sequences $(f_j)_{j=0}^{N-1}$, $(g_j)_{j=0}^{N-1}$, its convolution is given by the sequence $(c_j)_{j=0}^{N-1}$ with entries

$$c_j = \sum_{k=0}^{j} f_{j-k} g_k. \tag{1.43}$$

The sequence $(c_j)_{j=0}^{N-1}$ can also be computed with the aid of the FFT:

Example 1.60 *Let* $(f_j)_{j=0}^{N-1}, (g_j)_{j=0}^{N-1}$ *be finite sequences.*

goal: *compute* $(h_j)_{j=0}^{N-1}$ *given by* $h_j = \sum_{k=0}^{j} f_{j-k}\, g_k, \qquad j = 0, \ldots, N-1$

idea: *periodize the two sequences* $(f_j)_{j=0}^{N-1}$ *and* $(g_j)_{j=0}^{N-1}$*, so that Example 1.59 is applicable.*

Procedure: *Choose an* $n \geq 2N$ *of the form* $n = 2^p$ *for a* $p \in \mathbb{N}_0$ *and define* $\mathfrak{f}'$, $\mathfrak{g}' \in \mathbb{C}^n_{per}$ *by*

$$f'_j := \begin{cases} f_j & \text{for } j = 0, \ldots, N-1 \\ 0 & \text{for } j = N, \ldots, n-1 \end{cases}$$

$$g'_j := \begin{cases} g_j & \text{for } j = 0, \ldots, N-1 \\ 0 & \text{for } j = N, \ldots, n-1 \end{cases}$$

Then:

$$f'_j = 0 \quad \text{for } N - n \leq j \leq -1 \tag{1.44}$$

$$g'_j = 0 \quad \text{for } N \leq j \leq n-1 \tag{1.45}$$

For $k \in \{0, \ldots, N-1\}$ *we have:*

$$(\mathfrak{f}' * \mathfrak{g}')_k = \sum_{j=0}^{n-1} f'_{k-j}\, g'_j \overset{(1.45)}{=} \sum_{j=0}^{N-1} f'_{k-j}\, g_j = \sum_{j=0}^{k} \underbrace{f'_{k-j}}_{=f_{k-j}} g_j + \sum_{j=k+1}^{N-1} \underbrace{f'_{k-j}}_{=0 \text{ by (1.44)}} g_j = \sum_{j=0}^{k} f_{k-j}\, g_j$$

The convolution of non-periodic sequence arises, for example, when polynomials are multiplied.

Example 1.61 *Let polynomials $\pi_1(x) = \sum_{j=0}^{N-1} f_j x^j$ and $\pi_2(x) = \sum_{j=0}^{N-1} g_j x^j$ of degree $N-1$ be given. Then, the product $\pi_1\pi_2$ is a polynomial of degree $2N-2$ given by*

$$\pi_1(x)\pi_2(x) = \sum_{j=0}^{2(N-1)} h_j x^j, \qquad h_j = \sum_{k=0}^{j} f_{j-k} g_k,$$

where we implicitly assume that $f_k = g_k = 0$ for $k \in \{N, \dots, 2N-2\}$. Hence, Example 1.60 is applicable.

An application that exemplifies the use of the FFT in connection with the computation of the convolution of sequences is the multiplication of very large numbers.

Example 1.62 (multiplication of numbers with many digits) *The fast realization of the multiplication of numbers with many digits is nowadays done by FFT[2]. Consider the multiplication of two integers with n digits that are written as*

$$x = \sum_{j=0}^{n} f_j b^j, \qquad y = \sum_{j=0}^{n} g_j b^j,$$

where $b \in \mathbb{N}$ (e.g., $b = 10$) and the coefficients ("digits") satisfy $f_j,\ g_j \in \{0, \dots, b-1\}$. We seek the representation of $z = xy$ in the form $z = \sum_{j=0}^{2n} c_j b^j$ with $c_j \in \{0, \dots, b-1\}$. This is very similar to Example 1.61, and a formal multiplication yields

$$xy = \sum_{j=0}^{2n} h_j b^j, \qquad h_j = \sum_{k=0}^{j} f_{j-k} g_k,$$

where we again assumed that $f_j = 0 = g_j$ for $j \in \{n+1, \dots, 2n\}$. The sequence $(h_j)_j$ can be calculated with cost $O(n \log n)$ using the FFT as described in Example 1.60. The sought coefficients $(c_j)_j$ of z are obtained from the sequence $(h_j)_j$ by one more sweep through the sequence with cost $O(n)$ that ensures that the coefficients c_j satisfy $c_j \in \{0, \dots, b-1\}$. The following loop overwrites the h_j with the sought c_j:

for $j = 0 : 2n$ ***do***
  if $h_j \geq b$ ***then*** ▷ *carrying over is necessary*
    $h_j := h_j - \lfloor h_j/b \rfloor b$
    $h_{j+1} := h_{j+1} + \lfloor h_j/b \rfloor$
  end if
end for

[2]This is also a building block of arbitrary precision arithmetic

Example 1.63 (solving linear systems with circulant matrices) *A matrix* $\mathbf{C} \in \mathbb{C}^{n\times n}$ *is called* circulant*, if it has the form*

$$\mathbf{C} = \begin{pmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{pmatrix}.$$

Introduce the vector $\mathbf{c} := (c_0, \ldots, c_{n-1})^T$*. Observe that the matrix-vector product* $\mathbf{Cx}$ *is a convolution, i.e., the entries* $\mathbf{y}_j$ *of the vector* $\mathbf{y} = \mathbf{Cx}$ *are given by*

$$\mathbf{y}_j = \sum_{k=0}^{n-1} \mathbf{c}_{j-k}\mathbf{x}_k,$$

where we view the sequence $(c_j)_{j=0}^{n-1}$ *as an element of* $\mathbb{C}^n_{per}$ *(i.e., extend the sequence* $(c_j)_{j=0}^{n-1}$ *periodically). That is,*

$$(\mathbf{Cx})_j = (\mathbf{c} * \mathbf{x})_j, \qquad j = 0, \ldots, n-1.$$

Hence, given $\mathbf{b} \in \mathbb{C}^n$*, the linear system of equations* $\mathbf{Cx} = \mathbf{b}$ *can also be written as*

$$\mathbf{c} * \mathbf{x} = \mathbf{b}. \tag{1.46}$$

Solving for $\mathbf{x}$ *can be achieved with the FFT. To that end, write* $\widehat{\mathbf{c}} = \mathcal{F}_n(\mathbf{c})$*,* $\widehat{\mathbf{x}} = \mathcal{F}_n(\mathbf{x})$*,* $\widehat{\mathbf{b}} = \mathcal{F}_n(\mathbf{b})$ *and observe:*

1. *Applying DFT on both sides of (1.46) gives by the convolution theorem* $\widehat{c}_j\widehat{x}_j = \widehat{b}_j$*,* $j = 0, \ldots, n-1$*.*
2. *Hence,* $\widehat{x}_j = \widehat{b}_j/\widehat{c}_j$*.*
3. *an inverse DFT of* $\widehat{\mathbf{x}} = (\widehat{x}_j)_{j=0}^{n-1}$ *gives* $\mathbf{x}$*.*

Hence, the work to solve $\mathbf{Cx} = \mathbf{b}$ *is 2 FFTs of length* n *and* n *divisions.*

Example 1.64 *Circulant matrices arise in the discretization of differential equations with* periodic *boundary conditions. Consider the problem*

$$-u'' + u = f \quad \text{ on } (0,1), \qquad u(0) = u(1), \quad u'(0) = u'(1)$$

discretized by a finite difference method on the regular the grid $x_i = ih$*,* $i = 0, \ldots, N$*,* $h = 1/N$*. That is, denoting by* u_i *an approximation to* $u(x_i)$ *and replacing the differential operator by a difference quotient one arrives at the following system of equations*

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + u_i = f_i := f(x_i), \qquad i = 0, \ldots, N-1,$$

Inserting the periodicity condition, i.e., $u_N = u_0$ and $u_{-1} = u_{N-1}$ yields a linear system $\mathbf{Au} = \mathbf{f}$ with $\mathbf{A} \in \mathbb{R}^{N\times N}$ given by

$$\mathbf{A} = \frac{1}{h^2}\mathbf{A}_D + \mathbf{M}, \qquad \mathbf{A}_D = \begin{pmatrix} 2 & -1 & & & & -1 \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ -1 & & & & -1 & 2 \end{pmatrix}, \qquad \mathbf{M} = \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}$$

The matrix $\mathbf{A}$ is a circulant matrix. Hence, the linear system $\mathbf{Ax} = \mathbf{b}$ can be solved using the FFT.

2 Numerical Integration

goal: compute (approximately) $\int_a^b f(x)\,dx$.

Quadrature formulas: We consider quadrature formulas of the form

$$\int_a^b f(x)\,dx \approx Q_a^b(f) = \sum_{i=0}^{n} w_i f(x_i) \tag{2.1}$$

The points x_i are called *quadrature points*, the numbers w_i *quadrature weights*.

Example 2.1 *Partition $[a,b]$ in N subintervals $[t_i, t_{i+1}]$, $i = 0,\dots,N-1$ with $t_i = a + ih$, $h = (b-a)/N$. Let $m_i := (t_i + t_{i+1})/2$ be the midpoints. Then the* composite midpoint rule *is*

$$\int_a^b f(x)\,dx \approx Q_a^b(f) = \sum_{i=0}^{N-1} h f(m_i).$$

Example 2.2 *The* (composite) trapezoidal *rule is given, with the notation of Example 2.1, by*

$$\int_a^b f(x)\,dx \approx Q_a^b(f) = \sum_{i=0}^{N-1} h\frac{1}{2}\left[f(t_i) + f(t_{i+1})\right] = h\left[\frac{1}{2}f(a) + \sum_{i=1}^{N-1} f(t_i) + \frac{1}{2}f(b)\right].$$

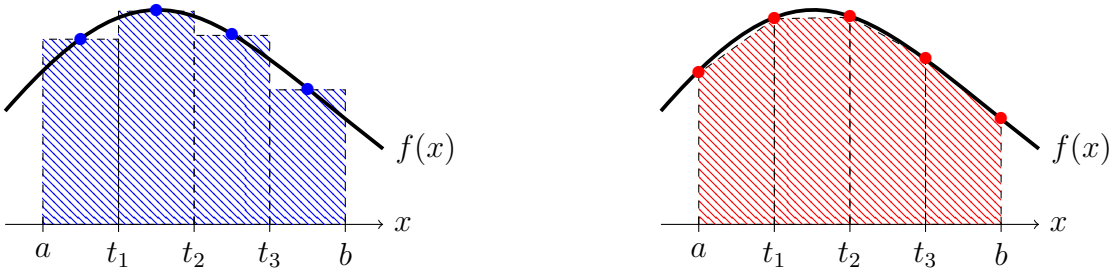


Figure 2.1: Left: composite midpoint rule, right: composite trapezoidal rule.

The Examples 2.1, 2.2 are typical representatives for the way composite quadrature rules are generated:

1. define a quadrature rule $\widehat{Q}(f) \approx \int_0^1 f(x)\,dx$ on a reference interval, e.g., $[0,1]$.
2. Partition the interval $[a,b]$ in subintervals (t_i, t_{i+1}) of lengths $h_i = t_{i+1} - t_i$.
3. The observation $\int_{t_i}^{t_{i+1}} f(x)\,dx = h_i \int_0^1 f(t_i + h_i\xi)\,d\xi$ motivates the definition

$$\int_a^b f(x)\,dx = \sum_{i=0}^{N-1}\int_{t_i}^{t_{i+1}} f(x)\,dx = \sum_{i=0}^{N-1} h_i \int_0^1 f(t_i + h_i\xi)\,d\xi \approx \sum_{i=0}^{N-1} h_i \widehat{Q}(f(t_i + h_i\cdot))$$

Remark 2.3 *Quadrature rules are normally formulated for a reference interval, which is typically $[0,1]$ or $[-1,1]$. For a general interval $[a,b]$, the rule is obtained by an affine change of variables (as done above).*

2.1 Newton-Cotes formulas

The Newton-Cotes formulas for the integration on $[0,1]$ are examples of *interpolatory* quadrature formulas. They are based on interpolating the integrand f and integrating the intepolating polynomial. The interpolation points are uniformly distributed over $[0,1]$.

Example 2.4 (closed Newton-Cotes formulas) *Let $n \geq 1$ and $x_i = \frac{i}{n}$, $i = 0, \ldots, n$. The interpolating polynomial $p \in \mathcal{P}_n$ is*

$$p(x) = \sum_{i=0}^{n} f(x_i)\ell_i(x), \qquad \ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

Hence, the quadrature fromula is

$$\int_0^1 f(x)\,dx \approx \int_0^1 p(x)\,dx = \int_0^1 \sum_{i=0}^{n} f(x_i)\ell_i(x)\,dx = \sum_{i=0}^{n} f(x_i) \underbrace{\int_0^1 \ell_i(x)\,dx}_{=:w_i} =: \widehat{Q}_n^{cNC}(f)$$

with the quadrature weights w_i, $i = 0, \ldots, n$, which are explicitly given in Fig. 2.2.

slide 4 - Newton-Cotes formulas

The endpoints of the interval are quadrature points for the "closed" formulas of Example 2.4. If, for example, integrands are not defined at an endpoint (e.g., $1/\sqrt{x}$, $\log x$), then it is more convenient to have formulas that do not sample the integrand at the endpoint. Hence, another very important class of Newton-Cotes formulas are the "open" formulas:

Example 2.5 (open Newton-Cotes-Formeln) *Let $n \geq 0$ and $x_i = \frac{2i+1}{2n+2}$, $i = 0, \ldots, n$. Then the quadrature is given by*

$$\int_0^1 f(x)\,dx \approx \sum_{i=0}^{n} f(x_i) \underbrace{\int_0^1 \ell_i(x)\,dx}_{=:w_i} =: \widehat{Q}_n^{oNC}(f), \qquad \ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

The choice $n = 0$ corresponds to the midpoint rule

$$\int_0^1 f(x)\,dx \approx Q^{mid}(f) = f(1/2).$$

By construction the Newton-Cotes formulas are exact for polynomials $f \in \mathcal{P}_n$ (as the interpolation reproduces polynomials). In fact, one can show that, if n is even, then both the closed and the open Newton-Cotes formulas are exact for polynomials $f \in \mathcal{P}_{n+1}$.

Example 2.6 *The midpoint rule integrates all linear polynomials exactly, as for arbitrary $p(x) = \alpha x + \beta$ there holds*

$$\int_0^1 p(x)\,dx = \frac{\alpha}{2} + \beta = p(1/2) = Q^{mid}(p).$$

n	weight	$Q(f) - \int_0^1 f(x)\,dx$	name
1	$\frac{1}{2}$ $\frac{1}{2}$	$\frac{1}{12}h^3 f^{(2)}(\xi)$	trapezoidal rule
2	$\frac{1}{6}$ $\frac{4}{6}$ $\frac{1}{6}$	$\frac{1}{90}h^5 f^{(4)}(\xi)$	Simpson rule
3	$\frac{1}{8}$ $\frac{3}{8}$ $\frac{3}{8}$ $\frac{1}{8}$	$\frac{3}{80}h^5 f^{(4)}(\xi)$	3/8 rule
4	$\frac{7}{90}$ $\frac{32}{90}$ $\frac{12}{90}$ $\frac{32}{90}$ $\frac{7}{90}$	$\frac{8}{945}h^7 f^{(6)}(\xi)$	Milne rule
5	$\frac{19}{288}$ $\frac{75}{288}$ $\frac{50}{288}$ $\frac{50}{288}$ $\frac{75}{288}$ $\frac{19}{288}$	$\frac{275}{12096}h^7 f^{(6)}(\xi)$	—
6	$\frac{41}{840}$ $\frac{216}{840}$ $\frac{27}{840}$ $\frac{272}{840}$ $\frac{27}{840}$ $\frac{216}{840}$ $\frac{41}{840}$	$\frac{9}{1400}h^9 f^{(8)}(\xi)$	Weddle rule

Figure 2.2: the closed Newton-Cotes formulas for the integration over $[0, 1]$. Quadrature points are $x_i = \frac{i}{n}$, $i = 0, \ldots, n$; $h = \frac{1}{n}$.

Exercise 2.7 *1. Show for the quadrature weights: $\sum_{i=0}^{n} w_i = 1 (=$ length of the interval $[0, 1])$ (*hint: *apply the quadrature formula to a suitable function f.)*

2. *Show that the quadrature formulas $\widehat{Q}_n^{cNC}$, $\widehat{Q}_n^{oNC}$ are exact for $f \in \mathcal{P}_n$.*

3. *Show the symmetry property $w_{n-i} = w_i$, $i = 0, \ldots, n$. (*hint: *Use the symmetry of the points with respect to $1/2$. The symmetry of the weights is visible in Fig. 2.2.).*

4. *Let $n = 2m$ be even. Consider the function $f = (x - 1/2)^{n+1}$, which is odd with respect to $1/2$. Show: $\int_0^1 f(x)\,dx = 0 = \widehat{Q}_n^{cNC}(f) = \widehat{Q}_n^{oNC}(f)$. Conclude that the quadrature formula is exact for polynomials of degree $n+1$. In particular, the midpoint rule is exact for polynomials in $\mathcal{P}_1$, and the Simpson rule is exact for polynomials in $\mathcal{P}_3$.*

The Newton-Cotes formulas are typically used for fixed n in composite rule. We illustrate the convergence behavior for two important cases, the composite trapezoidal rule and the composite Simpson rule. Let $a = x_0 < x_1 < \ldots < x_N = b$ be a partition of $[a, b]$ and $h_i := x_{i+1} - x_i$. We introduce the following notation for the composite trapezoidal rule

$$T_{\{x_0,\ldots,x_N\}}(f) := \sum_{i=0}^{N-1} h_i \frac{1}{2} \left(f(x_i) + f(x_{i+1})\right),$$

such that, we can easily write $T_{\{x_i,x_{i+1}\}}(f) = h_i \frac{1}{2}(f(x_i) + f(x_{i+1}))$ for the trapezoidal rule on the interval $[x_i, x_{i+1}]$.

We now aim to derive an estimate for the error between the the true integral $\int_a^b f$ and the trapezoidal rule.

- As the rule is exact for polynomials of degree $n = 1$, we can insert an arbitrary $p \in \mathcal{P}_1$ as follows:

$$\begin{aligned}\int_{x_i}^{x_{i+1}} f(x)\,dx - T_{\{x_i,x_{i+1}\}}(f) &= \int_{x_i}^{x_{i+1}} f(x) - p(x)\,dx + \int_{x_i}^{x_{i+1}} p(x)\,dx - T_{\{x_i,x_{i+1}\}}(f)\\ &= \int_{x_i}^{x_{i+1}} f(x) - p(x)\,dx + T_{\{x_i,x_{i+1}\}}(p) - T_{\{x_i,x_{i+1}\}}(f)\\ &= \int_{x_i}^{x_{i+1}} f(x) - p(x)\,dx - T_{\{x_i,x_{i+1}\}}(f-p).\end{aligned}$$

  Therefore,

$$\begin{aligned}\left|\int_{x_i}^{x_{i+1}} f(x)\,dx - T_{\{x_i,x_{i+1}\}}(f)\right| &\le (x_{i+1}-x_i)\|f-p\|_{\infty,[x_i,x_{i+1}]} + |T_{\{x_i,x_{i+1}\}}(f-p)|\\ &\le (x_{i+1}-x_i)\|f-p\|_{\infty,[x_i,x_{i+1}]} + (x_{i+1}-x_i)\|f-p\|_{\infty,[x_i,x_{i+1}]}\\ &\le 2h_i\|f-p\|_{\infty,[x_i,x_{i+1}]}.\end{aligned}$$

  Now, summing up over all sub-intervals gives

$$\left|\int_a^b f(x) - T_{\{x_0,\dots,x_N\}}(f)\right| = \left|\sum_{i=0}^{N-1}\int_{x_i}^{x_{i+1}} f(x)\,dx - T_{\{x_i,x_{i+1}\}}(f)\right| \le \sum_{i=0}^{N-1} 2h_i \min_{p\in\mathcal{P}_1}\|f-p\|_{\infty,[x_i,x_{i+1}]}$$

  which bounds the error by a polynomial best-approximation error.

- In order to provide a bound for this best-approximation term, we select for p the linear interpolant of f with Chebyshev knots in $[x_i, x_{i+1}]$. From the error bound of Theorem 1.17 together with $\|\omega_2^{Cheb}\|_{\infty,[x_i,x_{i+1}]} = \frac{(x_{i+1}-x_i)^2}{8}$ from Theorem 1.23 we obtain

$$\min_{v\in\mathcal{P}_1}\|f-p\|_{\infty,[x_i,x_{i+1}]} \le \frac{1}{16}(x_{i+1}-x_i)^2\|f''\|_{\infty,[x_i,x_{i+1}]},$$

  from which we arrive at

$$\left|\int_a^b f(x) - T_{\{x_0,\dots,x_N\}}(f)\right| \le \frac{1}{8}\sum_{i=0}^{N-1} h_i^3\|f''\|_{\infty,[x_i,x_{i+1}]}.$$

  With $h_i \le h := \max h_i$ we finally get

$$\begin{aligned}\left|\int_a^b f(x) - T_{\{x_0,\dots,x_N\}}(f)\right| &\le \frac{1}{8}\sum_{i=0}^{N-1} h_i^3\|f''\|_{\infty,[x_i,x_{i+1}]} \le \frac{1}{8}h^2\sum_{i=0}^{N-1} h_i\|f''\|_{\infty,[x_i,x_{i+1}]}\\ &\le \frac{1}{8}h^2\|f''\|_{\infty,[a,b]}\sum_{i=0}^{N-1} h_i = \frac{1}{8}h^2\|f''\|_{\infty,[a,b]}(b-a).\end{aligned}$$

We summarize the findings in the following, which also provides a similar estimate for the composite Simpson rules defined by

$$S_{\{x_0,\dots,x_N\}}(f) := \sum_{i=0}^{N-1} h_i\frac{1}{6}\left(f(x_i) + 4f(\frac{x_i+x_{i+1}}{2}) + f(x_{i+1})\right).$$

Theorem 2.8 *(i) Let $f \in C([a,b])$. Then:*

$$\left| \int_a^b f(x)\,dx - T_{\{x_0,\dots,x_N\}}(f) \right| \le 2 \sum_{i=0}^{N-1} h_i \min_{p\in\mathcal{P}_1} \|f-p\|_{\infty,[x_i,x_{i+1}]},$$

$$\left| \int_a^b f(x)\,dx - S_{\{x_0,\dots,x_N\}}(f) \right| \le 2 \sum_{i=0}^{N-1} h_i \min_{p\in\mathcal{P}_3} \|f-p\|_{\infty,[x_i,x_{i+1}]}.$$

(ii) Let $f \in C^2([a,b])$. Then for $h := \max_{i=0,\dots,N-1} h_i$

$$\left| \int_a^b f(x)\,dx - T_{\{x_0,\dots,x_N\}}(f) \right| \le \frac{1}{8} \sum_{i=0}^{N-1} h_i^3 \|f^{(2)}\|_{\infty,[x_i,x_{i+1}]} \le \frac{1}{8}(b-a)h^2 \|f^{(2)}\|_{\infty,[a,b]}$$

(iii) Let $f \in C^4([a,b])$. Then for $h := \max_{i=0,\dots,N-1} h_i$ with a constant $C > 0$

$$\left| \int_a^b f(x)\,dx - S_{\{x_0,\dots,x_N\}}(f) \right| \le C \sum_{i=0}^{N-1} h_i^5 \|f^{(4)}\|_{\infty,[x_i,x_{i+1}]} \le C(b-a)h^4 \|f^{(4)}\|_{\infty,[a,b]}$$

We say that a quadrature rule has *order* m if the the composite rule leads to error bounds of the form Ch^m (for sufficiently smooth f). The composite trapezoidal rule has therefore order $m = 2$, the composite Simpson rule order $m = 4$. More generally, the arguments leading to Theorem 2.8 show that a Newton-Cotes formula (or, more generally, any composite rule) that is exact for polynomials of degree n leads to a composite rule of order $n + 1$.

Example 2.9 **slide 5 - Composite Newton-Cotes formulas**
We compare the composite trapezoidal rule with the composite Simpson rule for integration on $[0,1]$. We partition $[0,1]$ in N subintervals of length $h = 1/N$. By Theorem 2.8 the errors E_{trap}, $E_{Simpson}$ satisfy (F denotes the number of function evaluations):

$$E_{trap}(h) \le Ch^2 \sim CF^{-2}, \qquad E_{Simpson} \le Ch^4 \sim CF^{-4}.$$

We show in Fig. 2.3 the error versus the number of function evaluations F, since this is a reasonable cost measure of the method. We note that methods of a higher order are more efficient than lower order methods.

The $O(h^2)$ convergence behavior of the composite trapezoidal rule and the $O(h^4)$ behavior of the compositive Simpson rule require $f \in C^2$ and $f \in C^4$, respectively:

Example 2.10 *Integration of $f(x) = x^{0.1}$ on $[0,1]$ does* not *yield $O(h^2)$ but merely $O(h^{1.1})$ as is visible in Figure 2.3 (right). Note that the function f is integrable and continuous, but the derivative $f'(x) = 0.1x^{-0.9}$ is not continuous at $x = 0$.*

2.2 Romberg extrapolation

Extrapolation can be used to accelerate convergence of composite rules for smooth integrands. We illustrate the procedure for the composite trapezoidal rule. For that, let the interval $[a,b]$ be partitioned in N subintervals (x_i, x_{i+1}) of length $h = (b-a)/N$ with $x_i = a + ih$. Define

$$T(h) := h \sum_{i=0}^{N-1} \frac{1}{2} \left(f(x_i) + f(x_{i+1}) \right)$$

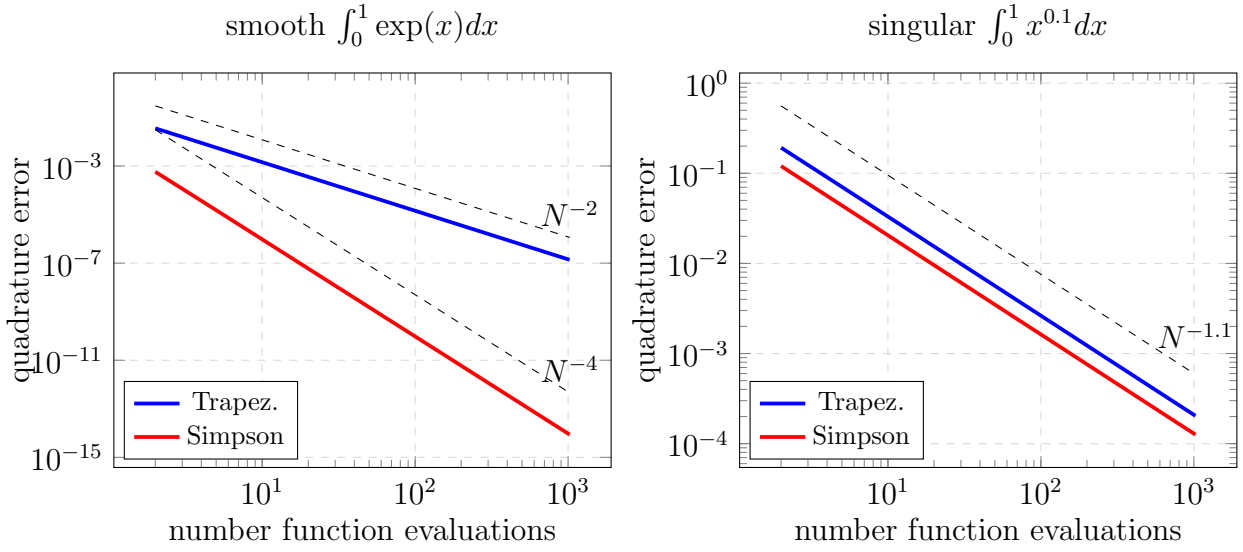


Figure 2.3: Convergence of composite trapezoidal and Simpson rule for smooth integrand $f(x) = \exp(x)$ (left) and singular integrand $f(x) = x^{0.1}$ (right).

The sought value of the integral $\int_a^b f(x)\,dx = \lim_{h\to 0} T(h)$, so that one may use extrapolation for the data $(h_i, T(h_i))$, $i = 0, 1, \ldots$, with $h_i = (b-a)M^{-i}$ for some chosen $M \in \mathbb{N}$, $M \geq 2$.[1]

In fact, $T(h)$ has "additional structure": There holds the *Euler McLaurin formula*

$$T(h) = \int_a^b f(x)\,dx + c_1 h^2 + c_2 h^4 + c_3 h^6 + \cdots, \tag{2.2}$$

where the coefficients c_i depend on higher derivatives of f. This means that $T(h)$ is actually a function depending only on h^2, i.e.,

$$T(h) = \widetilde{T}(h^2).$$

slide 6 - Euler-McLaurin formula, Romberg extrapolation

Therefore, one can obtain an approximation to $\int_a^b f(x)\,dx = \lim_{h\to 0} T(h)$ in two ways:

1. Interpolate the data $(h_i, T(h_i))$, $i = 0, \ldots, n$, and evaluate the interpolating polynomial at $h = 0$.

2. Interpolate the data $(h_i^2, T(h_i)) = (h_i^2, \widetilde{T}(h_i^2))$, $i = 0, \ldots, n$, and evaluate the interpolating polynomial at $h^2 = 0$.

Effectively, the first approach interpolates the function T, whereas the second approach interpolates the function $\widetilde{T}$. In practice, the interpolation of $\widetilde{T}$ is again realized with a Neville scheme and yields a much better accuracy for the same computational cost for smooth functions.

[1] strictly speaking, $T(h)$ is only defined for h of the form $h = (b-a)/N$, $N \in \mathbb{N}$, so that one should write $\int_a^b f(x)\,dx = \lim_{N\to\infty} T(h(N))$.

Remark 2.11 *Extrapolation of the composite trapezoidal rule for $M = 2$ yields in the first columns of the Neville scheme the composite Simpson rule; in the second column, the composite Milne rule arises. The choice $M = 3$ produces in the first column of the Neville scheme the composite 3/8-rule.*

2.3 Non-smooth integrands and adaptivity

Example 2.10 shows that, for non-smooth integrands, composite quadrature rules based on equidistant partitions $x_0 < x_1 < \cdots < x_N$ do not work very well. Our goal is to choose the partition in such a way that the composite trapezoidal rule yields convergence $O(N^{-2})$, where N is the number of quadrature points. In other words: the convergence (error vs. number of function evaluations) is similar to the case of smooth integrands.
This can be achieved for quite a few integrands f if the partition is suitably adapted to f. Basically, one should use small interval lengths h_i where f is large (in absolute value) or varies rapidly (i.e., higher derivatives of f are large):

Example 2.12 **slide 7 - Adaptive quadrature**

Consider the composite trapezoidal rule for $\int_0^1 f(x)\,dx$ mit $f(x) = x^{0.1}$ for two partitions of $0 = x_0 < x_1 \cdots < x_N = 1$ of the form

1. *equidistant points: $x_i = (i/N)$, $i = 0, \ldots, N$*
2. *points refined towards $x = 0$: $x_i = (i/N)^\beta$, $i = 0, \ldots, N$ mit $\beta = 2$*

The convergence behavior of the composite trapezoidal rule is shown in Fig. 2.4. While the convergence is only $O(N^{-1.1})$ for the equidistant points, it is $O(N^{-2})$ for the one where the points are refined towards $x = 0$.

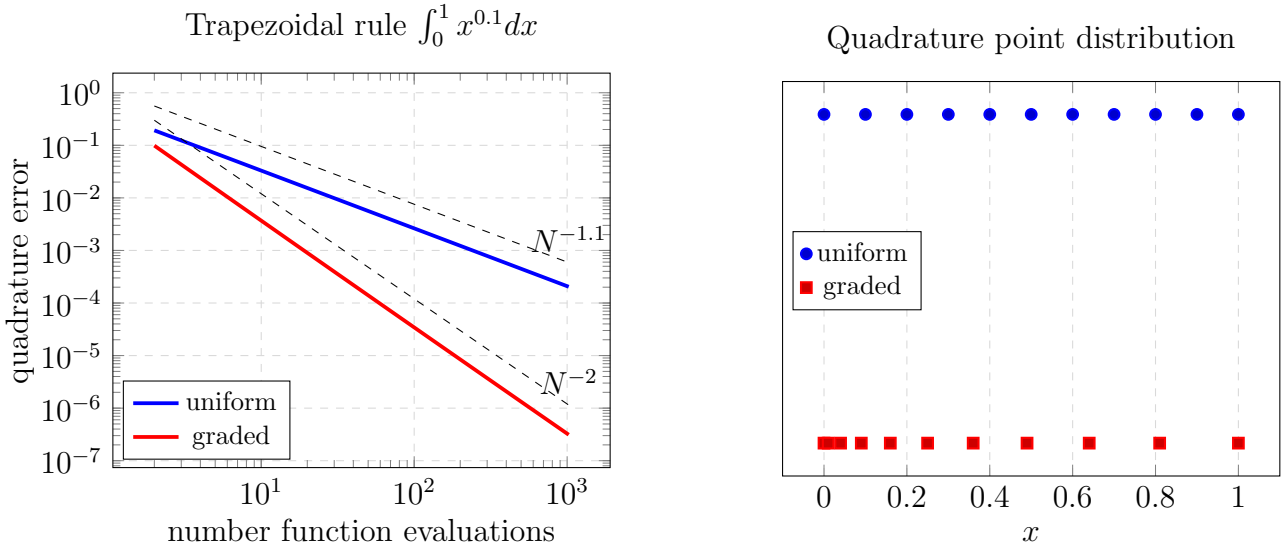


Figure 2.4: (cf. Example 2.12) numerical integration of $f(x) = x^{0.1}$ using composite trapezoidal rule based on a) equidistant nodes and b) nodes suitable refined towards $x = 0$.

In practice, it is difficult to construct a good partition for a given integrand. One is therefore interested in *adaptive algorithms*. Structurally, these algorithms proceed as outlined in Algorithm 2.13: the accuracy of an approximation for the integration on an interval $[a, b]$ (here: using the trapezoidal rule) is estimated with a better rule (here: Simpson rule). If the estimate accuracy does not meet the desired tolerance, then the interval $[a, b]$ is subdivided into two subintervals $[a, m]$, $[m, b]$ with midpoint $m = (a+b)/2$ and the quadrature routine is recursively called for the two subintervals.

Algorithm 2.13 (adaptive algorithm based on trapezoidal rule)

adapt*(f,a,b,τ)*
% approximates $\int_a^b f(x)\,dx$ *to given accuracy* τ
% h_{min} *= minimal interval length ;* $\rho \in (0,1)$ *safety factor*
% $T([a,b])$ *= trapezoidal rule für* $[a,b]$*;* $S([a,b])$ *= Simpson rule for* $[a,b]$

if $(b-a) \le h_{min}$ **return** $S([a,b])$ *%forced termination!*
if $|S([a,b]) - T([a,b])| \le \rho\tau$ *% desired accuracy reached*
    return *S([a,b])*
else
% desired accuracy not reached →subdivide $[a,b]$ *into* $[a,m]$ *and* $[m,b]$
    $m := (a+b)/2$
    $I :=$ **adapt**$(f,a,m,\tau/2) +$ **adapt**$(f,m,b,\tau/2)$
    return I

2.4 Gaussian quadrature

Question: How to choose $n+1$ quadrature points so that polynomials of the highest possible degree are integrated exactly?

Answer: Gaussian quadrature integrates polynomials of degree $2n+1$ exactly. The $n+1$ quadrature points ("Gaussian points") of this quadrature rule are the zeros of the Legendre polynomial L_{n+1}.

2.4.1 Legendre polynomials L_n as orthogonal polynomials

We consider the interval $[-1, 1]$. On the space $C([-1,1])$ we define a scalar product by

$$\langle u, v\rangle := \int_{-1}^{1} u(x)v(x)\,dx. \tag{2.3}$$

We seek a sequence of polynomials $L_n \in \mathcal{P}_n$, $n = 0,\ 1, \ldots,$, with the following properties:

(i) $\{L_0, \ldots, L_n\}$ is a basis of $\mathcal{P}_n$ (for each n)

(ii) L_n is orthogonal to the space $\mathcal{P}_{n-1}$, i.e,.

$$\langle L_n, v\rangle = 0 \qquad \forall v \in \mathcal{P}_{n-1}. \tag{2.4}$$

Such polynomials can be constructed inductively with a variant of the "Gram-Schmidt-orthogonalization": We choose[2],

$$\widetilde{L}_0(x) := 1, \qquad \widetilde{L}_1(x) := x.$$

We note that

$$\langle \widetilde{L}_1, \widetilde{L}_0\rangle = 0, \tag{2.5}$$

so that (2.4) is satisfied for $n = 1$.
For $\widetilde{L}_2 \in \mathcal{P}_2$ we make the ansatz

$$\widetilde{L}_2(x) = x\widetilde{L}_1(x) + r_1$$

for a polynomial $r_1 \in \mathcal{P}_1$ to be determined. Writing $r_1 = a_0\widetilde{L}_0 + a_1\widetilde{L}_1$ the orthogonality conditions (2.4) imply the two equations

$$\begin{aligned} 0 \stackrel{!}{=} \langle \widetilde{L}_2, \widetilde{L}_0\rangle &= \langle x\widetilde{L}_1(x), \widetilde{L}_0(x)\rangle + a_0 \underbrace{\langle \widetilde{L}_0, \widetilde{L}_0\rangle}_{>0} + a_1 \underbrace{\langle \widetilde{L}_1, \widetilde{L}_0\rangle}_{=0 \text{ b/c of (2.5)}}, \\ 0 \stackrel{!}{=} \langle \widetilde{L}_2, \widetilde{L}_1\rangle &= \langle x\widetilde{L}_1(x), \widetilde{L}_1\rangle + a_0 \underbrace{\langle \widetilde{L}_0, \widetilde{L}_1\rangle}_{=0 \text{ b/c of (2.5)}} + a_1 \underbrace{\langle \widetilde{L}_1, \widetilde{L}_1\rangle}_{>0}. \end{aligned}$$

for the coefficients a_0, a_1. This system of equations can obviously be solved and therefore $\widetilde{L}_2$ is determined. By construction, we have (2.4) for $n \leq 2$.
Inductively, we make for $\widetilde{L}_3$ the ansatz $\widetilde{L}_3(x) = x\widetilde{L}_2(x) + r_2(x)$ for an $r_2 \in \mathcal{P}_2$. Again, (2.4) yields after writing $r_2(x) = \sum_{i=0}^{2} a_i\widetilde{L}_i(x)$ a linear system of equations for the a_i:

$$\begin{aligned} 0 \stackrel{!}{=} \langle \widetilde{L}_3, \widetilde{L}_0\rangle &= \langle x\widetilde{L}_2(x), \widetilde{L}_0\rangle + \sum_{j=0}^{2} a_j\langle \widetilde{L}_j, \widetilde{L}_0\rangle, \\ 0 \stackrel{!}{=} \langle \widetilde{L}_3, \widetilde{L}_1\rangle &= \langle x\widetilde{L}_2(x), \widetilde{L}_1\rangle + \sum_{j=0}^{2} a_j\langle \widetilde{L}_j, \widetilde{L}_1\rangle, \\ 0 \stackrel{!}{=} \langle \widetilde{L}_3, \widetilde{L}_2\rangle &= \langle x\widetilde{L}_2(x), \widetilde{L}_2\rangle + \sum_{j=0}^{2} a_j\langle \widetilde{L}_j, \widetilde{L}_2\rangle. \end{aligned}$$

Again, since we already know (2.4) for $n \leq 2$, the system of equations simplifies to

$$0 \stackrel{!}{=} (\widetilde{L}_3, \widetilde{L}_i) = \langle x\widetilde{L}_2, \widetilde{L}_i\rangle + a_i \underbrace{\langle \widetilde{L}_i, \widetilde{L}_i\rangle}_{>0}, \qquad i = 0, 1, 2.$$

[2]since the "classical" Legendre polynomials L_n are scaled slightly differently (see below), we employ the notation $\widetilde{L}_n$

This yields the coefficients a_i and therefore $\widetilde{L}_3$. In this way, we can construct inductively the polynomials $\widetilde{L}_n \in \mathcal{P}_n$, $n = 0, 1, \ldots$. Our procedure yields the representation

$$\widetilde{L}_{n+1}(x) = x\widetilde{L}_n(x) - \sum_{i=0}^{n} \frac{1}{\langle \widetilde{L}_i, \widetilde{L}_i \rangle} \langle x\widetilde{L}_n, \widetilde{L}_i \rangle \widetilde{L}_i(x)$$

This can be simplified furthermore with the aid of (2.4):

$$\langle x\widetilde{L}_n(x), \widetilde{L}_i(x) \rangle = \langle \widetilde{L}_n(x), x\widetilde{L}_i(x) \rangle \overset{(2.4)}{=} 0 \quad \text{für } i+1 \leq n-1, \tag{2.6}$$

Hence, we arrive at the so-called "3-term recurrence relation"

$$\begin{aligned} \widetilde{L}_{n+1}(x) &= x\widetilde{L}_n(x) - \sum_{i=0}^{n} \frac{1}{\langle \widetilde{L}_i, \widetilde{L}_i \rangle} \langle x\widetilde{L}_n(x), \widetilde{L}_i(x) \rangle \widetilde{L}_i(x) \\ &\overset{(2.6)}{=} x\widetilde{L}_n(x) - \sum_{i=n-1}^{n} \frac{1}{\langle \widetilde{L}_i, \widetilde{L}_i \rangle} \langle x\widetilde{L}_n(x), \widetilde{L}_i(x) \rangle \widetilde{L}_i(x) \\ &= x\widetilde{L}_n(x) - \widetilde{a}_n \widetilde{L}_n(x) - \widetilde{b}_n \widetilde{L}_{n-1}(x) = \boxed{(x - \widetilde{a}_n)\widetilde{L}_n(x) - \widetilde{b}_n \widetilde{L}_{n-1}(x),} \end{aligned} \tag{2.7}$$

with

$$\widetilde{a}_n = \frac{\langle x\widetilde{L}_n(x), \widetilde{L}_n(x) \rangle}{\langle \widetilde{L}_n, \widetilde{L}_n \rangle}, \qquad \widetilde{b}_n = \frac{\langle x\widetilde{L}_n(x), \widetilde{L}_{n-1}(x) \rangle}{\langle \widetilde{L}_{n-1}, \widetilde{L}_{n-1} \rangle}.$$

Polynomials that satisfy the conditions (i), (ii) are not unique. For example, each L_n could be multiplied by a factor $c_n \neq 0$. However, this is the only freedom, i.e., each system L_n that satisfies the conditions (i), (ii) is of the form $L_n = c_n \widetilde{L}_n$ with the above constructed $\widetilde{L}_n$. The "classiscal" Legendre polynomials L_n are fixed by the "normalization condition" $L_n(1) = 1$. We have:

Theorem 2.14 (Legendre polynomials) *There holds:*

A. There is a unique sequence $(L_n)_{n \in \mathbb{N}_0}$ of polynomials $L_n \in \mathcal{P}_n$, the Legendre polynomials, *that satisfy the following conditions:*

(i) $\{L_0, \ldots, L_n\}$ is a basis of $\mathcal{P}_n$ (for each n)

(ii) L_n is orthogonal to the space $\mathcal{P}_{n-1}$, i.e., satisfies (2.4) for all $n \in \mathbb{N}_0$.

(iii) $L_n(1) = 1$ for all $n \in \mathbb{N}_0$.

*B. The Legendre polynomials L_n have the explicit representation ("*Rodrigues formula*")*

$$L_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n \tag{2.8}$$

C. The Legendre polynomials satisfy the "3-term recurrence relation"

$$(n+1)L_{n+1}(x) = (2n+1)xL_n(x) - nL_{n-1}(x) \tag{2.9}$$

Remark 2.15 *The 3-term recurrence (2.9) is the standard way to evaluate Legendre polynomials. The Legendre polynomials are a very important representative of the class of* orthogonal polynomials. *Other important families of orthogonal polynomials are the Chebyshev polynomials and the Jacobi polynomials. All orthogonal polynomials satisfy 3-term recurrence relations and are typically evaluated in this way.*

2.4.2 Gaussian quadrature

The following result is key for the definition of Gaussian quadrature (the proof of it can be found in literature).

Theorem 2.16 *For each $n \in \mathbb{N}_0$, the Legendre polynomial L_{n+1} has exactly $n+1$ (pairwise distinct) zeros $x_0, \dots, x_n$. Furthermore, $x_i \in (-1,1)$ for all i.*

With the aid of the $n+1$ zeros of L_{n+1} we define the Gaussian quadrature as the interpolatory quadrature, i.e., we interpolate the integrand in the $n+1$ zeros of L_{n+1} and integrate the interpolating polynomial:

$$\text{Gauss points: } x_{i,n}^G = \text{zeros of } L_{n+1} \tag{2.10a}$$

$$\boxed{\text{Gauss weights: } w_{i,n}^G = \int_{-1}^{1} \ell_i(x)\,dx,} \qquad \ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_{j,n}^G}{x_{i,n}^G - x_{j,n}^G} \tag{2.10b}$$

By construction, this is a quadrature formula $Q_n^{Gauss}(f) := \sum_{i=0}^n w_{i,n}^G f(x_{i,n}^G)$ that is exact for polynomials of degree n:

$$\int_{-1}^{1} g(x)\,dx = Q_n^{Gauss}(g) \qquad \forall g \in \mathcal{P}_n \tag{2.11}$$

In fact, for Gaussian quadrature there hold even stronger statements:

- Gaussian quadrature is exact for polynomials of degree $2n+1$:

  Let $f \in \mathcal{P}_{2n+1}$. With the aid of polynomial division ("Euklidian algorithm") we write f as

  $$f(x) = L_{n+1}(x)q_n(x) + r_n(x)$$

  with two polynomials q_n, $r_n \in \mathcal{P}_n$. Then

  $$\begin{aligned}\int_{-1}^{1} f(x)\,dx &= \underbrace{\int_{-1}^{1} L_{n+1}(x)q_n(x)\,dx}_{=0 \text{ by } (2.4)} + \int_{-1}^{1} r_n(x)\,dx \\ &= \int_{-1}^{1} r_n(x)\,dx \overset{(2.11)}{=} Q_n^{Gauss}(r_n) = \sum_{i=0}^{n} w_{i,n}^G r_n(x_{i,n}^G) \\ &\overset{L_{n+1}(x_{i,n}^G)=0}{=} \sum_{i=0}^{n} w_{i,n}^G L_{n+1}(x_{i,n}^G)q_n(x_{i,n}^G) + r_n(x_{i,n}^G) = Q_n^{Gauss}(f)\end{aligned}$$

- The Gauss weights $w_{i,n}^G > 0$ are positive (this is important to avoid cancelation):

  We apply the quadrature formula to ℓ_i to obtain

  $$\begin{aligned}w_{i,n}^G &\overset{\ell_i(x_{j,n}^G)=\delta_{i,j}}{=} \sum_{j=0}^{n} w_{j,n}\ell_i(x_{j,n}^G) \overset{\ell_i(x_{j,n}^G)=\delta_{i,j}}{=} \sum_{j=0}^{n} w_{j,n}(\ell_i(x_{j,n}^G))^2 \\ &= Q_n^{Gauss}(\ell_i^2) \overset{\ell_i^2 \in \mathcal{P}_{2n},(2.12)}{=} \int_{-1}^{1} \ell_i^2(x)\,dx > 0.\end{aligned}$$

- Gaussian quadrature is optimal (w.r.t. to degree of exactness):

  There is no rule with $n+1$ points that is exact for all polynomials of $\mathcal{P}_{2n+2}$: Let x_i, $i = 0, \dots, n$, be the quadrature points of a rule. Consider

$$f(x) = \prod_{j=0}^{n} (x - x_i)^2 \in \mathcal{P}_{2n+2}$$

  Then $0 < \int_{-1}^{1} f(x)\,dx$, but $Q_n(f) = 0$.

We summarize the findings in the following theorem.

Theorem 2.17 (Gaussian quadrature) *The quadrature rule Q_n^{Gauss} defined (2.10) satisfies:*

$$Q_n^{Gauss}(f) = \int_{-1}^{1} f(x)\,dx \qquad \forall f \in \mathcal{P}_{2n+1} \tag{2.12}$$

$$w_{i,n}^G > 0 \qquad i = 0, \dots, n. \tag{2.13}$$

Furthermore, there is no *quadrature rule with $n+1$ points that is exact for all polynomials of degree $2n+2$.*

Gaussian quadrature converges for integrands $f \in C([-1,1])$ if $n \to \infty$:

Theorem 2.18 (convergence of Gaussian quadrature) *There holds:*

$$\left| \int_{-1}^{1} f(x)\,dx - Q_n^{Gauss}(f) \right| \le 4 \min_{v \in \mathcal{P}_{2n+1}} \|f - v\|_{\infty,[-1,1]}. \tag{2.14}$$

In particular there holds $\int_{-1}^{1} f(x)\,dx = \lim_{n\to\infty} Q_n^{Gauss}(f)$ for each $f \in C([-1,1])$.

Proof: As for Theorem 2.8 we exploit that the quadrature rule is exact for polynomials of a particular degree. For arbitrary $v \in \mathcal{P}_{2n+1}$ we have

$$\begin{aligned}
\int_{-1}^{1} f(x)\,dx - Q_n^{Gauss}(f) &\overset{\text{Thm. 2.17}}{=} \int_{-1}^{1} (f(x) - v(x))\,dx + Q_n^{Gauss}(v) - Q_n^{Gauss}(f) \\
&= \int_{-1}^{1} (f(x) - v(x))\,dx + Q_n^{Gauss}(v - f)
\end{aligned}$$

and therefore (note: $\sum_{i=0} w_{i,n}^G = Q_n^{Gauss}(1) = \int_{-1}^{1} 1\,dx = 2$)

$$\begin{aligned}
\left| \int_{-1}^{1} f(x)\,dx - Q_n^{Gauss}(f) \right| &\le \left| \int_{-1}^{1} f(x) - v(x)\,dx \right| + \left| Q_n^{Gauss}(f - v) \right| \\
&\le 2\|f - v\|_{\infty,[-1,1]} + \sum_{i=0}^{n} \underbrace{|w_{i,n}^G|}_{= w_{i,n}^G \text{ b/c of } (2.13)} \underbrace{|f(x_{i,n}^G) - v(x_{i,n}^G)|}_{\le \|f-v\|_{\infty,[-1,1]}} \\
&\le (2 + \sum_{i=0}^{n} w_{i,n}^G) \|f - v\|_{\infty,[-1,1]} = 4\|f - v\|_{\infty,[-1,1]}.
\end{aligned}$$

Since $v \in \mathcal{P}_{2n+1}$ was arbitrary, we may pass to the minimum and arrive at the claim. □

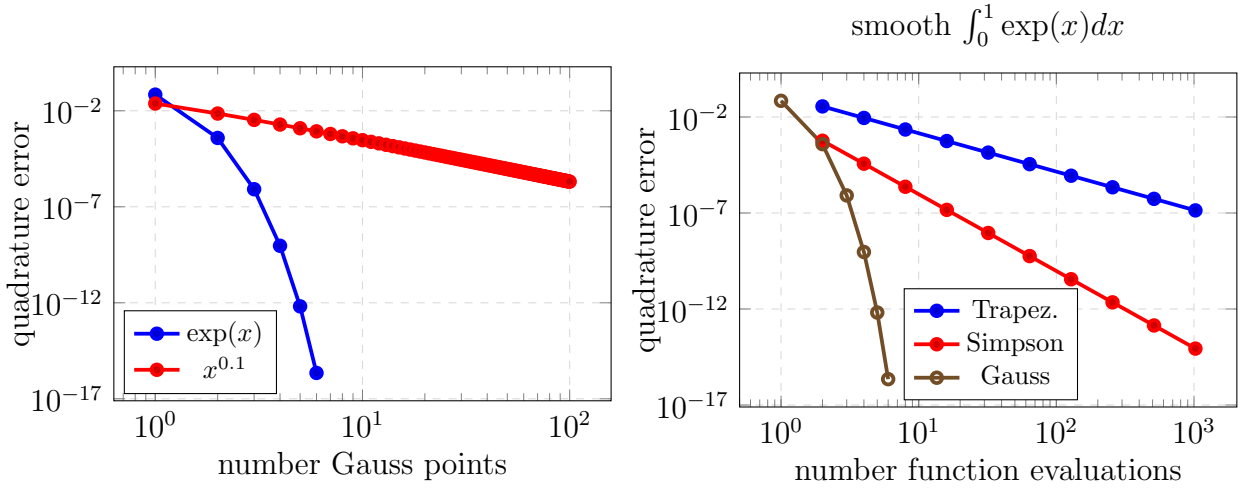


Figure 2.5: Gaussian quadrature on the interval $[0,1]$ for smooth integrand $f(x) = \exp(x)$ and non-smooth integrand $f(x) = x^{0.1}$ (left). Comparison of Gaussian quadrature with trapezoidal rule and Simpson rule for smooth integrand (right).

Gaussian quadrature is very efficient for *smooth* integrands:

Example 2.19 *We consider Gaussian quadrature with $n+1$ points on the interval $[0,1]$ (i.e., the quadrature points are $x_i = \frac{1}{2}(1 + x_{i,n}^G)$ and the weights $w_i = \frac{1}{2}w_{i,n}^G$) for $f_1(x) = \exp(x)$ and $f_2(x) = x^{0.1}$. While very rapid convergence is visible for the smooth integrand f_1, Gaussian quadrature is not very efficient for the non-smooth integrand f_2.*

slide 8 - Gaussian quadrature

Typically, Gaussian quadrature is also employed in composite rules. Then the number $n+1$ of Gaussian points (per subinterval) is typically fixed. Convergence results analogous to those for the composite trapezoidal and Simpson rule of Theorem 2.8 hold true.

Remark 2.20 *There is no explicit formula for the Gauss points and weigths for $n \geq 5$. There are may implementations, e.g.,* `gauleg.c` *from "Numerical Recipes" (also available as* `gauleg.m`*)) or* `numpy.polynomial.legendre.leggauss`*.*

2.5 Comments on the trapezoidal rule

The (composite) Gauss rules are much more efficient than then composite trapezoidal rule for smooth integrands. There is one exception: the integration of smooth *periodic* functions over one period. In this case, the trapezoidal rule converges very rapidly and is typically employed:

Example 2.21 **slide 9 - trapezoidal rule**
We employ the composite trapezoidal rule for the numerical integration on $[-1,1]$ for the following three periodic functions:

$$f_1(x) = \sin(\pi x), \qquad f_2(x) = (\cos(\pi x))^{10}, \qquad f_3(x) = \exp(\sin(8\pi x)).$$

We observe in Fig. 2.6: the composite trapezoidal rule is exact for rather large step sizes h for f_1; for somewhat large step sizes it is exact for the trigonometric polynomial f_2; for f_3 we also observe fast convergence.

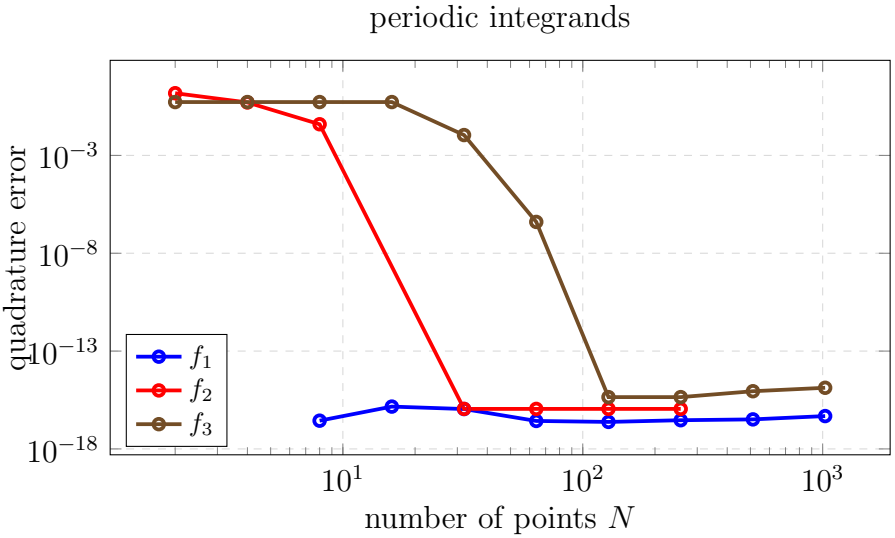


Figure 2.6: Composite trapezoidal rule on $[-1,1]$ for $f_1(x) = \sin(\pi x)$, $f_2(x) = (\cos(\pi x))^{10}$, $f_3(x) = \exp(\sin(8\pi x))$.

2.6 Quadrature in 2D

Goal: Determine $\int_T f(x)\,dx$, where $T \subset \mathbb{R}^2$ is the reference triangle $T = \{(x,y) \,|\, 0 < x < 1, 0 < y < 1 - x\}$ or the reference square $S = (0,1)^2$.
In principle, the typical construction of quarature rule for triangles or rectangles follows that in 1D: one selects quadrature points and weights in such a way that certain polynomials are integrated exactly.

2.6.1 Quadrature on squares

A quadrature rule for the square S is typically obtained from a 1D rule by a product construction. To that end, let

$$Q_n^{1D}(f) := \sum_{i=0}^{n} w_i f(x_i) \approx \int_0^1 f(x)\,dx \tag{2.15}$$

be a 1D rule. Then, one can define for functions $F(x,y)$ the 2D rule

$$Q_n^{2D}(F) := \sum_{i,j=0}^{n} w_i w_j F(x_i, x_j). \tag{2.16}$$

Exercise 2.22 *Let the 1D rule (2.15) be exact for polynomials of degree p, i.e., $Q_n^{1D}(f) = \int_0^1 f(x)\,dx$ for all $f \in \mathcal{P}_p$. Then the rule Q_n^{2D} is exact for all polynomials $F \in \operatorname{span}\{(x^i y^j \,|\, i,j = 0,\dots,p\}$.*

2.6.2 Quadrature on triangles

Quadrature rules on the triangle T are typically created in one of the following two ways:

1. One selects points in T. The condition that certain polynomials are integrated exactly determines the quadrature weights.

2. The triangle T is transformed to a square and a quadrature formula for the square S is employed.

Exercise 2.23 *The simplest case is a quadrature rule with* 1 *point, e.g., the barycenter of* T*. What is the corresponding quadrature weight so that the rule exact for polynomials of degree* 0*? Show that this rule is in fact exact for polynomials of degree* 1*, i.e., for polynomials* $F(x, y) = a + bx + cy$*.*
The next case is a quadrature rule with 3 *points, e.g., the vertices of* T*. Construct the weights such that the rule is exact for polynomials of degree* 1*.*

Example 2.24 *Let* Q^{2D} *be a quadrature rule on* S *with* N *points* $\mathbf{x}_i = (x_i, y_i) \in S$ *and corresponding weights* w_i*,* $i = 0, \ldots, N$*. The substitution ("Duffy transformation")*

$$\int_T F(x,y)\,dy\,dx = \int_{x=0}^{1}\int_{y=0}^{1-x} F(x,y)\,dy\,dx = \int_{x=0}^{1}\int_{\eta=0}^{1} F(x,(1-x)\eta)(1-x)\,d\eta\,dx$$

suggests the following quadrature rule on T*:*

$$\int_T F(x,y)\,dy\,dx = \int_{x=0}^{1}\int_{\eta=0}^{1} F(x,(1-x)\eta)(1-x)\,d\eta\,dx \approx \sum_i F(x_i,(1-x_i)y_i)(1-x_i)w_i.$$

Typically, rules Q^{2D} *for* S *are derived from 1D rules as described in Section 2.6.1. The 1D rule can be a Newton-Cotes formula or a Gauss rule or a composite Newton-Cotes or Gauss rule.*

2.6.3 Further comments

Integrals over "arbitray" domains $G \subset \mathbb{R}^2$ are typically done by composite rules, in which G is decomposed into triangles or quadrilaterals and each subdomain is then treated by a rule of the above type.

2.7 Comments on Gaussian quadrature (CSE)

goal: compute the Gaussian quadrature points and weights
In principle, there are two approaches to compute the Gaussian quadrature points, both of which are used in the numerical practice:

1. find the zeros of the Legendre polynomial L_{n+1} by some Newton method ($\rightarrow$ see below). The starting values are taken to be the Chebyshev points, which are explicitly available. The Legendre polynomials are evaluated using the three-term recurrence relation. Newton's method requires also the derivatives L'_{n+1}, which also satisfies a three-term recurrence relation.

2. Identify the zeros of L_{n+1} as eigenvalues of a suitable symmetric matrix in $\mathbb{R}^{(n+1)\times(n+1)}$ and compute those with some eigenvalue solver.

The following lemma shows how the zeros of L_n can be computed as the eigenalues of a matrix.

Lemma 2.25 *Let the functions L_i satisfy the three-term recurrence relation*

$$L_n(x) = (a_n x + b_n)L_{n-1}(x) - c_n L_{n-2}(x), \qquad n = 1, 2, \ldots, \qquad (L_0 := 1; L_1 := x) \tag{2.17}$$

Assume that a_i, $c_i > 0$ for all i. Then, the zeros of L_n are the eigenvalues of the matrix $\mathbf{J}$ of (2.19). Associated with each eigenvalue x_i, $i = 0, \ldots, n-1$ is an eigenvector $\mathbf{v}_i$ of $\mathbf{J}$. These eigenvectors are pairwise orthogonal.

Proof: The recurrence relation can be written as

$$x \begin{pmatrix} L_0(x) \\ L_1(x) \\ \vdots \\ \vdots \\ L_{n-1}(x) \end{pmatrix} = \underbrace{\begin{pmatrix} -\frac{b_1}{a_1} & \frac{1}{a_1} & 0 & \cdots & \\ \frac{c_2}{a_2} & -\frac{b_2}{a_2} & \frac{1}{a_2} & \cdots & \\ 0 & \ddots & \ddots & \ddots & \\ & & \frac{c_{n-1}}{a_{n-1}} & -\frac{b_{n-1}}{a_{n-1}} & \frac{1}{a_{n-1}} \\ & & & \frac{c_n}{a_n} & -\frac{b_n}{a_n} \end{pmatrix}}_{=:\mathbf{T}} \begin{pmatrix} L_0(x) \\ L_1(x) \\ \vdots \\ \vdots \\ L_{n-1}(x) \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ \frac{1}{a_n} L_n(x) \end{pmatrix} \tag{2.18}$$

we can write this as

$$x\mathbf{L} = \mathbf{T}\mathbf{L} + \frac{1}{a_n} L_n(x)\mathbf{e}_n,$$

where $\mathbf{T} \in \mathbb{R}^{n\times n}$ is a tridiagonal matrix and $\mathbf{e}_n = (0, 0, \ldots, 0, 1)^\top$ is a unit vector. This shows that $\mathbf{L} = (L_0(\xi), L_1(\xi), \ldots, L_{n-1}(\xi))^\top$ is an eigenvector for the eigenvalue ξ of $\mathbf{T}$ if and only if $L_n(\xi) = 0$. Hence, the eigenvalues of $\mathbf{T}$ are the zeros x_i of L_n with eigenvector $(L_0(x_i), \ldots, L_{n-1}(x_i))^\top$.
The tridiagonal matrix $\mathbf{T}$ can be made symmetric with a similarity transformation: for suitable

diagonal matrix $\mathbf{D} = \operatorname{diag}(d_0, \ldots, d_{n-1})$, there holds

$$\mathbf{DTD}^{-1} = \mathbf{J} = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ 0 & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \beta_{n-1} \\ & & & & \beta_{n-1} & \alpha_n \end{pmatrix}, \qquad \alpha_i := -\frac{b_i}{a_i}, \quad \beta_i = \left(\frac{c_{i+1}}{a_i a_{i+1}}\right)^{1/2}. \tag{2.19}$$

(Exercise: check this!) Since $\mathbf{D}$ is a diagonal matrix, the eigenvectors of $\mathbf{J}$ are the form

$$\mathbf{v}_i = \mathbf{D} \begin{pmatrix} L_0(x_i) \\ \vdots \\ L_{n-1}(x_i) \end{pmatrix} = \begin{pmatrix} d_i L_0(x_i) \\ \vdots \\ d_i L_{n-1}(x_i) \end{pmatrix}. \tag{2.20}$$

Since $\mathbf{J}$ is a symmetric matrix, its eigenvectors $\mathbf{v}_i$ are pairwise orthogonal. □

Once the Gauss points $x_0, \ldots, x_{n-1}$ (i.e., the zeros of L_n) have been determined, the weights w_i, $i = 0, \ldots, n-1$, can be computed by solving a linear system of equations from the exactness condition

$$\int_{-1}^{1} f(x)\,dx = \sum_{j=0}^{n-1} w_j f(x_j), \qquad \forall f \in \mathcal{P}_{n-1}. \tag{2.21}$$

In fact, if the eigenvectors of the matrix $\mathbf{J}$ are available, then the weights w_i can easily be determined directly:

Lemma 2.26 *Let $\mathbf{v}_0, \ldots, \mathbf{v}_{n-1}$ be a basis of $\mathbb{R}^n$ of eigenvectors of $\mathbf{J}$ corresponding to the eigenvalues x_i, $i = 0, \ldots, n-1$. Then the quadrature weights are given by*

$$w_i(\mathbf{v}_i^\top \mathbf{v}_i) = \int_{-1}^{1} L_0^2(x)\,dx = 2((\mathbf{v}_i)_1)^2, \qquad i = 0, \ldots, n-1.$$

Proof: By Lemma 2.25, the eigenvectors $\mathbf{v}_i$, $i = 0, \ldots, n-1$, of the matrix $\mathbf{J}$ are orthogonal, i.e., $\mathbf{v}_i^\top \mathbf{v}_j = 0$ for $i \neq j$. Formula (2.20) shows that

$$\mathbf{v}_i = d_i (L_0(x_i), \ldots, L_{n-1}(x_i))^\top, \qquad i = 0, \ldots, n-1.$$

From the exactness condition (2.21) applied to the function $f(x) = d_i L_i(x)$, we get

$$\sum_j w_j d_i L_i(x_j) = \int_{-1}^{1} d_i L_i(x)\,dx \overset{L_0=1}{=} d_i \int_{-1}^{1} L_0(x) L_i(x)\,dx \overset{L_j \text{ orthog.}}{=} \delta_{i0} d_i \|L_0\|^2_{L^2(-1,1)} \overset{L_0=1}{=} 2 d_i \delta_{i0} \tag{2.22}$$

With the matrix $\mathbf{V}$ and the unit vector $\mathbf{e}_1$ given by

$$\mathbf{V} := (\mathbf{v}_1, \ldots, \mathbf{v}_{n-1}), \qquad \mathbf{e}_1 = (1, 0, \ldots, 0)^\top$$

the n equations in (2.22) can be written as

$$\mathbf{V}\mathbf{w} = 2 d_i \mathbf{e}_1,$$

where $\mathbf{w} = (w_0, \dots, w_n)^\top$. Multiplying from the left by the vector $\mathbf{v}_i^\top$ and using that the eigenvectors are pairwise orthogonal yields

$$\mathbf{v}_i^\top \mathbf{v}_i w_i = 2d_i \mathbf{v}_i^\top \mathbf{e}_1 = 2d_i (\mathbf{v}_i)_1 \overset{(\mathbf{v}_i)_1 = d_i L_0 = d_i}{=} 2((\mathbf{v}_i)_1)^2$$

□

2.7.1 Gaussian quadrature with weights

goal: given a positive function ω on $(-1, 1)$, determine quadrature points x_i, $i = 0, \dots, n$ and weights w_i such that the exactness condition

$$\int_{-1}^{1} f(x)\omega(x)\,dx = \sum_{i=0}^{n} w_i f(x_i) \tag{2.23}$$

holds for polymials of as a degree as possible. Proceeding as in the case of the classical Gaussian quadrature (i.e., $\omega \equiv 1$) one can show that

Theorem 2.27 *If the function ω is positive on $(-1, 1)$ and satisfies $\int_{-1}^{1} \omega(x)\,dx < \infty$ there are points $x_i \in (-1, 1)$, $i = 0, \dots, n$, and weights $w_i > 0$ such that*

$$\int_{-1}^{1} f(x)\omega(x)\,dx = \sum_{i=0}^{n} w_i f(x_i) \qquad \forall f \in \mathcal{P}_{2n+1}. \tag{2.24}$$

In particular, for the quadrature error one has

$$\left| \int_{-1}^{1} f(x)\omega(x)\,dx - \sum_{i=0}^{n} w_i f(x_i) \right| \leq 2 \left(\int_{-1}^{1} \omega(x)\,dx \right) \inf_{v \in \mathcal{P}_{2n+1}} \|f - v\|_{\infty,[-1,1]}. \tag{2.25}$$

The theory is set up completely analogously to the case of the Gaussian quadrature: one computes polynomials that are pairwise orthogonal with respect to the weighted inner product

$$\langle u, v \rangle = \int_{-1}^{1} u(x)v(x)\omega(x)\,dx.$$

Denoting these orthogonal polynomials P_n, the quadrature points x_i, $i = 0, \dots, n$ are the zeros of P_{n+1}. The weights are obtained by requiring exactness of the quadrature rule for polynomials of degree n.
Important examples are:

1. $\omega \equiv 1$: the orthogonal polynomials are the Legendre polynomials L_n
2. $\omega(x) = (1 - x^2)^{-1}$: the orthogonal polynomials are the Chebyshev polynomials T_n
3. $\omega(x) = (1 - x)^\alpha (1 + x)^\beta$ for some α, $\beta > -1$: the orthogonal polynomials are the Jacobi polynomials, usually denoted $P_n^{(\alpha,\beta)}$. Note that the special case $\alpha = \beta = 0$ corresponds to the Legendre polynomials and $\alpha = \beta = -1/2$ to the Chebyshev polynomials.

3 Conditioning and Error Analysis

3.1 Error measures

Numerical simulations contain errors that come from various sources:

- Modelling error: when describing a problem with mathematical equations, various effects are typically neglected (e.g., continuum models versus the atomic structure of gases or solids)
- measurement errors: models typically contain parameters that have to be measured
- roundoff errors: computers work with finite precision numbers (typically floating point numbers), so that an error is made in each floating point operation
- discretization errors: numerical methods are not exact. Examples we have encountered are numerical differentiation and integration

Errors are typically measured using *norms* (see appendix).

slide 10 - errors

3.2 Conditioning

The condition number of a problem measures how the (exact) mathematical problem deals with perturbations/errors in the input data:

Definition 3.1 *The condition number of a problem (described as the evaluation of a function f) is the factor by which input perturbations are amplified in the worst case. One distinguishes:*

(a) absolute condition number $\kappa_{abs}(x)$ *is the smallest number such that for all sufficiently small* Δx*:*

$$\|f(x) - f(x + \Delta x)\| \leq \kappa_{abs}(x)\|\Delta x\|.$$

(b) relative condition number $\kappa_{rel}(x)$ *is the smallest number such that for all sufficiently small* Δx*:*

$$\frac{\|f(x) - f(x + \Delta x)\|}{\|f(x)\|} \leq \kappa_{rel}(x)\frac{\|\Delta x\|}{\|x\|}$$

In practice, one can compute the condition number in terms of the derivative of f. In the interest of simplicity, we consider the simple case $f : \mathbb{R} \to \mathbb{R}$. If $f \in C^1$, then Taylor expansion yields $f(x + \Delta x) = f(x) + f'(x)\Delta x + \cdots$ so that (approximately) $|f(x + \Delta x) - f(x)| \leq |f'(x)|\,|\Delta x|$. Hence, we see that (essentially)

$$\kappa_{abs}(x) = |f'(x)|$$

For the relative condition number we obtain analogously (for $f(x) \neq 0$)

$$\frac{|f(x + \Delta x) - f(x)|}{|f(x)|} \approx \frac{|f'(x)\Delta x|}{|f(x)|} = \frac{|f'(x)|\,|x|}{|f(x)|}\frac{|\Delta x|}{|x|}.$$

That is, we expect

$$\kappa_{rel}(x) = \frac{|f'(x)|}{|f(x)|}|x|.$$

In the following, we consider the relative condition number of a problem. We say that a problem is *well conditioned*, if $\kappa_{rel}(x)$ is "moderate" and it is called *ill conditioned*, if $\kappa_{rel}(x)$ is "large". The notion of "moderate" and "large" are vague, since it depends on the setting and the ultimate goal of the calculation whether a certain amplification of input errors is acceptable or not.

Example 3.2 *The addition of two* positive *numbers is well conditioned: Let x, $y > 0$ and Δx, Δy with $|\Delta x|/x \leq \delta$ and $|\Delta y|/y \leq \delta$. Then*

$$\frac{|(x+\Delta x)+(y+\Delta y)-(x+y)|}{|x+y|} \leq \frac{|\Delta x|+|\Delta y|}{x+y} \overset{x,y>0}{\leq} \frac{\delta x+\delta y}{x+y} \leq \delta,$$

i.e. $\kappa_{rel} \leq 1$. The (relative) error in the result is at most as large as the (relative) input error.

Example 3.3 **slide 11 - Cancellation**
Subtracting two numbers of similar size *is ill-conditioned ("cancellation"). Consider the subtraction*

$$x_1 = 1.2345\mathbf{89}? \cdot 10^0$$
$$x_2 = 1.2345\mathbf{79}? \cdot 10^0$$

where ? stands for an error/uncertainty in the input. The relative input error is thus of size 10^{-8}. For the difference

$$x_1 - x_2 = 0.0000011? \cdot 10^0 = 1.1? \cdot 10^{-6}$$

we get a relative error/uncertainty of 10^{-2}. Thus, we have lost 6 digits. Correspondingly, the (relative) condition number is $\kappa_{rel} \approx 1.8 \cdot 10^6$. Auxiliary computation:

$$\left|\frac{(x+\Delta x)-(y+\Delta y)-(x-y)}{x-y}\right| = \left|\frac{\Delta x-\Delta y}{x-y}\right| \leq \frac{|\Delta x|+|\Delta y|}{|x-y|}.$$

This leads to $2 \cdot 10^{-8}/(1.1 \cdot 10^{-6}) \approx 1.8 \cdot 10^{-2}$.

Exercise 3.4 *Show that multiplication and division are well conditioned (relative conditioning).*

3.3 Stability of algorithms

The algorithmic realization of a mathematical function f is typically done as a concatenation

$$f = f_1 \circ f_2 \cdots \circ f_N$$

of functions $f_1, \ldots, f_N$, where one may think of the functions f_i as "elementary functions" such as the addition, subtraction, multiplication, division or as more complex subproblems such as the evaluation of integrals, finding zeros of functions, solutions of differential equations. An algorithm will typically not realize a function exactly, i.e., f will be approximated by

$$\widehat{f} = \widehat{f}_1 \circ \widehat{f}_2 \cdots \circ \widehat{f}_N.$$

Examples of such approximations are:

- A computer realizes numbers typically as floating point numbers. Hence, already the input is rounded. The elementary operations $+$, $-$, $*$, / cannot be realized exactly.
- Subproblems f_i such as the evaluation of integrals are not exact but are tainted with discretization errors.

An inaccuracy/error that results from using an approximation $\widehat{f}_i$ instead of f_i is potentially amplified by the subsequent functions $\widehat{f}_1, \ldots, \widehat{f}_{i-1}$. A stability analysis of algorithms tries to identify ill-conditioned subproblems $\widehat{f}_i$ and will possibly modify them. Modifying subproblems $\widehat{f}_i$ (or choosing a different decomposition $f_1 \circ \cdots \circ f_{N'}$) is a sensible approach if some subproblems are ill-conditioned but if at the same time the corresponding "exact" functions are well conditioned. We illustrate this procedure with some simple examples in which cancellation (cf. Example 3.3) is the culprit.

Example 3.5 **slide 12 - stability**
Consider the evaluation of the function $f(x) = \log(1+x)$ for small x. The problem is well-conditioned since

$$\kappa_{rel}(x) = \frac{|f'(x)||x|}{|f(x)|} = \frac{|x|}{(1+x)|\log(1+x)|} \leq 2 \qquad \textit{(for } x \textit{ sufficiently close to } 0\textit{)}$$

The "naive" numerical realization is

$$x \overset{f_2}{\mapsto} w := (x+1) \overset{f_1}{\mapsto} \log w.$$

The mapping f_1 is ill-conditioned near $w = x + 1$:

$$\kappa_{rel}(w) \approx \frac{w}{w|\log w|} = \frac{1}{|\log w|} = \frac{1}{\log(1+x)} \approx \frac{1}{x}.$$

Hence, we observe the following: The intermediate result $1 + x$ has a relative accuracy of 16 digits but the subsequent application of f_2 may amplify (relative) inaccuracies by a factor $\approx 1/x$. For example, for $x = 10^{-10}$ one has to fear that one loses 10 digits. Indeed, in `matlab`*:*

```
>> x=1.234567890123456e-10;
>> w=1+x; f=log(w)
f =
     1.234568003306966e-10
```

The true value (rounded to 16 digits) is $f = \mathbf{1.23456}7890047248e-10$. *That is, although the IEEE-floating point arithmetic of* `matlab` *uses 16 digits, the result has only 6 corret digits, i.e., 10 digits were lost.*
Since the original function f *is well-conditioned, one may hope to find another algorithm that circumvents this cancellation problem. Indeed, using, e.g., the Taylor approximation of* f *for small* x *gives*

$$f(x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \cdots \tag{3.1}$$

and one obtains for $x - x^2/2$ *the value* $1.234567890047248e-10$, *which is correct to all digits. This example is not untypical. The situation is such that the final result (here:* x*) is small but that the intermediate results (here:* $1 + x \approx 1$*) are large relative to the final result. One should fear that the small final result is then somehow obtained by subtracting numbers of similar size. A different way of understanding the problem is: by (3.1) the final result is approximately* x *so that one shouldn't lose information contained in the digits of* x. *However, the intermediate results remove information about* x *as the following calculation with 16 digits shows:*

$$\begin{array}{l} \mathbf{1.000000000000000} \\ \mathbf{0.0000000001234567}890123456 \\ \hline \mathbf{1.0000000001234568} \end{array}$$

Example 3.6 *The two zeros of the quadratic equation* $x^2 - 2px - q = 0$ *are given by*

$$x_0 = p - \sqrt{p^2 + q}, \qquad x_1 = p + \sqrt{p^2 + q}. \tag{3.2}$$

A (mathematically equivalent) alternative formula is given by

$$x_1 = p + \sqrt{p^2 + q}, \tag{3.3a}$$

$$x_0 = p - \sqrt{p^2 + q} = \frac{(p - \sqrt{p^2 + q})(p + \sqrt{p^2 + q})}{p + \sqrt{p^2 + q}} = \frac{-q}{p + \sqrt{p^2 + q}} = -\frac{q}{x_1} \tag{3.3b}$$

Consider the case $p, q > 0$. *If* $p^2 \gg q$ *we expect again cancellation when computing* x_0. *Indeed, in* `matlab`*:*

```
>> p = 400000; q = 1.234567890123456;
>> r = sqrt(p^2+q); x0=p-r
x0 =
    -1.543201506137848e-06
```

The exact solution is $-\mathbf{1.54320}9862651343129e-06$. *The reason is again cancellation in the last step of the realization of the formula for* x_0. *The alternative formula (3.3b) avoids this subtraction and yields a result with 16 correct digits:*

```
>> x1=p+sqrt(p^2+q);x0=-q/x1
x0 =
    -1.543209862651343e-06
```

4 Gaussian Elimination

Goal: solve, for given $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{b} \in \mathbb{R}^n$, the linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{4.1}$$

In the sequel, we will often denote the entries of matrices by lower case letters, e.g., the entries of $\mathbf{A}$ are $(\mathbf{A})_{ij} = a_{ij}$. Likewise for vectors, we sometimes write $\mathbf{x}_i = x_i$.

Remark 4.1 *In* `matlab`*, the solution of (4.1) is realized by* $x = A\backslash b$*. In* `python`*, the function* `numpy.linalg.solve` *performs this. In both cases, a routine from* `lapack`[1] *realizes the actual computation. The* `matlab` *realization of the backslash operator* $\backslash$ *is in fact very complex.*

4.1 Lower and upper triangular matrices

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

- an *upper triangular matrix* if $\mathbf{A}_{ij} = 0$ for $j < i$;
- a *lower triangular matrix* if $\mathbf{A}_{ij} = 0$ for $j > i$.
- a *normalized lower triangular matrix* if, in addition to being lower triangular, it satisfies $\mathbf{A}_{ii} = 1$ for $i = 1, \ldots, n$.

$$\begin{pmatrix} * & & & \\ * & * & & \\ * & * & * & \\ * & * & * & * \end{pmatrix} \qquad \begin{pmatrix} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{pmatrix}$$

Figure 4.1: schematic representation of lower (left) and upper (right) matrices ($n = 4$); blank spaces represent a 0

Linear systems where $\mathbf{A}$ is a lower or upper triangular matrix are easily solved by "forward substitution" or "back substitution":

Algorithm 4.2 (solve $\mathbf{L}\mathbf{x} = \mathbf{b}$ using *forward substitution*)
Input: $\mathbf{L} \in \mathbb{R}^{n\times n}$ *lower triangular, invertible,* $\mathbf{b} \in \mathbb{R}^n$
Output: *solution* $\mathbf{x} \in \mathbb{R}^n$ *of* $\mathbf{L}\mathbf{x} = \mathbf{b}$

for $j = 1{:}n$ ***do***
$$x_j := \left(b_j - \sum_{k=1}^{j-1} l_{jk} x_k \right) \Big/ l_{jj} \qquad [\![\textit{ convention: empty sum} = 0]\!]$$
end for

[1] linear algebra package, see `en.wikipedia.org/wiki/LAPACK`

Algorithm 4.3 (solution of $\mathbf{Ux} = \mathbf{b}$ using *back substitution*)
Input: $\mathbf{U} \in \mathbb{R}^{n\times n}$ *upper triangular, invertible,* $\mathbf{b} \in \mathbb{R}^n$
Output: *solution* $\mathbf{x} \in \mathbb{R}^n$ *of* $\mathbf{Ux} = \mathbf{b}$

for *j = n:-1:1* ***do***
$$x_j := \left(b_j - \sum_{k=j+1}^{n} u_{jk} x_k \right) \Big/ u_{jj}$$
end for

The cost of Algorithms 4.2 and 4.3 are $O(n^2)$:

Exercise 4.4 *Compute the number of multiplications and additions in Algorithms 4.2 and 4.3.*

The set of upper and lower triangular matrices are closed under addition and matrix multiplication[2]:

Exercise 4.5 *Let* $\mathbf{L}_1$, $\mathbf{L}_2 \in \mathbb{R}^{n\times n}$ *be two lower triangular matrices. Show:* $\mathbf{L}_1 + \mathbf{L}_2$ *and* $\mathbf{L}_1\mathbf{L}_2$ *are lower triangular matrices. If* $\mathbf{L}_1$ *is additionally invertible, then its inverse* $\mathbf{L}_1^{-1}$ *is also a lower triangular matrix. Analogous results hold for upper triangular matrices.*

Remark 4.6 (representation via scalar products) *Alg. 4.2 (and analogously Alg. 4.3) can be writen using scalar products:*

for *j = 1:n* ***do***
$$x(j) := \Big[b(j) - L(j, 1:j-1) * x(1:j-1) \Big] \Big/ L(j,j)$$
end for

The advantage of such a formulation is that efficient libraries are available such as BLAS level 1[3]*. More generally, rather than realizing dot-products, matrix-vector products, or matrix-matrix-products directly by loops, it is typically advantageous to employ optimized routines such as BLAS.*

Remark 4.7 *In Remark 4.6, the matrix* $\mathbf{L}$ *is accessed in row-oriented fashion. One can reorganize the two loops so as to access* $\mathbf{L}$ *in a column-oriented way. The following algorithm overwrites* $\mathbf{b}$ *with the solution* $\mathbf{x}$ *of* $\mathbf{Lx} = \mathbf{b}$*:*

for *j = 1:n-1* ***do***
$b(j) = b(j)/L(j,j)$
$b(j+1:n) := b(j+1:n) - b(j)L(j+1:n, j)$
end for

[2]That is, they have the mathematical structure of a ring
[3]Basic Linear Algebra Subprograms, see `en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms`

4.2 Classical Gaussian elimination

slide 13 - Gaussian elimination

The classical Gaussian elimination process transforms the linear system (4.1) into upper triangular form, which can then be solved by back substitution. We illustrate the procedure:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & & & & & & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array} \tag{4.2}$$

Multiplying the $1st$ equation by

$$l_{21} := \frac{a_{21}}{a_{11}}$$

and subtracting this from the $2nd$ equation produces:

$$\underbrace{(a_{21} - \frac{a_{21}}{a_{11}}a_{11})}_{0}\, x_1 + \underbrace{(a_{22} - \frac{a_{21}}{a_{11}}a_{12})}_{=:a_{22}^{(2)}}\, x_2 + \cdots + \underbrace{(a_{2n} - \frac{a_{21}}{a_{11}}a_{1n})}_{=:a_{2n}^{(2)}}\, x_n \;=\; \underbrace{b_2 - \frac{a_{21}}{a_{11}}b_1}_{=:b_2^{(2)}} \tag{4.3}$$

Multiplying the $1st$ equation by

$$l_{31} := \frac{a_{31}}{a_{11}}$$

and subtracting this from the $3rd$ equation produces:

$$\underbrace{(a_{31} - \frac{a_{31}}{a_{11}}a_{11})}_{0}\, x_1 + \underbrace{(a_{32} - \frac{a_{31}}{a_{11}}a_{12})}_{=:a_{32}^{(2)}}\, x_2 + \cdots + \underbrace{(a_{3n} - \frac{a_{31}}{a_{11}}a_{1n})}_{a_{3n}^{(2)}} \cdot x_n \;=\; \underbrace{b_3 - \frac{a_{31}}{a_{11}}b_1}_{b_3^{(2)}} \tag{4.4}$$

Generally, multiplying for $i = 2, \ldots, n$, the $1st$ equation by $l_{i1} := a_{i1}/a_{11}$ and subtracting this from the ith equation yields the following equivalent system of equations:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ & & a_{22}^{(2)}x_2 & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)} \\ & & \vdots & & & & & & \vdots \\ & & a_{n2}^{(2)}x_2 & + & \cdots & + & a_{nn}^{(2)}x_n & = & b_n^{(2)} \end{array} \tag{4.5}$$

Repeating this process for the $(n-1) \times (n-1)$ subsystem

$$\begin{array}{ccccccc} a_{22}^{(2)}x_2 & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)} \\ \vdots & & & & & & \vdots \\ a_{n2}^{(2)}x_2 & + & \cdots & + & a_{nn}^{(2)}x_n & = & b_n^{(2)} \end{array}$$

of (4.5) yields

$$\begin{array}{ccccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ & & a_{22}^{(2)}x_2 & + & a_{23}^{(2)}x_3 & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)} \\ & & & & a_{33}^{(3)}x_3 & + & \cdots & + & a_{3n}^{(3)}x_n & = & b_3^{(3)} \\ & & & & \vdots & & & & \vdots & & \\ & & & & a_{n3}^{(3)}x_3 & & \cdots & & a_{nn}^{(3)}x_n & = & b_n^{(3)} \end{array} \tag{4.6}$$

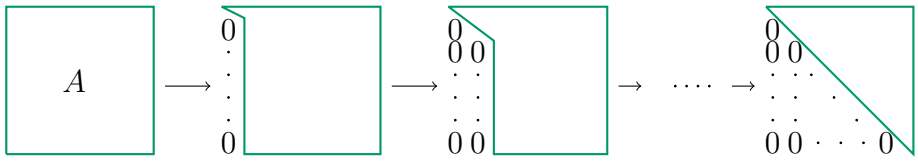


Figure 4.2: Gaussian elimination: reduction to upper triangular form

One repeats this procedure until one has reached triangular form. The following Alg. 4.8 realizes the above procedure: It overwrites the matrix $\mathbf{A}$ so that its upper triangle contains the final triangular form, which we will denote $\mathbf{U}$ below; the entries l_{ik} computed on the way will be collected in a normalized lower triangular matrix $\mathbf{L}$.

Algorithm 4.8 (Gaussian elimination without pivoting)
Input: $\mathbf{A}$
Output: *non-trivial entries of* $\mathbf{L}$ *and* $\mathbf{U}$*;* $\mathbf{A}$ *is overwritten by* $\mathbf{U}$

for $k = 1 : (n-1)$ ***do***
  for $i = (k+1) : n$ ***do***
    $l_{ik} := \frac{a_{ik}}{a_{kk}}$
    $A(i, [k+1:n]) += -l_{ik} \cdot A(k, [k+1:n])$
  end for
end for

Remark 4.9 *In* (4.8) *below, we will see that* $\mathbf{A} = \mathbf{LU}$*, where* $\mathbf{U}$ *and* $\mathbf{L}$ *are computed in Alg. 4.8. Typically, the off-diagonal entries of* $\mathbf{L}$ *are stored in the lower triangular part of* $\mathbf{A}$ *so that effectively,* $\mathbf{A}$ *is overwritten by its LU-factorization.*

Exercise 4.10 *Expand Alg. 4.8 so as to include the modifications of the right-hand side* $\mathbf{b}$.

4.2.1 Interpretation of Gaussian elimination as an LU-factorization

With the coefficients l_{ij} computed above (e.g., in Alg. 4.8) one can define the matrices

$$\mathbf{L}^{(k)} := \begin{pmatrix} 1 & & & & & & & \\ 0 & \ddots & & & & & & \\ \vdots & \ddots & \ddots & & & & & \\ \vdots & & 0 & 1 & & & & \\ \vdots & & \vdots & l_{k+1,k} & \ddots & & & \\ \vdots & & \vdots & \vdots & 0 & \ddots & & \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \cdots & 0 & l_{n,k} & 0 & \cdots & 0 & 1 \end{pmatrix}, \qquad k = 1, \ldots, n-1$$

Exercise 4.11 *Check that the inverse of* $\mathbf{L}^{(k)}$ *is given by*

$$(\mathbf{L}^{(k)})^{-1} = \begin{pmatrix} 1 & & & & & & & \\ 0 & \ddots & & & & & & \\ \vdots & \ddots & \ddots & & & & & \\ \vdots & & 0 & 1 & & & & \\ \vdots & & \vdots & -l_{k+1,k} & \ddots & & & \\ \vdots & & \vdots & \vdots & 0 & \ddots & & \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \cdots & 0 & -l_{n,k} & 0 & \cdots & 0 & 1 \end{pmatrix}. \tag{4.7}$$

Each step of the above Gaussian elimination process sketched in Fig. 4.2 is realized by a multiplication from the left by a matrix, in fact, the matrix $(\mathbf{L}^{(k)})^{-1}$ (cf. (4.7)). That is, the Gaussian elimination process can be described as

$$\begin{aligned} &\mathbf{A} = \mathbf{A}^{(1)} \;\to\; \mathbf{A}^{(2)} = (\mathbf{L}^{(1)})^{-1}\mathbf{A}^{(1)} \;\to\; \mathbf{A}^{(3)} = (\mathbf{L}^{(2)})^{-1}\mathbf{A}^{(2)} = (\mathbf{L}^{(2)})^{-1}(\mathbf{L}^{(1)})^{-1}\mathbf{A}^{(1)} \;\to\; \dots \\ &\to \underbrace{\mathbf{A}^{(n)}}_{=:\mathbf{U} \text{ upper triangular}} = (\mathbf{L}^{(n-1)})^{-1}\mathbf{A}^{(n-1)} = \dots = (\mathbf{L}^{(n-1)})^{-1}(\mathbf{L}^{(n-2)})^{-1}\dots(\mathbf{L}^{(2)})^{-1}(\mathbf{L}^{(1)})^{-1}\mathbf{A}^{(1)} \end{aligned}$$

Rewriting this yields, the LU-factorization

$$\mathbf{A} = \underbrace{\mathbf{L}^{(1)}\cdots\mathbf{L}^{(n-1)}}_{=:\mathbf{L}}\mathbf{U}$$

The matrix $\mathbf{L}$ is a lower triangular matrix as the product of lower triangular matrices (cf. Exercise 4.5). In fact, due to the special structure of the matrices $\mathbf{L}^{(k)}$, it is given by

$$\mathbf{L} = \begin{pmatrix} 1 & & & \\ l_{21} & \ddots & & \\ \vdots & \ddots & \ddots & \\ l_{n1} & \cdots & l_{n,n-1} & 1 \end{pmatrix}$$

as the following exercise shows:

Exercise 4.12 *For each* k *the product* $\mathbf{L}^{(k)}\mathbf{L}^{(k+1)}\cdots\mathbf{L}^{(n-1)}$ *is given by*

$$\mathbf{L}^{(k)}\mathbf{L}^{(k+1)}\cdots\mathbf{L}^{(n-1)} = \begin{pmatrix} 1 & & & & & & & \\ 0 & \ddots & & & & & & \\ \vdots & \ddots & \ddots & & & & & \\ \vdots & & 0 & 1 & & & & \\ \vdots & & \vdots & l_{k+1,k} & 1 & & & \\ \vdots & & \vdots & \vdots & l_{k+2,k+1} & \ddots & & \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \cdots & 0 & l_{n,k} & l_{n,k+1} & \cdots & l_{n,n-1} & 1 \end{pmatrix}.$$

Thus, we have shown that Gaussian elimination produces a *factorization*

$$\mathbf{A} = \mathbf{LU}, \tag{4.8}$$

where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices determined by Alg. 4.8.

4.3 LU-factorization

In numerical practice, linear systems are solved by computing the factors $\mathbf{L}$ and $\mathbf{U}$ in (4.8) and the system is then solved with one forward and one back substitution:

1. compute $\mathbf{L}$, $\mathbf{U}$ such that $\mathbf{A} = \mathbf{LU}$
2. solve $\mathbf{Ly} = \mathbf{b}$ using forward substitution
3. solve $\mathbf{Ux} = \mathbf{y}$ using back substitution

Remark 4.13 *In* `matlab`*, the LU-factorization is realized by* `lu(A)`*. In* `python` *one can use* `scipy.linalg.lu`.

As we have seen in Section 4.2.1, the LU-factorization can be computed with Gaussian elimination. An alternative way of computing the factors $\mathbf{L}$, $\mathbf{U}$ is given in the following section[4]

4.3.1 Crout's algorithm for computing LU-factorization

We seek $\mathbf{L}$, $\mathbf{U}$ such that

$$\begin{pmatrix} 1 & & & \\ l_{21} & \ddots & & \\ \vdots & \ddots & \ddots & \\ l_{n1} & \cdots & l_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & \cdots & \cdots & u_{1n} \\ & \ddots & & \vdots \\ & & \ddots & \vdots \\ & & & u_{nn} \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} a_{11} & \cdots & \cdots & a_{1n} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{pmatrix}$$

This represents n^2 equations for n^2 unknowns, i.e., we are looking for l_{ij}, u_{ij}, such that

$$a_{ik} \stackrel{!}{=} \sum_{j=1}^{n} l_{ij}\ u_{jk}, \quad \forall i,k = 1,\ldots,n.$$

$\mathbf{L}$ is lower triangular, $\mathbf{U}$ is upper triangular $\Longrightarrow$

$$a_{ik} \stackrel{!}{=} \sum_{j=1}^{\min(i,k)} l_{ij}\ u_{jk} \qquad \forall i,k = 1,\ldots,n \tag{4.9}$$

[4]One reason for studying different algorithms is that the entries of $\mathbf{L}$ and $\mathbf{U}$ are computed in a different order so that these algorithms differ in their memory access and thus potentially in actual timings.

Idea:
Traverse the n^2 equations in (4.9) in following order: ("**Crout ordering**")]]

$$\begin{array}{c}(1,1)\ ,\ (1,2)\ ,\ \ldots\ ,(1,n)\\ (2,1)\ ,\ (3,1)\ ,\ \ldots\ ,(n,1)\\ (2,2)\ ,\ (2,3)\ ,\ \ldots\ ,(2,n)\\ (3,2)\ ,\ (4,2)\ ,\ \ldots\ ,(n,2)\\ etc.\end{array}$$

Procedure:

1. step: $i = 1,\ k = 1,\ldots,n$ in (4.9):

$$\underbrace{l_{11}}_{=1} u_{1k} \stackrel{!}{=} a_{1k}$$

$\Rightarrow$ $U(1,:)$ can be computed

2. step: $k = 1,\ i = 2,\ldots,n$ in (4.9):

$$l_{i1}u_{11} \stackrel{!}{=} a_{i1}$$

$\Rightarrow$ $L([2:n],1)$ can be determined

3. step: $i = 2,\ k = 2,\ldots,n$ in (4.9):

$$\underbrace{l_{21}}_{\substack{\text{is known}\\ \text{by 2. step}}}\ \underbrace{u_{1k}}_{\substack{\text{is known}\\ \text{by 1. step}}} + \underbrace{l_{22}}_{=1} u_{2k} \stackrel{!}{=} a_{2k} \qquad \text{for } k = 2,\ldots,n$$

$\Rightarrow$ can compute $U(2,[2:n])$

4. step: $k = 2,\ i = 3,\ldots,n$ in (4.9):

$$\underbrace{l_{i1}}_{\substack{\text{known by}\\ \text{2. step}}}\ \underbrace{u_{12}}_{\substack{\text{known by}\\ \text{1. step}}} + l_{i2}\ \underbrace{u_{22}}_{\substack{\text{known by}\\ \text{3. step}}} \stackrel{!}{=} a_{i2} \qquad \text{for } i = 3,\ldots,n$$

$\Rightarrow$ can compute $L([3:n],2)$

$\vdots$

$\vdots$

$\vdots$

The procedure is formalized in the following algorithm.

Algorithm 4.14 (Crout's LU-factorization)
Input: invertible matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ *that has an* LU*-factorization*
Output: the non-trivial entries of the normalized LU*-factorization*

for $i = 1 : n$ ***do***
  for $k = i : n$ ***do***
    $u_{ik} := a_{ik} - \sum_{j=1}^{i-1} l_{ij} u_{jk}$
  end for
  for $k = i+1 : n$ ***do***
    $l_{ki} := \left(a_{ki} - \sum_{j=1}^{i-1} l_{kj} u_{ji} \right) \Big/ u_{ii}$
  end for
end for

Remark 4.15 (cost when solving (4.1) with LU-factorization)

- *The* LU*-factorization dominates with* $O(n^3)$ *(more precisely:* $2/3n^3 + O(n^2)$ *floating point operations) the total cost, since the cost of back substitution and forward substitution are* $O(n^2)$
- *An advantage of an* LU*-factorization arises, when problems with multiple right-hand sides are considered: solving* $\mathbf{Ax} = \mathbf{b}$ *for* M *right-hand sides* $\mathbf{b}$*, requires only a single* LU*-factorization, i.e., the cost are* $\frac{2}{3}n^3 + 2Mn^2$

In practice $\mathbf{A}$ is overwritten by its LU-decomposition:

Algorithm 4.16 (LU-factorization with overwriting A)
Input: $\mathbf{A}$*, invertible,* A *has a* LU*-factorization*
Output: algorithm replaces a_{ij} *with* u_{ij} *for* $j \geq i$
and with l_{ij} *for* $j < i$

for $i = 1 : n$ ***do***
  for $k = i : n$ ***do***
    $a_{ik} := a_{ik} - \sum_{j=1}^{i-1} a_{ij}\, a_{jk}$
  end for
  for $k = (i+1) : n$ ***do***
    $a_{ki} := \left(a_{ki} - \sum_{j=1}^{i-1} a_{kj}\, a_{ji} \right) \Big/ a_{ii}$
  end for
end for

$$\begin{pmatrix} a_{11} & \cdots & a_{1,q+1} & & & & \\ \vdots & \ddots & & \ddots & & & \\ a_{p+1,1} & & \ddots & & \ddots & & \\ & \ddots & & \ddots & & \ddots & \\ & & \ddots & & \ddots & & a_{n-q,n} \\ & & & \ddots & & \ddots & \vdots \\ & & & & a_{n,n-p} & \cdots & a_{nn} \end{pmatrix}$$

Figure 4.3: banded matrix with upper bandwidth q and lower bandwidth p.

4.3.2 banded matrices

A matrix $A \in \mathbb{R}^{n\times n}$ is a *banded matrix* with *upper bandwidth* q and *lower bandwidth* p if $a_{ik} = 0$ for all i, k with $i > k + p$ or $k > i + q$. The following theorem shows that banded matrices are of interest if p and q are small (compared to n):

Theorem 4.17 *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a banded matrix with upper bandwidth q and lower bandwidth p. Let $\mathbf{A}$ be invertible and admit an LU-factorization. Then:*

(i) $\mathbf{L}$ *has lower bandwidth* p *and* $\mathbf{U}$ *has upper bandwidth* q.

(ii) Cost to solve $\mathbf{Ax} = \mathbf{b}$*:*

- *(a)* $O(npq)$ *floating point operations (flops) to determine LU-factorization*
- *(b)* $O(np)$ *flops to solve* $\mathbf{Ly} = \mathbf{b}$
- *(c)* $O(nq)$ *flops to solve* $\mathbf{Ux} = \mathbf{y}$

Proof: (Exercise) Prove (i) for the special case of a tridiagonal matrix, i.e., $p = q = 1$. To that end, proceed by induction on the matrix size n:

- $n = 1$ ✓
- for the induction step $n \to n+1$ make the ansatz

$$A = \left(\begin{array}{ccc|c} & & & 0 \\ & A_n & & \vdots \\ & & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 \; a_{n+1,n} & a_{n,n+1} \\ & & & a_{n+1,n+1} \end{array}\right) \overset{!}{=} \left(\begin{array}{ccc|c} & & & 0 \\ & L_n & & \vdots \\ & & & 0 \\ \hline & l^\top & & 1 \end{array}\right) \left(\begin{array}{ccc|c} & & & u \\ & U_n & & \vdots \\ & & & u \\ \hline & 0 & & \rho \end{array}\right)$$

and compute $l^\top$, u, and ρ. Use the structure of L_n, U_n given by the induction hypothesis.

□

4.3.3 Cholesky-factorization

A particularly important class of matrices $\mathbf{A}$ is that of symmetric positive definite (SPD) matrices:

- $\mathbf{A}$ is symmetric, i.e., $\mathbf{A}_{ij} = \mathbf{A}_{ji}$ for all i, j
- $\mathbf{A}$ is positive definite, i.e., $\mathbf{x}^\top \mathbf{A}\mathbf{x} > 0$ for all $\mathbf{x} \neq 0$.

Remark 4.18 *An alternative criterion for positive definiteness of a symmetric matrix is that all its eigenvalues are positive.*

For SPD matrices, one typically employs a variant of the *LU*-factorization, namely, the Cholesky-factorization, i.e.,

$$\mathbf{A} = \mathbf{C}\mathbf{C}^\top, \tag{4.10}$$

where the Cholesky factor $\mathbf{C}$ is lower triangular (but not normalized, i.e., the entries $\mathbf{C}_{ii}$ are not necessarily 1).

Exercise 4.19 *Formulate an algorithm to compute* $\mathbf{C}$. Hint: *Proceed as in Crout's method for the LU-factorization.*

Remark 4.20 *If an SPD matrix* $\mathbf{A}$ *is banded with bandwidth* $p = q$*, then the Cholesky factor* $\mathbf{C}$ *is also banded with the same bandwidth.*

Remark 4.21 *The cost of a Cholesky factorization (of either a full matrix or a banded matrix) is about half of that of the corresponding LU-factorization since only half the entries need to be computed.*

Remark 4.22 *A Cholesky factorization is computed in* `matlab` *with* `chol`.

4.3.4 Skyline matrices

slide 14 - banded and skyline matrices

Banded matrices are a particular case of *sparse matrices*, i.e., matrices with "few" non-zero entries. We note that the *LU*-factors have the same sparsity pattern, i.e., the zeros of $\mathbf{A}$ outside the band are inherited by the factors $\mathbf{L}$, $\mathbf{U}$.
Another important special case of sparse matrices are so-called *skyline matrices* as depicted on the left side of Fig. 4.4. More formally, a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is called a *skyline matrix*, if for $i = 1, \ldots, n$ there are numbers p_i, $q_i \in \mathbb{N}_0$ such that

$$a_{ij} = 0 \quad \text{if } j < i - p_i \text{ or } i < j - q_j. \tag{4.11}$$

We have without proof:

Theorem 4.23 *Let* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *be a skyline matrix, i.e., there are* p_i, q_i *with (4.11). Let* $\mathbf{A}$ *have an LU-factorization* $\mathbf{A} = \mathbf{L}\mathbf{U}$*. Then the matrices* $\mathbf{L}$, $\mathbf{U}$ *satisfy:*

$$l_{ij} = 0 \quad \textit{for } j < i - p_i, \qquad u_{ij} = 0 \quad \textit{for } i < j - q_j.$$

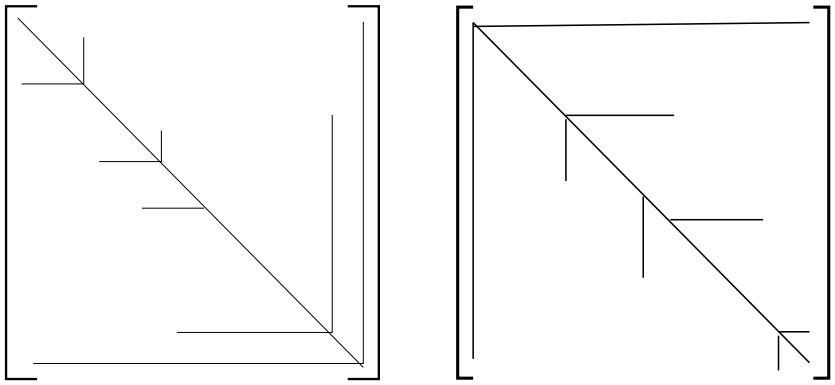


Figure 4.4: lines indicate non-zero entries. Left: *skyline* matrix, whose sparsity pattern is inherited by LU-factorization. Right: not a *skyline*-Matrix and the LU-factorization does *not* inherit the sparsity pattern.

$$A = \begin{pmatrix} 1 & & & 1 & & & 1 \\ & 1 & & 2 & & & 2 \\ & & 1 & 3 & & & 3 \\ 1 & 2 & 3 & 5 & & & 18 \\ & & & & 1 & & 5 \\ & & & & & 1 & 6 \\ 1 & 2 & 3 & 18 & 5 & 6 & 92 \end{pmatrix} \qquad L = U^\top = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ 1 & 2 & 3 & 1 & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ 1 & 2 & 3 & 4 & 5 & 6 & 1 \end{pmatrix}$$

Figure 4.5: $\mathbf{A} \in \mathbb{R}^{7\times 7}$ and its LU-factorization.

Theorem 4.23 states that the factors $\mathbf{L}$ and $\mathbf{U}$ have the same sparsity pattern as $\mathbf{A}$. Figure 4.5 illustrates this for a simple example. Obviously, this can be exploited algorithmically to economize on memory requirement and computing time by simply computing the non-zero entries of $\mathbf{L}$ and $\mathbf{U}$. Note that the matrices in Fig. 4.4 should not be treated as banded matrices as then the bands p, q would be n. The right example in Fig. 4.4 is not a skyline matrix, and the sparsity pattern of $\mathbf{A}$ is lost in the course of the LU-factorization: $\mathbf{L}$ ist in general a fully populated lower triangular matrix and $\mathbf{U}$ a fully populated upper triangular matrix. This is called *fill in*.

Exercise 4.24 *The sparsity pattern of matrices can be checked in* `matlab` *with the command* `spy`. *Check the sparsity patterns of the LU-factorization of the matrices $\mathbf{A}$ given above.*

Remark 4.25 *Modern solvers for sparse linear systems typically perform as a preprocessing step row and column permutations so as to minimize fill-in during factorization. (→ see Approximate Minimum Degree, Reverse Cuthill-McKee).*

4.4 Gaussian elimination with pivoting

4.4.1 Motivation

So far, we *assumed* that $\mathbf{A}$ admits a factorization $\mathbf{A} = \mathbf{LU}$. However, even if $\mathbf{A}$ is invertible, this need not be the case as the following example shows:

Exercise 4.26 *Prove that the matrix*

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 3 & 2 \end{pmatrix}$$

does not *have a factorization* $\mathbf{A} = \mathbf{LU}$ *with normalized lower triangular matrix* $\mathbf{L}$ *and upper triangular matrix* $\mathbf{U}$.

The key observation is that permuting the rows of $\mathbf{A}$ leads to a matrix that has an *LU*-factorization: Let

$$\mathbf{P} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

be the permutation matrix that interchanges the rows 1 and 2 of $\mathbf{A}$:

$$\mathbf{PA} = \begin{pmatrix} 3 & 2 \\ 0 & 1 \end{pmatrix}$$

This matrix has an *LU*-factorization. The general principle is:

Theorem 4.27 *Let* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *be invertible. Then there exists a permutation matrix* $\mathbf{P}$*, a normalized lower triangular matrix* $\mathbf{L}$*, and an upper triangular matrix* $\mathbf{U}$ *such that* $\mathbf{LU} = \mathbf{PA}$*. Here* $\mathbf{PA}$ *is a permutation of the rows of* $\mathbf{A}$.

Exercise 4.28 *Let* $\mathbf{P}$ *be given by*

$$\mathbf{P} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & 1 & \\ & & & & & & \ddots \end{pmatrix}$$

where the off-diagonal 1 *are in the positions* (i_1, i_2) *and* (i_2, i_1) *(with* $i_1 \neq i_2$*). Show: The matrix* $\mathbf{PA}$ *is the matrix* $\mathbf{A}$ *with rows* i_1 *and* i_2 *interchanged. Furthermore,* $\mathbf{P}^{-1} = \mathbf{P}^\top = \mathbf{P}$.

4.4.2 Algorithms

A factorization as given in Theorem 4.27 can be obtained by modifying Alg. 4.8: if a definition of an l_{ij} is not possible because $a_{jj}^{(j)} = 0$, then a row j' from the rows $\{j+1, \ldots, n\}$ is chosen with $a_{jj'}^{(j)} \neq 0$ (this is possible since otherwise $\mathbf{A}$ is rank deficient). One interchanges rows j and j' and continues with Alg. 4.8.
Mathematically, it is immaterial, which row j' is chosen. Numerically, one typically chooses the row j' such that the corresponding entry $a_{jj'}^{(j)}$ is the largest (in absolute value) from the set $\{a_{jJ}^{(j)} \; J = j+1, \ldots, n\}$. This is called *partial pivoting*.
To formalize the procedure, we need the concept of permutation matrices:

Definition 4.29 *Let* $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ *be a permutation*[5]*. Then,*

$$\mathbf{P}_\pi := \begin{pmatrix} e_{\pi(1)} & , & \ldots & , & e_{\pi(n)} \end{pmatrix}$$

denotes the corresponding permutation matrix.

Theorem 4.30 *Let* $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ *be a permutation. Then:*

(i) $\mathbf{P}_\pi \, e_i := e_{\pi(i)} \qquad \forall i$

(ii) $\mathbf{P}_\pi^{-1} = \mathbf{P}_\pi^\top$

(iii) $\mathbf{P}_\pi \mathbf{A}$ *is obtained from* $\mathbf{A}$ *by row permutation: the* i*-th row of* $\mathbf{A}$ *becomes the* $\pi(i)$*-th row of* $\mathbf{P}_\pi \mathbf{A}$*. Put differently:* $(\mathbf{P}_\pi \mathbf{A})_{i,:} = \mathbf{A}_{\pi^{-1}(i),:}$ *or, still equivalently,* $(\mathbf{P}_{\pi^{-1}} \mathbf{A})_{i,:} = \mathbf{A}_{\pi(i),:}$*.*

(iv) $\mathbf{A}\mathbf{P}_\pi$ *is obtained from* $\mathbf{A}$ *by column permutation:* $(\mathbf{A}\mathbf{P}_\pi)_{:,i} = \mathbf{A}_{:,\pi(i)}$*.*

Proof: Exercise. (Prove (ii), then (iv). Finally (iii) using (ii) and transposes.) □

In practice, the LU-factorization of $\mathbf{A}$ with (row) pivoting operates directly on the matrix $\mathbf{A}$, i.e., overwrites the matrix $\mathbf{A}$ and the row permutations are not done explicitly but implicitly with pointers. This leads to:

Algorithm 4.31 (Gaussian elimination with row pivoting) *Input: invertible* $\mathbf{A} \in \mathbb{R}^{n \times n}$
Output: factorization $\mathbf{PA} = \mathbf{LU}$*, where* $\mathbf{A}$ *is overwritten by* $\mathbf{U}$*:*
$u_{ij} = a_{\pi(i),j}$ *and* $\mathbf{P} = \mathbf{P}_\pi^{-1} = \mathbf{P}_\pi^\top$ *is implicitly given by the vector* π

$\pi := (1, 2, \ldots, n)$
for $k = 1 : (n-1)$ ***do***
  seek $p \in \{k, \ldots, n\}$ *s.t.* $|a_{pk}| \geq |a_{ik}| \qquad \forall i \geq k$
  interchange k*-th and* p*-th entry of vector* π
  for $i = (k+1) : n$ ***do***
    $l_{\pi(i),k} := \frac{a_{\pi(i),k}}{a_{\pi(k),k}}$
    for $j = (k+1) : n$ ***do***
      $a_{\pi(i),j} := a_{\pi(i),j} - l_{\pi(i),k} \, a_{\pi(k),j}$
    end for
  end for
end for

Theorem 4.32 *Let* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *be invertible. Then Algorithm 4.31 yields a factorization* $\mathbf{LU} = \mathbf{PA}$*, where* $\mathbf{L}$ *satisfies* $|l_{ij}| \leq 1 \ \forall i, j$*.* $\mathbf{P}$ *is a permutation matrix.*

Remark 4.33 *The* `matlab`/`python` *commands to compute LU-factorization typically return matrices* $\mathbf{L}$*,* $\mathbf{U}$*,* $\mathbf{P}$ *of a factorization* $\mathbf{LU} = \mathbf{PA}$ *and perform (at least some) pivoting.*

Exercise 4.34 *Given a factorization* $\mathbf{LU} = \mathbf{PA}$*, determine the solution* $\mathbf{x}$ *of* $\mathbf{Ax} = \mathbf{b}$*.*

[5]That is, π is a bijection

4.4.3 Numerical difficulties: choice of the pivoting strategy

Alg. 4.31 selected the largest element from among the possible pivot elements. Why this is a good strategy becomes more clear when one studies the case that the pivot element is non-zero but small as in the following example.
Consider for small ε the matrix

$$\mathbf{A} = \begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix}$$

Its LU-factorization is

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ \varepsilon^{-1} & 1 \end{pmatrix} \begin{pmatrix} \varepsilon & 1 \\ 0 & 1-\varepsilon^{-1} \end{pmatrix}$$

Let now $\varepsilon = 10^{-20}$. In typical floating point arithmetic (16 digits) one therefore expects this to be realized as with approximate factors $\widehat{\mathbf{L}}$, $\widehat{\mathbf{U}}$ given by

$$\widehat{\mathbf{L}} = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix}, \qquad \widehat{\mathbf{U}} = \begin{pmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{pmatrix}$$

If one performs (in `matlab`, say) the forward and back substitution for the linear system

$$\widehat{\mathbf{L}}\widehat{\mathbf{U}}\mathbf{x} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

one obtains $\mathbf{x} = (0, 1)$ whereas the correct solution of the original problem is (up to machine precision) $\mathbf{x} = (-1, 1)$. That is, the solution is completely inaccurate. In contrast, solving the row-pivoted problem yields the correct solution.
Rather than fully analyzing round errors for the solution of linear systems, let us give a heuristic, why the pivoting strategy is reasonable. Let us assume that the entries of the matrix $\mathbf{A}$ and the right-hand side vector $\mathbf{b}$ and the solution vector $\mathbf{x}$ are "moderate" in size. If small pivots are used, i.e., some $a_{kk}^{(k)}$ is small during Gaussian elimination, then one should expect the entries of $\mathbf{L}$ to be large (as in the above example). Hence, in the course of the forward or back substitution, one should expect large intermediate values. If the final result is again "moderate", then one should fear that this is achieved by subtracting numbers of similar size. That is, one should fear cancellation and thus loss of accuracy. In Alg. 4.31 the pivoting choice ensures that the entries of $\mathbf{L}$ are all bounded by 1, thus moderate. We stress that this is not an insurance against roundoff problems as the pivoting strategy does not control the size of the entries of $\mathbf{U}$. While this is possible ("full pivoting"), it is usually avoided due to the cost considerations.

slide 15 - pivoting

4.5 Condition number of a matrix A

An important quantity to assess the effect of errors in the data (i.e., the right-hand side $\mathbf{b}$ or the matrix $\mathbf{A}$) on the solution is the *condition number* (see (4.13) ahead). In order to define it, let $\|\cdot\|$ be a norm on $\mathbb{R}^n$. On the space of matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$, we define the *induced matrix norm* by

$$\|\mathbf{A}\| := \max_{0\neq\mathbf{x}\in\mathbb{R}^n} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{4.12}$$

Exercise 4.35 *Show:*

1. *If $\|\cdot\| = \|\cdot\|_\infty$, then the induced matrix norm $\|\cdot\|_\infty$ is given by ("row sum norm")*

$$\|\mathbf{A}\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$$

2. *If $\|\cdot\| = \|\cdot\|_1$, then the induced matrix norm $\|\cdot\|_\infty$ is given by ("column sum norm")*

$$\|\mathbf{A}\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$$

3. *For $\|\cdot\|_2$ one has $\|\mathbf{A}\|_2^2 = \lambda_{max}(\mathbf{A}^\top \mathbf{A})$, where λ_{max} denotes the maximal eigenvalue.*

Exercise 4.36 *Prove: For $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$ there holds $\|\mathbf{AB}\| \le \|\mathbf{A}\|\|\mathbf{B}\|$.*

We study the effect of perturbing the right-hand side $\mathbf{b}$. We consider

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{A}(\mathbf{x}+\Delta\mathbf{x}) &= \mathbf{b}+\Delta\mathbf{b} \end{aligned}$$

In order to estimate $\Delta\mathbf{x}$ in terms of $\Delta\mathbf{b}$ we note $\mathbf{A}\Delta\mathbf{x} = \Delta\mathbf{b}$ as well as $\|\mathbf{b}\| = \|\mathbf{A}\mathbf{A}^{-1}\mathbf{b}\| \le \|\mathbf{A}\|\|\mathbf{A}^{-1}\mathbf{b}\|$ so that

$$\begin{aligned} \text{absolute error: } & \|\Delta\mathbf{x}\| = \|\mathbf{A}^{-1}\Delta\mathbf{b}\| \le \|\mathbf{A}^{-1}\|\|\Delta\mathbf{b}\|, \\ \text{relative error: } & \frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\|\mathbf{A}^{-1}\Delta\mathbf{b}\|}{\|\mathbf{A}^{-1}\mathbf{b}\|} \le \frac{\|\mathbf{A}^{-1}\|\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|/\|\mathbf{A}\|} = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}. \end{aligned}$$

The quantity

$$\kappa(\mathbf{A}) := \|\mathbf{A}\|\|\mathbf{A}^{-1}\| \tag{4.13}$$

is called the *condition number* of the matrix $\mathbf{A}$ (with respect to the norm $\|\cdot\|$). It measures how a perturbation in the right-hand side $\mathbf{b}$ could impact the solution of the linear system.

Remark 4.37 *In floating point arithmetic, a rounding error $\|\Delta\mathbf{b}\|/\|\mathbf{b}\| = O(\varepsilon)$ with machine precision ε is typically unavoidable. Thus, $\varepsilon\kappa(\mathbf{A})$ indicates of the level of accuracy that could at best be expected.*

Remark 4.38 *The condition number also appears when one assesses the impact of perturbations of matrix entries. One has (see, e.g., the class notes of Schranz-Kirlinger)*

$$\frac{\|\Delta\mathbf{x}\|}{\|\widetilde{\mathbf{x}}\|} \le \kappa(\mathbf{A})\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}$$

where $\mathbf{x}$ and $\widetilde{\mathbf{x}}$ solve

$$\mathbf{Ax} = \mathbf{b}, \qquad (\mathbf{A}+\Delta\mathbf{A})\widetilde{\mathbf{x}} = \mathbf{b}$$

4.6 QR-factorization (CSE)

The basic idea above to solve linear systems is to write $\mathbf{Ax} = \mathbf{b}$ as $\mathbf{LUx} = \mathbf{b}$ since the linear systems $\mathbf{Ly} = \mathbf{b}$ and $\mathbf{Ux} = \mathbf{y}$ are easily solved by forward and back substitution. We now present a further factorization $\mathbf{A} = \mathbf{QR}$, the $\mathbf{QR}$-*factorization*, where the factors $\mathbf{Q}$ and $\mathbf{R}$ are such that the linear systems $\mathbf{Qy} = \mathbf{b}$ and $\mathbf{Rx} = \mathbf{y}$ are easily solved. Although computing the $\mathbf{QR}$-factorization is about twice as expensive as the LU-factorization it is the preferred method for ill-conditioned matrices $\mathbf{A}$.

4.6.1 orthogonal matrices

Definition 4.39 *A matrix $Q \in \mathbb{R}^{n\times n}$ is* ***orthogonal****, if $\mathbf{Q}^\top\mathbf{Q} = I$. $\mathcal{O}_n$ denotes the set of orthogonal $n \times n$-matrices.*

Theorem 4.40 *(i) The product of two orthogonal matrices is orthogonal; the inverse of an orthogonal matrix is orthogonal.*[6]

(ii) If $\mathbf{Q} \in \mathcal{O}_{n-k}$, then $\begin{pmatrix} I_k & 0 \\ 0 & \mathbf{Q}_{n-k} \end{pmatrix} \in \mathcal{O}_n$

(iii) $\mathbf{Q} \in \mathcal{O}_n \;\Rightarrow\; \|\mathbf{Qx}\|_2 = \|\mathbf{x}\|_2 \;\; \forall \mathbf{x} \in \mathbb{R}^n$ ⟦ *that is, $\mathbf{Q}$ preserves length/euclidean norm—it is this property that makes orthogonal matrices so attractive in numerics.* ⟧

Proof: *Exercise.* □

Remark 4.41 (multiplication by $\mathbf{Q} \in \mathcal{O}_n$ is numerically stable) *Consider relative errors:*

$$\frac{\|\mathbf{Q}(\mathbf{x}+\Delta\mathbf{x}) - \mathbf{Qx}\|_2}{\|\mathbf{Qx}\|_2} = \frac{\|\mathbf{Q}\Delta\mathbf{x}\|_2}{\|\mathbf{Qx}\|_2} = \underbrace{1}_{\substack{\text{“amplification” factor} \\ \text{for rel. error}}} \cdot \frac{\|\Delta\mathbf{x}\|_2}{\|\mathbf{x}\|_2}$$

Exercise 4.42 *Check that the Gram-Schmidt orthogonalization process for a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ produces an upper triangular matrix $\mathbf{R}$ and an orthogonal matrix $\mathbf{Q}$ with $\mathbf{AR} = \mathbf{Q}$. Hence, for invertible $\mathbf{R}$ (i.e., invertible $\mathbf{A}$), Gram-Schmidt provides a $\mathbf{QR}$-factorization of $\mathbf{A}$.*

4.6.2 QR-factorization by Householder reflections

Definition 4.43 *Let $m \geq n$.*

(i) $\mathbf{R} \in \mathbb{R}^{m\times n}$ *is a* ***generalized upper triangular matrix*** *if $r_{ij} = 0 \;\; \forall i > j$, i.e.,*

$$\mathbf{R} = \begin{pmatrix} \widetilde{\mathbf{R}} \\ 0 \end{pmatrix} \qquad \text{with } \widetilde{R} \in \mathcal{U}_m.$$

$\mathcal{U}_m$ denotes the set of $m \times m$ upper triangular matrices.

[6] In other words: $\mathcal{O}_n$ is a group with respect to matrix multiplication.

(ii) A factorization $\mathbf{A} = \mathbf{QR}$ *of a matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *with an orthogonal* $\mathbf{Q} \in \mathcal{O}_m$ *and a generalized upper triangular matrix* $\mathbf{R}$ *is called a* **QR**-factorization *of* $\mathbf{A}$.

Theorem 4.44 *Let* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *be invertible. Then:* $\mathbf{A}$ *has a* **QR***-factorization. It is unique if one fixes the signs of the diagonal entries* r_{ii} *of* $\mathbf{R}$.

Proof: The existence follows from the explicit construction in Alg. 4.49 below. For definiteness' sake, assume that the signs of diagonal entries are fixed to be positive: $r_{ii} > 0$. Let $\mathbf{QR} = \tilde{\mathbf{Q}}\tilde{\mathbf{R}} = \mathbf{A}$ be two QR-factorizations. Since $\mathbf{A}$ is invertible, so is $\mathbf{R}$. Hence, $\mathbf{Q}' := \tilde{\mathbf{Q}}^\top\mathbf{Q} = \tilde{\mathbf{R}}\mathbf{R}^{-1} =: \mathbf{R}'$. We have to $\mathbf{R}'$ is upper triangular (as the product of two upper triangular matrices) and an orthogonal matrix (since $\mathbf{Q}'$ is orthogonal as the product of two orthogonal matrices). Thus, the columns of $\mathbf{R}'$ are orthogonal and by using that $\mathbf{R}'$ is upper triangular, checking inner products of columns of $\mathbf{R}'$ reveals that $\mathbf{R}'$ is a diagonal matrix (e.g., $0 = (\mathbf{R}'_{:,1})^\top(\mathbf{R}'_{:,2}) = r'_{11}r'_{12}$ and $r_{11} \neq 0$). Diagonal matrices that are orthogonal have $+1$ or -1 on the diagonal. One can show (e.g., by induction on n) that $(\mathbf{R})_{ii}^{-1} = 1/r_{ii}$ and it is not difficult to see that the diagonal entries of $(\tilde{\mathbf{R}}\mathbf{R}^{-1})_{ii} = \tilde{\mathbf{R}}_{ii}(\mathbf{R}^{-1})_{ii} = \tilde{r}_{ii}/r_{ii}$. Since, by assumption, $\tilde{r}_{ii}$ and r_{ii} have the same sign, $\tilde{r}_{ii}/r_{ii} = 1$. Hence, $\tilde{\mathbf{Q}}^\top\mathbf{Q} = \mathbf{Q}' = \tilde{\mathbf{R}}\mathbf{R}^{-1} = \mathrm{I}$. That is, $\tilde{\mathbf{Q}} = \mathbf{Q}$ and $\tilde{\mathbf{R}} = \mathbf{R}$. □

The **QR**-factorization of $\mathbf{A}$ is schematically obtained as follows:

$$\mathbf{A} =: \mathbf{A}^{(0)} = \begin{pmatrix} * & \dots & * \\ \vdots & & \vdots \\ * & \dots & * \end{pmatrix} \xrightarrow{\mathbf{Q}_1\in\mathcal{O}_n} \mathbf{Q}_1\mathbf{A}^{(0)} =: A^{(1)} = \begin{pmatrix} * & \dots & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{pmatrix}$$

$$\mathbf{A}^{(1)} = \begin{pmatrix} * & \dots & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{pmatrix} \xrightarrow{\mathbf{Q}_2\in\mathcal{O}_n} \mathbf{Q}_2A^{(1)} =: \mathbf{A}^{(2)} = \begin{pmatrix} * & \dots & \dots & \dots & * \\ 0 & * & \dots & \dots & * \\ \vdots & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & \dots & * \end{pmatrix}$$

$$\mathbf{A}^{(2)} = \begin{pmatrix} * & \dots & \dots & \dots & * \\ 0 & * & \dots & \dots & * \\ \vdots & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & \dots & * \end{pmatrix} \xrightarrow{\mathbf{Q}_3\in\mathcal{O}_n} \dots \xrightarrow{Q_{n-1}\in\mathcal{O}_n} \mathbf{A}^{(n-1)} = \begin{pmatrix} * & \dots & \dots & * \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & * \end{pmatrix}$$

Then: $\mathbf{Q}_{n-1}\mathbf{Q}_{n-2}\dots\mathbf{Q}_1\mathbf{A} = \mathbf{R}$.
That is, the sought **QR**-factorization is $\mathbf{A} = \mathbf{Q}_1^\top\dots\mathbf{Q}_{n-1}^\top\mathbf{R}$.
The $\mathbf{Q}_i$ are constructed using so-called *Householder reflections*, which are "elementary" orthogonal transformations:

Definition 4.45 (Householder reflections) *Given* $\mathbf{v} \in \mathbb{R}^n$ *with* $\|\mathbf{v}\|_2 = 1$ *the matrix* $\mathbf{H} = I - 2\mathbf{v}\mathbf{v}^\top$ *is called the induced* Householder reflection.

Lemma 4.46 (properties of Householder reflections) *Let* $\mathbf{v} \in \mathbb{R}^n$ *with* $\|\mathbf{v}\|_2 = 1$. *Then the matrix* $\mathbf{H} = I - 2\mathbf{v}\mathbf{v}^\top$ *satisfies:*

(i) $\mathbf{H}$ *is symmetric, i.e.,* $\mathbf{H}^\top = \mathbf{H}$*)*

(ii) $\mathbf{H}$ *is an involution (*$\mathbf{H}^2 = I$*)*

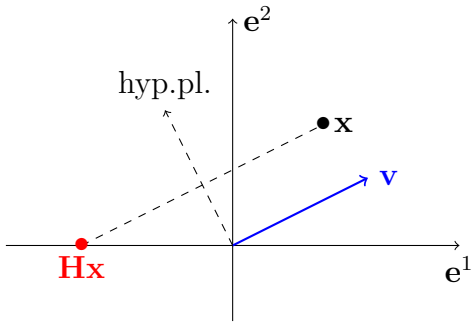


Figure 4.6: The Householder $\mathbf{H}$ with $\mathbf{Hx} \parallel \mathbf{e}^1$; cf. proof of Lemma 4.47

(iii) $\mathbf{H}$ *is orthogonal* ($\mathbf{H}^\top\mathbf{H} = I$)

Proof: Exercise.
The geometric interpretation of $\mathbf{H}$ is that the linear map represented by $\mathbf{H}$ is a reflection at the hyperplane $\{\mathbf{x} \in \mathbb{R}^n | \mathbf{v}^\top\mathbf{x} = 0\}$. □

Lemma 4.47 *Let* $\mathbf{x} \in \mathbb{R}^n \setminus \{0\}$ *and* $\mathbf{e}^1 = (1, 0, \ldots, 0)^\top \in \mathbb{R}^n$. *Then* $\exists\ \mathbf{Q} \in \mathcal{O}_n$ *with* $\mathbf{Qx} \in \operatorname{span}\{\mathbf{e}^1\}$. *In particular:*

(i) if $\mathbf{x} \parallel \mathbf{e}^1$, *then* $\mathbf{Q} := I$

(ii) if $\mathbf{x} \not\parallel \mathbf{e}^1$, *then set*[7] $\lambda = \operatorname{sign}\mathbf{x}_1 \|\mathbf{x}\|_2$. *Then* $\mathbf{H} = I - 2\mathbf{v}\mathbf{v}^\top$ *with* $\mathbf{v} = \frac{\mathbf{x}+\lambda\mathbf{e}^1}{\|\mathbf{x}+\lambda\mathbf{e}^1\|_2}$ *has the desired property* $\mathbf{Hx} = -\lambda\mathbf{e}^1$.

Proof:

(i) ✓

(ii)

$$\mathbf{x} + \lambda\mathbf{e}^1 = \begin{pmatrix} \mathbf{x}_1 + (\operatorname{sign}\mathbf{x}_1)\|\mathbf{x}\|_2 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{pmatrix}$$

$$\|\mathbf{x} + \lambda\mathbf{e}^1\|_2^2 = \big(|\mathbf{x}_1| + \|\mathbf{x}\|_2\big)^2 + \sum_{i=2}^{n} \mathbf{x}_i^2 = 2\|\mathbf{x}\|_2^2 + 2|\mathbf{x}_1|\|\mathbf{x}\|_2 \neq 0$$

$$\big(I - 2\mathbf{v}\mathbf{v}^\top\big)\mathbf{x} = \mathbf{x} - 2\frac{\mathbf{x} + \lambda\mathbf{e}^1}{2\|\mathbf{x}\|_2^2 + 2|\mathbf{x}_1|\|\mathbf{x}\|_2} \underbrace{\Big(\mathbf{x} + \operatorname{sign}\mathbf{x}_1\|\mathbf{x}\|_2\mathbf{e}^1\Big)^\top\mathbf{x}}_{\|\mathbf{x}\|_2^2 + |\mathbf{x}_1|\|\mathbf{x}\|_2} = -\lambda\mathbf{e}^1$$

□

[7] we assume $\operatorname{sign}\mathbf{x}_1 \neq 0$. If $\mathbf{x}_1 = 0$, then select $\operatorname{sign}\mathbf{x}_1$ arbitrarily as 1 or -1.

Remark 4.48 (choice of λ) *Householder reflections* $\mathbf{H}$ *with* $\mathbf{Hx} \in \operatorname{span}\{\mathbf{e}^1\}$ *are not unique. For example,* $\mathbf{v} = \frac{\mathbf{x}+\lambda\mathbf{e}^1}{\|\mathbf{x}+\lambda\mathbf{e}^1\|_2}$ *with* $\lambda = -\big(\operatorname{sign}\mathbf{x}_1\big)\|\mathbf{x}\|_2$ *is also possible. This choice, however, is* numerically unstable *if* $\mathbf{x}$ *and* $\mathbf{e}^1$ *are nearly parallel, i.e.,* $|\mathbf{x}_1| \approx \|\mathbf{x}\|_2$*. Then cancellation occurs when computing* $\mathbf{x} + \lambda\mathbf{e}^1$*.*

Algorithm 4.49 (Householder QR-factorization) *Input:* $\mathbf{A} \in \mathbb{R}^{m\times n}$, $m \geq n$, $\operatorname{rank}(\mathbf{A}) = n$
Output: factorization $\mathbf{A} = \mathbf{QR}$ *with* $\mathbf{Q} \in \mathcal{O}_m$ *and* $\mathbf{R} \in \mathbb{R}^{m\times n}$ *generalized upper triangular matrix.* $\mathbf{Q}$ *is given implicitly as* $\mathbf{Q}^{-1} = \mathbf{Q}_{n-1}\cdots\mathbf{Q}_1$ ⟦ *note:* $\mathbf{Q} = \mathbf{Q}_1\cdots\mathbf{Q}_{n-1}$ *since the* $\mathbf{Q}_i$ *are symmetric, i.e.,* $\mathbf{Q}_i^\top = \mathbf{Q}_i$ ⟧

- *set* $\mathbf{A}^{(0)} := \mathbf{A}$ *and select* $\mathbf{Q}_1$ *as a Householder reflection s.t.* $\mathbf{QA}^{(0)}_{:,1} \parallel \mathbf{e}^1 \in \mathbb{R}^m$
- *"Householder step":*

$$\mathbf{A}^{(1)} \;:=\; \mathbf{Q}_1\mathbf{A}^{(0)} = \begin{pmatrix} a^{(1)}_{11} & a^{(1)}_{12} & \dots & a^{(1)}_{1n} \\ 0 & a^{(1)}_{22} & \dots & a^{(1)}_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & a^{(1)}_{n2} & \dots & a^{(1)}_{nn} \end{pmatrix}$$

- *select* $\widetilde{\mathbf{Q}}_1$ *as a Householder reflection s.t.* $\widetilde{\mathbf{Q}}_1\mathbf{A}^{(1)}_{[2:m],2} \parallel \mathbf{e}^1 \in \mathbb{R}^{m-1}$
- *set*

$$\mathbf{Q}_2 = \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widetilde{\mathbf{Q}_1} \\ & \end{array}\right)$$

- *"Householder step":*

$$\begin{pmatrix} a^{(1)}_{11} & a^{(1)}_{12} & \dots & a^{(1)}_{1n} \\ 0 & a^{(1)}_{22} & \dots & a^{(1)}_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & a^{(1)}_{n2} & \dots & a^{(1)}_{nn} \end{pmatrix} = \mathbf{A}^{(1)} \;\longrightarrow \mathbf{Q}_2\mathbf{A}^{(1)} =: \mathbf{A}^{(2)} = \begin{pmatrix} a^{(1)}_{11} & \dots & \dots & \dots & a^{(1)}_{1n} \\ 0 & a^{(2)}_{22} & \dots & \dots & a^{(2)}_{2n} \\ \vdots & 0 & a^{(2)}_{33} & \dots & a^{(2)}_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a^{(2)}_{n3} & \dots & a^{(2)}_{nn} \end{pmatrix}$$

- *analogously, the next steps are:*

$$\begin{pmatrix} a_{11}^{(1)} & \dots & \dots & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \dots & \dots & a_{2n}^{(2)} \\ \vdots & 0 & a_{33}^{(2)} & \dots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \dots & a_{nn}^{(2)} \end{pmatrix} = \mathbf{A}^{(2)} \longrightarrow \left(\begin{array}{cc|c} 1 & 0 & \\ 0 & 1 & \\ \hline & & \widetilde{\mathbf{Q}_2} \\ & & \end{array}\right) \mathbf{A}^{(2)} \longrightarrow \dots$$

$$\dots \longrightarrow \mathbf{A}^{(n)} = \begin{pmatrix} * & \dots & \dots & * \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & * \\ 0 & \dots & \dots & 0 \\ \vdots & & & \vdots \\ 0 & \dots & \dots & 0 \end{pmatrix}$$

Remark 4.50 *(i) see literature (e.g., the book by Golub–van Loan) for a precise formulation*

(ii) Algorithm 4.49 does not stop prematurely since $\operatorname{rank} A = n$ *⟦ if a column* $(a_{k+1,k}^{(k)}, \dots, a_{k+1,n}^{(k)})^\top$ *is zero, then* $\mathbf{A}$ *cannot have full column rank n! ⟧*

(iii) <u>*cost:*</u> *For* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *the algorithm requires* $\frac{4}{3}n^3$ *arithmetic operations → twice as expense as* **LU***-factorization and 4 times as expensive as a Cholesky decomposition.*

(iv) <u>*storage:*</u> $\mathbf{Q}$ *is typically not stored explicitly but merely the Householder vectors are stored. One possibility of storing the factorization in place of* $\mathbf{A}$*:*

- *store the entries* r_{ij} *,* $j > i$ *in place of* a_{ij}
- *store the k-th Householder vector* $\mathbf{w}_k \in \mathbb{R}^{m+j-k}$ *in place of* a_{ik} *,* $i \geq k$
- *store the* r_{ii} *separately*

Remark 4.51 *Although the* **QR***-factorization is more expensive than the* **LU***-factorization, it is employed for its better numerical stability properties if the condition number of* $\mathbf{A}$ *is large.*
slide 16a - Comparison LU,QR

4.6.3 QR-factorization with pivoting

Analogously to **LU**-factorizations with pivoting one can perform **QR**-factorizations with pivoting by constructing factorizations $\mathbf{QR} = \mathbf{AP}$ for a permutation matrix $\mathbf{P}$. This is useful, for example, to treat the case when $m \geq n$ and $\operatorname{rank} \mathbf{A} < n$ ("rank-deficient case").

Procedure:

$$\mathbf{A}^{(0)} \xrightarrow[\mathbf{A}^{(0)} \text{ with the largest } \|\cdot\|_2 \text{ norm to the first column}]{\mathbf{P}_1 = \text{ permutation matrix that moves the column of}} \widetilde{\mathbf{A}}^{(1)} := \mathbf{A}^{(0)}\mathbf{P}_1 \xrightarrow{\text{Householder}} \mathbf{A}^{(1)} := \mathbf{Q}_1\widetilde{\mathbf{A}}^{(1)}$$

$$\mathbf{A}^{(1)} \xrightarrow[p\geq 2 \text{ and } \|\mathbf{A}^{(1)}_{[2:n],p}\|_2 = \max_{i\geq 2}\|\mathbf{A}^{(1)}_{[2:n],i}\|_2]{\mathbf{P}_2\text{: exchange columns 2 and } p \text{ where}} \widetilde{\mathbf{A}}^{(2)} := \mathbf{A}^{(1)}\mathbf{P}_2 \xrightarrow{\text{Householder}} \mathbf{A}^{(2)} := \mathbf{Q}_2\widetilde{\mathbf{A}}^{(2)}$$

$$\mathbf{A}^{(2)} \longrightarrow \ldots \longrightarrow \mathbf{A}^{(k)} = \begin{pmatrix} * & \cdots & \cdots & \cdots & \cdots & \cdots & * \\ 0 & \ddots & & & & & \vdots \\ \vdots & \ddots & \ddots & & & & \vdots \\ \vdots & & \ddots & * & \cdots & \cdots & * \\ \vdots & & & 0 & \cdots & \cdots & 0 \\ \vdots & & & & & & \vdots \\ \vdots & & & & & & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \end{pmatrix} = \text{final form}$$

termination:

- The procedure termines if the "remaining matrix" $\left(a_{ij}^{(k)}\right)_{i,j,\geq k+1}$ is the null matrix. Then $\operatorname{rank}\mathbf{A} = k$
- The diagonal entries r_{ii} satisfy $|r_{11}| \geq |r_{22}| \geq \cdots \geq |r_{kk}| > 0$ (exercise: why?). If the submatrix $\mathbf{A}^{(k)}_{[k'+1:\text{end}],[k'+1:\text{end}]}$ has small norm, e.g., $\|\mathbf{A}^{(k)}_{[k'+1:\text{end}],[k'+1:\text{end}]}\|_2 \leq \varepsilon_{\text{mach}}\|\mathbf{A}^{(k)}\|_2$ with $\varepsilon_{\text{mach}}$ being on the order of machine precision, then the rank of $\mathbf{A}$ is effectively k'.

4.6.4 Givens rotations

The application of a single Householder reflection affects many entries of the matrix. Sometimes, it is useful to work with orthogonal matrices that introduce zeros in a matrix in more selective way, i.e., affect rather few entries at the same time. *Givens rotations* are then typically employed. We mention that, for *full* matrices, a **QR**-factorization using Givens rotations is (by a factor) more expensive than with Householder reflections.
For $\theta \in [0, 2\pi)$ set $c := \cos\theta$, $s := \sin\theta$. Then the *Givens rotation* $\mathbf{G}(i, j, \theta)$ with $i \neq j$ is

defined as

$$\mathbf{G}(i,j,\theta) := \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & c & & & +s & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & -s & & & & c & \\ & & & & & & & & 1 \\ & & & & & & & & & \ddots \end{pmatrix}$$

Geometrically, $\mathbf{G}(i,j,\theta)$ is a rotation by an angle θ in the two-dimensional plane $\text{span}\{e_i, e_j\}$. Thus is an orthogonal matrix. We have

Lemma 4.52 *Given $i \neq j$, $\theta \in [0, 2\pi)$ abbreviate*

$$\widehat{\mathbf{G}} := \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

Then:

(i) $\mathbf{G}(i,j,\theta)$ *is orthogonal*

(ii) $\mathbf{AG}$ *differs from* $\mathbf{A}$ *only in columns i and j and these are linear combinations of the columns i, j of* $\mathbf{A}$*:*

$$(\mathbf{AG})(:,[i,j]) = \mathbf{A}(:,[i,j])\widehat{\mathbf{G}}$$

(iii) $\mathbf{G}^\top\mathbf{A}$ *differs from* $\mathbf{A}$ *only in rows i and j and these are linear combinations of the rows i, j of* $\mathbf{A}$*:*

$$(\mathbf{G}^\top\mathbf{A})([i,j],:) = \widehat{\mathbf{G}}^\top\mathbf{A}([i,j],:)$$

(iv) Let $i \neq j$ and $i' \neq i$. Then there is a Givens rotation $\mathbf{G}(i,i',\theta)$ such that $(\mathbf{G}^\top\mathbf{A})_{ij} = 0$.

Proof: We only show (iv). For that, we note

$$\begin{aligned} (\mathbf{G}^\top\mathbf{A})([i,i'],[i,j]) &= \widehat{\mathbf{G}}^\top\mathbf{A}([i,i'],[i,j]) = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}\begin{pmatrix} \mathbf{A}_{ii} & \mathbf{A}_{ij} \\ \mathbf{A}_{i'i} & \mathbf{A}_{i'j} \end{pmatrix} \\ &= \left(\begin{array}{c|c} * & c\mathbf{A}_{ij} - s\mathbf{A}_{i'j} \\ \hline * & * \end{array}\right). \end{aligned}$$

Hence, the requirement $(\mathbf{G}^\top\mathbf{A})_{ij} = 0$ implies that θ should be chosen such that $s\mathbf{A}_{i'j} = c\mathbf{A}_{ij}$.
1. case: $\mathbf{A}_{ij} = 0$: select $c = 1$, $s = 0$.
2. case: $\mathbf{A}_{ij} \neq 0$: select $\theta \in (0, \pi)$ as the solution of $\cot\theta = -\frac{\mathbf{A}_{i'j}}{\mathbf{A}_{ij}}$. □

Lemma 4.52 informs us that one could also compute a **QR**-factorization of **A** using Givens rotations. We sketch the procedure:

$$\mathbf{A} = \begin{pmatrix} * & \dots & * \\ * & \dots & * \\ \vdots & & \vdots \\ * & \dots & * \end{pmatrix} \xrightarrow{\mathbf{G}(1,2)} \begin{pmatrix} * & \dots & \dots & * \\ 0 & * & \dots & * \\ * & * & \dots & * \\ \vdots & \vdots & & \vdots \\ * & * & \dots & * \end{pmatrix} \xrightarrow{\mathbf{G}(1,3)} \begin{pmatrix} * & \dots & \dots & * \\ 0 & * & \dots & * \\ 0 & * & & \vdots \\ * & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ * & * & \dots & * \end{pmatrix}$$

$$\to \cdots \xrightarrow{\mathbf{G}(1,n)} \begin{pmatrix} * & \dots & \dots & * \\ 0 & * & \dots & * \\ 0 & * & & \vdots \\ 0 & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{pmatrix} \xrightarrow{\mathbf{G}(2,3)} \begin{pmatrix} * & * & \dots & * \\ 0 & * & \dots & * \\ 0 & 0 & & \vdots \\ 0 & * & & \vdots \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{pmatrix} \to \cdots \xrightarrow{\mathbf{G}(n-1,n)} \begin{pmatrix} * & * & \dots & * \\ 0 & * & \dots & * \\ 0 & 0 & & \vdots \\ 0 & 0 & & \vdots \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & * \end{pmatrix}$$

The construction of a **QR**-factorization using Givens rotations is more expensive than the one using Householder reflections for full matrices. Givens rotations are typically employed if the matrix has already many zeros that one wishes to preserve by orthogonal transformations as the following example shows.

Example 4.53 *We compute the* **QR***-factorization of an upper Hessenberg matrix.*
(Upper) Hessenberg matrices $\mathbf{A}$ *are matrices with* $\mathbf{A}_{ij} = 0$ *for* $i > j + 1$ *(i.e., upper triangular and one additional subdiagonal may be nonzero). The basic step of the so-called* **QR***-algorithm for the (iterative) computation of eigenvalue of a matrix* $\mathbf{A}$ *is to compute the* **QR***-factorization of* $\mathbf{A}$ *and then compute the product of these factors in reverse order, i.e.,* $\mathbf{RQ}$*. We show that, if the matrix* $\mathbf{A}$ *is upper Hessenberg, then the product* $\mathbf{RQ}$ *is again upper Hessenberg.*
We compute the matrix $\mathbf{Q}^\top$ *as the product* $\mathbf{Q}^\top = \mathbf{G}(n-1,n)\cdots\mathbf{G}(2,3)\mathbf{G}(1,2)$ *of* $n-1$ *Givens rotation to annihilate the subdiagonal entries of* $\mathbf{A}$*. By construction* $\mathbf{Q}^\top\mathbf{A}$ *is thus upper triangular and is the factor* $\mathbf{R}$*. Next, we multiply from the right by* $\mathbf{Q}$*, i.e., we compute* $(\mathbf{Q}^\top\mathbf{A})\mathbf{Q} = (\mathbf{Q}^\top\mathbf{A})\mathbf{G}(1,2)^\top\mathbf{G}(2,3)^\top\cdots\mathbf{G}(n-1,n)^\top$*. One then checks the multiplication of* $(\mathbf{Q}^\top\mathbf{A})$ *by* $\mathbf{G}(1,2)$ *introduces an additional non-zero term in the* $(2,1)$ *position. The subsequent multiplication by* $\mathbf{G}(2,3)$ *introduces one in the* $(3,2)$ *position. Continuing in this fashion, we see that* $\mathbf{Q}^\top\mathbf{HQ}$ *is an (upper) Hessenberg matrix.*

5 Least Squares

goal: Given $\mathbf{A} \in \mathbb{R}^{m\times n}$ (i.e. $n \neq m$ is also allowed), $\mathbf{b} \in \mathbb{R}^m$, determine a "reasonable" solution to

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{5.1}$$

.

slide 17 - Least Squares examples

Remark 5.1 *For $m > n$, problem (5.1) is* overdetermined *so one cannot expect existence of a classical solution. For $m < n$, problem (5.1) is* underdetermined *so one cannot expect uniqueness.*

A reasonable approach is to minimize the residual $\mathbf{b} - \mathbf{A}\mathbf{x}$ in some norm of interest. The ℓ^2-norm $\|\cdot\|_2$ is particularly convenient as we will later see.

Definition 5.2 (least squares solution) $\mathbf{x} \in \mathbb{R}^n$ *is called a* least squares solution *of* $\mathbf{A}\mathbf{x} = \mathbf{b}$*, if it solves the following minimization problem:*

$$\textit{Find } \mathbf{x} \in \mathbb{R}^n \textit{ s.t.} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2 = \min\{\|\mathbf{b} - \mathbf{A}\mathbf{y}\|_2 \,|\, \mathbf{y} \in \mathbb{R}^n\} \tag{5.2}$$

Although a theory for general $\mathbf{A} \in \mathbb{R}^{m\times n}$ can be developed, we consider, in the interest of simplicity and brevity, in the present section only the case that $\mathbf{A}$ has full rank. That is, if $m \geq n$, then $\mathbf{A}$ has n linearly independent columns and if $n \geq m$, then $\mathbf{A}$ has m linearly independent rows.

Example 5.3 *The* `matlab` *command* `polyfit` *actually solves a least squares problem: given $n+1$ data points (x_i, y_i), $i = 0, \ldots, n$ and $m \leq n$, the coefficients $(a_j)_{j=0}^m$ of the polynomial $\pi(x) := \sum_{j=0}^m a_j x^j$ are found such that $\sum_{i=0}^n (\pi(x_i) - y_i)^2$ is minimized.* `matlab` *actually uses the technique based on the QR-factorization described below.*

5.1 Method of the normal equations

goal: derive a linear system of equations for the solution $\mathbf{x}$ of (5.2).
To that end, let $\mathbf{x} \in \mathbb{R}^n$ be the solution of (5.2) and let $\mathbf{v} \in \mathbb{R}^n$ be arbitrary but fixed. Define

$$\pi : \mathbb{R} \to \mathbb{R}$$
$$t \mapsto \|\mathbf{b} - \mathbf{A}(\mathbf{x} + t\mathbf{v})\|_2^2 = \|\mathbf{b} - \mathbf{A}\mathbf{x} - t\mathbf{A}\mathbf{v}\|_2^2 = \langle \mathbf{b} - \mathbf{A}\mathbf{x}, \mathbf{b} - \mathbf{A}\mathbf{x}\rangle_2 - 2t\langle \mathbf{b} - \mathbf{A}\mathbf{x}, \mathbf{A}\mathbf{v}\rangle_2 + t^2\|\mathbf{A}\mathbf{v}\|_2^2$$

π is (as a function of t) a quadratic polynomial and has, by the choice of $\mathbf{x}$, a minimum at $t = 0$ $\big($choose $\mathbf{y} = \mathbf{x} + t\mathbf{v}$ in (5.2)$\big)$. Hence,

$$0 = \pi'(0) = 2\langle \mathbf{b} - \mathbf{A}\mathbf{x}, \mathbf{A}\mathbf{v}\rangle_2 = 2\mathbf{v}^\top \mathbf{A}^\top (\mathbf{b} - \mathbf{A}\mathbf{x}).$$

Since $\mathbf{v} \in \mathbb{R}^n$ is arbitrary, we conclude that

$$0 = \mathbf{v}^\top \mathbf{A}^\top (\mathbf{b} - \mathbf{A}\mathbf{x}) \quad \forall \mathbf{v} \in \mathbb{R}^n \qquad \Rightarrow \quad \mathbf{A}^\top (\mathbf{b} - \mathbf{A}\mathbf{x}) = 0 \ \in \mathbb{R}^n.$$

Hence, $\mathbf{x}$ satisfies the *normal equations*

$$\mathbf{A}^\top \mathbf{A}\mathbf{x} = \mathbf{A}^\top \mathbf{b} \tag{5.3}$$

The normal equations (5.3) are a necessary condition for solutions $\mathbf{x}$ of (5.2). They are also sufficient: By tracing back the above steps, one observes that, if $\mathbf{x}$ solves (5.3) then for every fixed $\mathbf{v}$ the polynomial $t \mapsto \|\mathbf{b}-\mathbf{A}(\mathbf{x}+t\mathbf{v})\|_2^2$ has a minimum at $t=0$. Since also $\|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2^2 = \pi(0) \le \pi(1) = \|\mathbf{b}-\mathbf{A}(\mathbf{x}+\mathbf{v})\|_2^2$ for every $\mathbf{v}$, one concludes $\|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2^2 \le \|\mathbf{b}-\mathbf{A}\mathbf{y}\|_2^2 \quad \forall \mathbf{y} \in \mathbb{R}^n$. We have thus proved:

Theorem 5.4 *$\mathbf{x} \in \mathbb{R}^n$ solves (5.2), if and only if it solves (5.3).*

In many applications the square system (5.3) is solvable and thus an option to solve the least squares problem.

Theorem 5.5 *Let $m \ge n$ and let the columns of $\mathbf{A}$ be linearly independent. Then $\mathbf{A}^\top\mathbf{A}$ is an invertible matrix, and the unique solution of (5.3) is the unique solution of (5.2).*

Proof: If the columns of $\mathbf{A}$ are linearly independent, then $\mathbf{A}^\top\mathbf{A}\mathbf{y} = 0$ implies $\mathbf{y} = 0$ (Exercise!). Since $\mathbf{A}^\top\mathbf{A} \in \mathbb{R}^{n\times n}$ is a square matrix, it is invertible. Thus (5.3) is uniquely solvable. By Theorem 5.4 the problem (5.2) is uniquely solvable. ❑

5.2 Least squares using QR-factorizations

A problem often encountered when solving the least squares problem (5.2) using the normal equations (5.3) is that the matrix $\mathbf{A}^\top\mathbf{A}$ is ill-conditioned, i.e., $\kappa(\mathbf{A}^\top\mathbf{A})$ is very large. In many applications, therefore, one solves (5.2) using the QR-factorization of $\mathbf{A}$ in spite of the increased cost.[1]

5.2.1 QR-factorization

Definition 5.6 (orthogonal matrix) *A matrix $\mathbf{Q} \in \mathbb{R}^n$ is an* orthogonal matrix, *if $\mathbf{Q}^{-1} = \mathbf{Q}^\top$.*

Example 5.7 *In $\mathbb{R}^3$, reflections at a plane, rotations, or permutations matrices:*

$$\begin{pmatrix} 1 & & \\ & 1 & \\ & & -1 \end{pmatrix}, \qquad \begin{pmatrix} 1 & & \\ & \cos\theta & \sin\theta \\ & -\sin\theta & \cos\theta \end{pmatrix}, \qquad \begin{pmatrix} & & 1 \\ & 1 & \\ 1 & & \end{pmatrix}$$

Orthgonal matrices realize transformations of $\mathbb{R}^n$ that preserve a) (euklidean) length and b) angles:

Exercise 5.8 *Let $\mathbf{Q}$ be an orthogonal matrix. Show:*

[1] In the typically setting of $m \gg n$, the cost based on QR-factorization is $2mn^2$ versus mn^2 for the method based on the normal equations.

(a) $\|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ *for all* $\mathbf{x} \in \mathbb{R}^n$.

(b) $\mathbf{x}^\top \mathbf{y} = ((\mathbf{Q})\mathbf{x})^\top (\mathbf{Q}\mathbf{y})$ *for all* $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^n$.

(c) $\|\mathbf{Q}\|_2 = \|\mathbf{Q}^\top\|_2 = 1$ *and conclude* $\kappa(\mathbf{Q}) = 1$ *(with respect to* $\|\cdot\|_2$*).*

(d) The columns of $\mathbf{Q}$ *have length 1 and are pairwise orthogonal.*

We have

Theorem 5.9 *Let* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *with linearly independent columns. Then* $\mathbf{A}$ *can be written as* $\mathbf{A} = \mathbf{Q}\mathbf{R}$*, where* $\mathbf{Q} \in \mathbb{R}^{m\times m}$ *is orthogonal and* $\mathbf{R}$ *"upper triangular" matrix in the sense that* $(\mathbf{R}_{ij} = 0$ *for* $i > j$*.*

Proof: Applying the Gram-Schmidt orthogonalization process to the vectors $\mathbf{A}_{:,1}, \mathbf{A}_{:,2}, \ldots, \mathbf{A}_{:,n}$ yields the first n columns of $\mathbf{Q}$ as well as $\mathbf{R}$. The remaining $m - n$ columns of $\mathbf{Q}$ have to be selected such that the $\mathbf{Q}$ is orthogonal. □

Remark 5.10 *There are several algorithms to compute the* QR*-factorization of* $\mathbf{A}$*. Their cost is* $O(m^2 n)$*. In* `matlab`*,* QR*-factorization is realized with* `qr`*, in* `python` *as* `numpy.linalg.qr`*.*

Remark 5.11 *If* $m \geq n$ *and if* $\mathbf{A}$ *has full rank (i.e., the columns of* $\mathbf{A}$ *are linearly independent), then the first* n *columns of* $\mathbf{Q}$ *are an orthogonal basis of the range of* $\mathbf{Q}$*, i.e., the span of the first* n *columns of* $\mathbf{Q}$ *is the span of the columns of* $\mathbf{A}$*.*

5.2.2 Solving least squares problems with QR-factorization

Let $\mathbf{A} = \mathbf{Q}\mathbf{R}$ where $\mathbf{Q} \in \mathbb{R}^{m\times m}$ is orthogonal and $\mathbf{R} \in \mathbb{R}^{m\times n}$ is upper triangular. We assume $m \geq n$. We partition

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}^\star \\ 0 \end{pmatrix}, \qquad \mathbf{R}^* \in \mathbb{R}^{n\times n} \quad \text{upper triangular.}$$

If we assume that the columns of $\mathbf{A}$ are linearly independent, then the diagonal entries of the matrix $\mathbf{R}^*$ are non-zero, i.e., $\mathbf{R}^*$ is invertible (since the columns of $\mathbf{R}$ are linearly independent). We partition $\mathbf{Q}^\top \mathbf{b}$ as

$$\mathbf{Q}^\top \mathbf{b} = \begin{pmatrix} \mathbf{b}^* \\ \widetilde{\mathbf{b}} \end{pmatrix}, \qquad \mathbf{b}^* = (\mathbf{Q}^\top \mathbf{b})([1:n]) \in \mathbb{R}^n, \qquad \widetilde{\mathbf{b}} = (\mathbf{Q}^\top \mathbf{b})([n+1:m]) \in \mathbb{R}^{m-n},$$

We observe that for arbitrary $\mathbf{y}$ we have

$$\|\mathbf{A}\mathbf{y} - \mathbf{b}\|^2 = \|\mathbf{Q}\mathbf{R}\mathbf{y} - \mathbf{b}\|^2 = \|\mathbf{Q}(\mathbf{R}\mathbf{y} - \mathbf{Q}^\top \mathbf{b})\|^2 \overset{\text{Ex. 5.8}}{=} \|\mathbf{R}\mathbf{y} - \mathbf{Q}^\top \mathbf{b}\|^2 = \|\mathbf{R}^*\mathbf{y} - \mathbf{b}^*\|_2^2 + \|\widetilde{\mathbf{b}}\|_2^2.$$

This expression is minimized for the choice $\mathbf{y} = (\mathbf{R}^*)^{-1}\mathbf{b}^*$. We have thus arrived at the following way to compute the minimizer:

1. $[\mathbf{Q}, \mathbf{R}] = \mathtt{qr}(\mathbf{A})$

2. compute $\mathbf{Q}^\top\mathbf{b}$ and set $\mathbf{b}^* = (\mathbf{Q}^\top\mathbf{b})([1:n])$

3. solve $\mathbf{R}^*\mathbf{x} = \mathbf{b}^*$ with back substitution

Remark 5.12 *The QR-factorization can also be used in the case $m = n$ to solve a linear system $\mathbf{Ax} = \mathbf{b}$ with the following three steps:*

1. *compute the QR-factorization of $\mathbf{A}$*

2. *solve $\mathbf{Qy} = \mathbf{b}$ by computing $\mathbf{y} = \mathbf{Q}^\top\mathbf{b}$*

3. *solve $\mathbf{Rx} = \mathbf{y}$ by back substitution*

The cost is about twice that of the procedure using an LU-factorization. It is, however, preferred if $\kappa(\mathbf{A})$ is large.

Example 5.13 *Consider*

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ \varepsilon & 0 \\ 0 & \varepsilon \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 2 \\ \varepsilon \\ \varepsilon \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \mathbf{A}^\top\mathbf{A} = \begin{pmatrix} 1+\varepsilon^2 & 1 \\ 1 & 1+\varepsilon^2 \end{pmatrix}.$$

Note $\mathbf{Ax} = \mathbf{b}$ so that $\mathbf{x}$ is the exact solution of the least squares problem. We note $\kappa(\mathbf{A}^\top\mathbf{A}) = \frac{2}{\varepsilon^2} + 1$ so that $\mathbf{A}$ is ill-conditioned for small ε. In `matlab`*:*

```
>> e = 1e-7;
>> A = [1 1; e 0; 0 e]; b = [2;e;e];
>> x = (A'*A)\(A'*b) %solution using normal equations
x =
   1.011235955056180
   0.988764044943820
>> [Q,R] = qr(A) ;
>> bb=Q'*b ;
>> xx = R(1:2,1:2)\bb(1:2) %solution using QR-factorization
xx =
   1.000000000000000
   1.000000000000000
```

The method using the normal equations yields a solution with two digits of accuracy (consistent with $\kappa(\mathbf{A}^\top\mathbf{A}) \approx 10^{14}$) whereas the method based on the QR-factorization yields the correct solution.

5.3 Underdetermined systems

The system (5.1) is underdetermined if $m < n$. Let us assume that $\mathbf{A}$ has full rank, i.e., it has m linearly independent rows. Then (5.1) has a solution. However, the solution is not unique. One way to fix the solution is to seek the *minimum norm solution*, i.e., to find $\mathbf{x}^*$ such that

$$\|\mathbf{x}^*\|_2 = \min\{\|\mathbf{y}\|_2 \,|\, \mathbf{Ay} = \mathbf{b}\}.$$

A convenient tool to solve this minimization problem is the *singular value decomposition* (SVD) of $\mathbf{A}$.

$$\mathbf{A} = \begin{pmatrix} * & * \\ * & * \\ * & * \end{pmatrix} = \mathbf{U} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{pmatrix} \mathbf{V}^\top$$

Figure 5.1: structure of the SVD of an 3×2 matrix; $\mathbf{U}$, $\mathbf{V}$ are orthogonal

5.3.1 SVD

The SVD is a very important tool in the analysis of matrices. Without proof, we state its existence:

Theorem 5.14 (SVD) *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ (m, n arbitrary). Then there exist $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min\{m,n\}} \geq 0$ and orthogonal matrices $\mathbf{U} \in \mathbb{R}^{m\times m}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$, and $\Sigma \in \mathbb{R}^{m\times n}$ with $\Sigma_{ij} = \delta_{ij}\sigma_i$, $\sigma_i \geq 0$, such that*

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top, \tag{5.4}$$

The values σ_i are called the *singular values*, the columns of $\mathbf{U}$ the left singular vectors and the columns of the $\mathbf{V}$ the right singular vectors.
The SVD of a matrix $\mathbf{A}$ reveals many important properties of $\mathbf{A}$:

Exercise 5.15 *Let the singular values σ_i be sorted in descending order. Then:*

1. *Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_{\min\{m,n\}} = 0$. Then r is the* rank *of $\mathbf{A}$. (If all singular values are positive, then the matrix $\mathbf{A}$ has full rank).*
2. *The columns $\mathbf{U}(:, [1:r])$ form an orthogonal basis of the range $\operatorname{Im}\mathbf{A}$ of $\mathbf{A}$*
3. *The columns $\mathbf{V}(:, [r+1:n])$ form an orthogonal basis of the kernel of $\mathbf{A}$. The columns $\mathbf{V}(:, [1:r])$ form an orthogonal basis of $(\ker\mathbf{A})^\perp$, the orthogonal complement of the kernel of $\mathbf{A}$.*

Hint: *The range $\operatorname{Im}\mathbf{B}$ of a matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$ is defined as $\{\mathbf{Bx} \,|\, \mathbf{x} \in \mathbb{R}^n\}$. One way to define the* rank *of $\mathbf{B}$ is to set $\operatorname{rank}\mathbf{B} = \dim\operatorname{Im}\mathbf{B}$. Try to show that $\operatorname{Im}\Sigma = \operatorname{span}\{\mathbf{e}_1, \ldots, \mathbf{e}_r\}$. Convince yourself that also $\operatorname{Im}\Sigma\mathbf{V}^\top = \operatorname{span}\{\mathbf{e}_1, \ldots, \mathbf{e}_r\}$ and that therefore $\operatorname{Im}\mathbf{U}\Sigma\mathbf{V}^\top = \operatorname{span}\{\mathbf{U}(:, 1), \ldots, \mathbf{U}(:, r)\}$.*

Exercise 5.16 *Let $\mathbf{U}\Sigma\mathbf{V}^\top$ be the SVD of $\mathbf{A}$. Show: the eigenvalues of $\mathbf{A}^\top\mathbf{A}$ are the eigenvalues of the diagonal matrix $\Sigma^\top\Sigma$ and those of $\mathbf{AA}^\top$ the eigenvalues of the diagonal matrix $\Sigma\Sigma^\top$. What can you say about the eigenvectors of the matrices $\mathbf{A}^\top\mathbf{A}$ and $\mathbf{AA}^\top$?*

Remark 5.17 *In* `matlab`/`python`*, the SVD is available as* `svd`/`numpy.linalg.svd`.

For $r = \operatorname{rank}(\mathbf{A})$, we introduce the matrices

$$\widetilde{\mathbf{U}} = \mathbf{U}(:, [1:r]), \qquad \widetilde{\mathbf{V}} = \mathbf{V}(:, [1:r]), \qquad \widetilde{\Sigma} = \Sigma([1:r], [1:r]), \qquad \mathbf{V}' := \mathbf{V}(:, [r+1, n]).$$

We note that $\mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\Sigma}\widetilde{\mathbf{V}}^\top$ and $\widetilde{\Sigma}$ is invertible. This factorization of $\mathbf{A}$ is called the *reduced SVD*. We also note that the columns ov $\mathbf{V}'$ span the kernel of $\mathbf{A}$.

Remark 5.18 *A slightly different interpretation of the SVD is obtained by writing it as* $\mathbf{AV} = \mathbf{U}\Sigma$. *Writing* $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_n)$, $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_m)$, *this means* $\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i$, $i = 1, \ldots, r$, *where* $r = \operatorname{rank}(\mathbf{A})$. *That is, we have found pairwise orthogonal vectors* $\mathbf{v}_i$ *that are mapped under* $\mathbf{A}$ *to an orthogonal basis of the range of* $\mathbf{A}$.

Exercise 5.19 *Show:* $\mathbf{V}'(\mathbf{V}')^\top \mathbf{x}$ *is the orthogonal projection of* $\mathbf{x}$ *onto* $\operatorname{Ker}\mathbf{A}$. $\widetilde{\mathbf{V}}\widetilde{\mathbf{V}}^\top \mathbf{x}$ *is the orthogonal projection of of* $\mathbf{x}$ *onto* $(\operatorname{Ker}\mathbf{A})^\perp$. *Analogously,* $\widetilde{\mathbf{U}}\widetilde{\mathbf{U}}^\top \mathbf{x}$ *is the orthgonal projection of* $\mathbf{x}$ *onto* $\operatorname{Range}\mathbf{A}$.

5.3.2 Finding the minimum norm solution using the SVD

Let $m \leq n$ and assume (for simplicity) that $\mathbf{A}$ has full rank, i.e., $r = \operatorname{rank}(\mathbf{A}) = m$. Then $\widetilde{\mathbf{U}} = \mathbf{U}$ and the reduced SVD then takes the form $\mathbf{A} = \mathbf{U}\widetilde{\Sigma}\widetilde{\mathbf{V}}^\top$. We observe that

$$\widetilde{\mathbf{x}}^* := \widetilde{\mathbf{V}}\widetilde{\Sigma}^{-1}\mathbf{U}^\top \mathbf{b}$$

satisfies $\mathbf{A}\mathbf{x}^* = \mathbf{b}$ since

$$\mathbf{A}\widetilde{\mathbf{x}}^* = \mathbf{U}\widetilde{\Sigma}\widetilde{\mathbf{V}}^\top \widetilde{\mathbf{V}}\widetilde{\Sigma}^{-1}\mathbf{U}^\top \mathbf{b} = \mathbf{U}\widetilde{\Sigma}\widetilde{\Sigma}^{-1}\mathbf{U}^\top \mathbf{b} = \mathbf{U}\mathbf{U}^\top \mathbf{b} = \mathbf{b}$$

We note that every solution $\mathbf{x}$ of $\mathbf{A}\mathbf{x} = \mathbf{b}$ has the form $\mathbf{x} = \widetilde{\mathbf{x}}^* + \mathbf{V}'\mathbf{y}$ for a $\mathbf{y} \in \mathbb{R}^{n-r}$. We also note that $\widetilde{\mathbf{x}}^*$ is orthogonal to $\ker \mathbf{A}$ (which is spanned by $\mathbf{V}'$). That is: for every solution $\mathbf{x}$ of $\mathbf{A}\mathbf{x} = \mathbf{b}$ we have

$$\|\mathbf{x}\|^2 = \|\widetilde{\mathbf{x}}^*\|^2 + \|\mathbf{V}'\mathbf{y}\|^2,$$

which is obviously minimized by $\mathbf{y} = 0$. Hence, $\widetilde{\mathbf{x}}^*$ is the sought minimum norm solution.

5.3.3 Solution of the least squares problem with the SVD

The least squares problem could, alternatively to using the QR-factorization, also be solved with the SVD:

Exercise 5.20 *Assume that* $m \geq n$ *and that an SVD of* $\mathbf{A}$ *(with full rank) is given. Formulate a method to compute the solution of (5.2).* Remark: *Since computing an SVD is more expensive than computing a QR-factorization, this is rarely done in practice.*

5.3.4 Further properties of the SVD

Exercise 5.21 *Let* $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$ *be the SVD of a matrix* $\mathbf{A}$. *Show:*

(a) $\|\mathbf{A}\|_F^2 = \sum_i \sigma_i^2$, *where the* Frobenius norm *of* $\mathbf{A}$ *is given by* $\|\mathbf{A}\|_F^2 = \sum_{i,j} |\mathbf{A}_{ij}|^2$.

(b) $\|\mathbf{A}\|_2^2 = \max_i \sigma_i^2 = \sigma_1^2$.

We have:

Theorem 5.22 *Let $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$ be the SVD of the matrix $\mathbf{A}$ with rank r. Let the singular values be sorted in descending order. Then for every $\nu \in \{1, \dots, r\}$ the matrix $\mathbf{A}_\nu := \mathbf{U}(:, [1:\nu])\Sigma([1:\nu], [1:\nu])\mathbf{V}(:, [1:\nu])^\top$ satisfies*

$$\|\mathbf{A} - \mathbf{A}_\nu\|_2 = \min_{\mathbf{B}\in\mathbb{R}^{m\times n}:\ \mathrm{rank}(\mathbf{B})=\nu} \|\mathbf{A} - \mathbf{B}\|_2,$$
$$\|\mathbf{A} - \mathbf{A}_\nu\|_F = \min_{\mathbf{B}\in\mathbb{R}^{m\times n}:\ \mathrm{rank}(\mathbf{B})=\nu} \|\mathbf{A} - \mathbf{B}\|_F.$$

slide 18 - SVD

Remark 5.23 *The SVD can be used to determine the rank of a matrix by checking the number of non-zero singular values. In practice, one has to select a cut-off $\varepsilon > 0$ (typically a little larger than machine precision) and defines the rank $r = \#\{\sigma_i \,|\, \sigma_i \geq \varepsilon\}$.*

5.3.5 The Moore-Penrose Pseudoinverse (CSE)

We consider the least squares problem without conditions on m, n, and the rank of $\mathbf{A}$:

$$\text{find } \mathbf{x} \in \mathbb{R}^n \text{ s.t. } \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \leq \|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 \quad \forall \mathbf{y} \in \mathbb{R}^n. \tag{5.5}$$

This problem has solutions but possibly more than one. To enforce uniqueness, we seek again the "minimum norm" solution, i.e., the $\mathbf{x}^* \in \mathbb{R}^n$ with the smallest norm. We have:

Theorem 5.24 *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $\mathrm{rank}\,\mathbf{A} = r$. Let $\mathbf{A} = \widetilde{\mathbf{U}}\widetilde{\Sigma}\widetilde{\mathbf{V}}^\top$ be the reduced SVD of $\mathbf{A}$. Then $\mathbf{x}^* := \mathbf{A}^+\mathbf{b}$ with the* Moore-Penrose pseudoinverse

$$\mathbf{A}^+ := \widetilde{\mathbf{V}}\widetilde{\Sigma}^{-1}\widetilde{\mathbf{U}}^\top \tag{5.6}$$

is the minimum norm solution of the least squares problem (5.5).

Before proving Theorem 5.24, we formulate a representation of the orthogonal projection onto a subspace, which takes a particularly simple form if an orthonormal basis of the space is available:

Lemma 5.25 *Let $\mathbf{V} \in \mathbb{R}^{n\times k}$ have orthonormal columns. Then the map $\mathbf{x} \mapsto \mathbf{V}\mathbf{V}^\top\mathbf{x}$ is the orthogonal projection onto the subspace $\mathcal{V}$ spanned by the columns of $\mathbf{V}$. If $\widetilde{\mathbf{V}} \in \mathbb{R}^{n\times(n-k)}$ is such that $(\mathbf{V}, \widetilde{\mathbf{V}})$ is an orthogonal matrix (i.e., the space $\widetilde{\mathcal{V}}$ spanned by the columns of $\widetilde{\mathbf{V}}$ is the orthogonal complement of $\mathcal{V}$) then*

$$\mathbf{x} = \mathbf{V}\mathbf{V}^\top\mathbf{x} + \widetilde{\mathbf{V}}\widetilde{\mathbf{V}}^\top\mathbf{x} \qquad \forall \mathbf{x} \in \mathbb{R}^n. \tag{5.7}$$

Proof: We recall that the orthogonal projection $P\mathbf{x} \in \mathcal{V}$ of $\mathbf{x}$ onto $\mathcal{V}$ is characterized by

$$(\mathbf{x} - P\mathbf{x}, \mathbf{y})_2 = 0 \qquad \forall \mathbf{y} \in \mathcal{V}. \tag{5.8}$$

We now check that $P\mathbf{x} := \mathbf{V}\mathbf{V}^\top\mathbf{x}$ satisfies (5.8). We note that $\mathbf{V}\mathbf{V}^\top\mathbf{x} \in \mathcal{V}$ and that any $\mathbf{y}' \in \mathcal{V}$ can be written as $\mathbf{y}' = \mathbf{V}\mathbf{y}$ for some $\mathbf{y} \in \mathbb{R}^k$. We compute for arbitrary $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^k$:

$$(\mathbf{x} - \mathbf{V}\mathbf{V}^\top\mathbf{x}, \mathbf{V}\mathbf{y})_2 = (\mathbf{V}^\top(\mathbf{x} - \mathbf{V}\mathbf{V}^\top\mathbf{x}), \mathbf{y})_2 = ((\mathbf{V}^\top\mathbf{x} - \underbrace{\mathbf{V}^\top\mathbf{V}}_{=\mathrm{I}}\mathbf{V}^\top\mathbf{x}), \mathbf{y})_2 = 0,$$

which shows (5.8). Similarly, $\widetilde{\mathbf{V}}\widetilde{\mathbf{V}}^\top\mathbf{x}$ is the orthogonal projection of $\mathbf{x}$ onto the space $\widetilde{\mathcal{V}}$. By construction $\mathbf{x} - \mathbf{V}\mathbf{V}^\top\mathbf{x}$ is in the orthogonal complement of $\mathcal{V}$, i.e., in the space $\widetilde{\mathcal{V}}$. Hence, by the projection property $\widetilde{\mathbf{V}}\widetilde{\mathbf{V}}^\top(\mathbf{x} - \mathbf{V}\mathbf{V}'\mathbf{x}) = \mathbf{x} - \mathbf{V}\mathbf{V}'\mathbf{x}$. Since $\widetilde{\mathbf{V}}^\top\mathbf{V} = 0$, we obtain (5.7) by rearranging the terms. □

Proof of Theorem 5.24: We decompose $\mathbf{b}$ into its component in Range $\mathbf{A}$ and the rest using Lemma 5.25:

$$\mathbf{b} = \widetilde{\mathbf{U}}\widetilde{\mathbf{U}}^\top\mathbf{b} + \mathbf{U}'(\mathbf{U}')^\top\mathbf{b}, \qquad \mathbf{U}' := \mathbf{U}(:, [r+1:n]).$$

Next, we compute for arbitrary $\mathbf{x} \in \mathbb{R}^n$

$$\begin{aligned}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 &= \|\widetilde{\mathbf{U}}\widetilde{\Sigma}\widetilde{\mathbf{V}}^\top\mathbf{x} - \mathbf{b}\|_2^2 = \|\widetilde{\mathbf{U}}(\widetilde{\Sigma}\widetilde{\mathbf{V}}^\top\mathbf{x} - \widetilde{\mathbf{U}}^\top\mathbf{b}) + \mathbf{U}'(\mathbf{U}')^\top\mathbf{b}\|_2^2 \\ &= \|\widetilde{\mathbf{U}}(\widetilde{\Sigma}\widetilde{\mathbf{V}}^\top\mathbf{x} - \widetilde{\mathbf{U}}^\top\mathbf{b})\|_2^2 + \|\mathbf{U}'(\mathbf{U}')^\top\mathbf{b}\|_2^2 \\ &= \|\widetilde{\Sigma}\widetilde{\mathbf{V}}^\top\mathbf{x} - \widetilde{\mathbf{U}}^\top\mathbf{b}\|_2^2 + \|\mathbf{U}'(\mathbf{U}')^\top\mathbf{b}\|_2^2\end{aligned}$$

This expression is minimal if we can find $\mathbf{x}$ such that

$$\widetilde{\mathbf{V}}^\top\mathbf{x} = \widetilde{\Sigma}^{-1}\widetilde{\mathbf{U}}^\top\mathbf{b}. \tag{5.9}$$

(We will see at the end of the proof that indeed such $\mathbf{x}$ exist.) Let us now seek the $\mathbf{x}^*$ from all $\mathbf{x}$ satisfying (5.9) with minimal norm. We write again with Lemma 5.25

$$\mathbf{x} = \widetilde{\mathbf{V}}\widetilde{\mathbf{V}}^\top\mathbf{x} + \mathbf{V}'(\mathbf{V}')^\top\mathbf{x}.$$

Hence, any $\mathbf{x}$ that satisfies (5.9) has to satisfy

$$\|\mathbf{x}\|_2^2 = \|\widetilde{\mathbf{V}}\widetilde{\mathbf{V}}^\top\mathbf{x}\|_2^2 + \|\mathbf{V}'(\mathbf{V}')^\top\mathbf{x}\|_2^2 \overset{(5.9)}{=} \|\widetilde{\mathbf{V}}\widetilde{\Sigma}\widetilde{\mathbf{U}}^\top\mathbf{b}\|_2^2 + \|\mathbf{V}'(\mathbf{V}')^\top\mathbf{x}\|_2^2.$$

We see that $\mathbf{x}^*$ with the smallest norm should be such that $(\mathbf{V}')^\top\mathbf{x}^* = 0$. Then, we get

$$\mathbf{x}^* \overset{(\mathbf{V}')^\top\mathbf{x}^*=0}{=} \widetilde{\mathbf{V}}\widetilde{\mathbf{V}}^\top\mathbf{x}^* \overset{(5.9)}{=} \widetilde{\mathbf{V}}\widetilde{\Sigma}^{-1}\widetilde{\mathbf{U}}^\top\mathbf{b}.$$

Indeed, this $\mathbf{x}^*$ satisfies $(\mathbf{V}')^\top\mathbf{x}^* = 0$ as well as (5.9). Hence, we have found the unique minimum norm solution. □

Let us interpret the Moore-Penrose pseudoinverse. To that end, we let us restrict $\mathbf{A}$ to $(\operatorname{Ker}\mathbf{A})^\perp$, which we denote by $\mathbf{A}_K$ to emphasize that the domain of definition and range has changed:

$$\begin{aligned}\mathbf{A}_K : (\operatorname{Ker}\mathbf{A})^\perp &\to \operatorname{Range}\mathbf{A} \\ \widetilde{\mathbf{V}}z &\mapsto \mathbf{A}\widetilde{\mathbf{V}}z = \widetilde{\mathbf{U}}\widetilde{\Sigma}\widetilde{\mathbf{V}}^\top\widetilde{\mathbf{V}}z = \widetilde{\mathbf{U}}\widetilde{\Sigma}z\end{aligned}$$

This map is a bijection. Indeed, since the columns of $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$ are linearly independent, the inverse $\mathbf{A}_K^{-1}$ is easily read off to be:[2]

$$\mathbf{A}_K^{-1} : \widetilde{\mathbf{U}}\zeta \mapsto \widetilde{\mathbf{V}}\widetilde{\Sigma}^{-1}\zeta$$

[2] An alternative way to see that $\mathbf{A}_K$ is a bijection is to check the dimensions: $\dim(\operatorname{Ker}\mathbf{A})^\perp = n - \dim\operatorname{Ker}\mathbf{A}$ and by a linear algebra fact $n = \dim(\operatorname{Ker}\mathbf{A})^\perp + \dim\operatorname{Range}\mathbf{A}$ so that $\dim(\operatorname{Ker}\mathbf{A})^\perp = \dim\operatorname{Range}\mathbf{A}$

We now consider

$$\begin{array}{ccccc} \mathbb{R}^m & \overset{\text{ortho. Proj.}}{\longrightarrow} & \operatorname{Range}\mathbf{A} & \overset{\mathbf{A}_K^{-1}}{\longrightarrow} & (\operatorname{Ker}\mathbf{A})^\perp \\ \mathbf{b} & \mapsto & \widetilde{\mathbf{U}}(\widetilde{\mathbf{U}}^\top\mathbf{b}) & \mapsto & \widetilde{\mathbf{V}}\widetilde{\Sigma}^{-1}\widetilde{\mathbf{U}}^\top\mathbf{b} \end{array}$$

This is precisely $\mathbf{A}^+$! Hence, the Moore-Penrose pseudoinverse takes from a vector $\mathbf{b}$ its component in $\operatorname{Range}\mathbf{A}$ and then applies the well-defined inverse $\mathbf{A}_K^{-1}$ that maps from $\operatorname{Range}\mathbf{A}$ to $(\operatorname{Ker}\mathbf{A})^\perp$.

Exercise 5.26 *Let* $\operatorname{rank}\mathbf{A} = r$. *Show:* $\|\mathbf{A}^+\|_2 = \sigma_r^{-1}$.

5.3.6 Further remarks

- The Moore-Penrose pseudoinverse is the inverse of $\mathbf{A}$ if $\mathbf{A} \in \mathbb{R}^{n\times n}$ is invertible.
- In general, $\mathbf{A}^+$ shares some properties with the inverse: $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$ and $(\mathbf{A}^+)^+ = \mathbf{A}$.

Computing the SVD

The SVD is computed with variants of algorithms that compute eigenvalues and eigenvectors. Since $\mathbf{A}^\top\mathbf{A} = \mathbf{V}^\top\Sigma^\top\Sigma\mathbf{V}$ and $\mathbf{A}\mathbf{A}^\top = \mathbf{U}^\top\Sigma\Sigma^\top\mathbf{U}^\top$, one could compute the SVD by computing the eigenvalues and eigenvectors of $\mathbf{A}^\top\mathbf{A}$ or $\mathbf{A}\mathbf{A}^\top$. However, since $\mathbf{A}^\top\mathbf{A}$ and $\mathbf{A}\mathbf{A}^\top$ are typically ill conditioned, one resorts to computing the eigenvalues and eigenvectors of the symmetric matrix

$$\begin{pmatrix} 0 & \mathbf{A}^\top \\ \mathbf{A} & 0 \end{pmatrix},$$

whose eigenvalues are $\pm\sigma_i$. A popular algorithm for the SVD is $\rightarrow$ Golub-Kahan.

6 Nonlinear Equations and Newton's Method

goal: determine zero $\mathbf{x}^*$ of $\mathbf{f}(\mathbf{x}^*) = 0$
Since there are typically no exact solution formulas, the zero $\mathbf{x}^*$ is approximated by iterates $\mathbf{x}_n$ with $\lim_{n\to\infty} \mathbf{x}_n = \mathbf{x}^*$. The most common form is that of a *fixed point iteration*

$$\mathbf{x}_{n+1} = \Phi(\mathbf{x}_n) \tag{6.1}$$

with an initial guess $\mathbf{x}_0$ that is taken sufficiently close to $\mathbf{x}^*$. Thus, the iterative method is described by the function Φ.

Exercise 6.1 *Show: If $\mathbf{x}_n \to \mathbf{x}^*$ then $\mathbf{x}^*$ is a* fixed point *of Φ, i.e., $\mathbf{x}^* = \Phi(\mathbf{x}^*)$ (assumption: Φ is continuous at $\mathbf{x}^*$).*

6.1 Newton's method in 1D

goal: Find zero x^* of $f(x^*) = 0$
Idea: *linearize* f at the current iterate x_n and find zero of the linearization.
procedure:

1. x_n = current iterate
2. $L(x) := f(x_n) + f'(x_n)(x - x_n)$ ⟦ linearization is the tangent at x_n, i.e., the Taylor expansion up to the linear term ⟧
3. x_{n+1} := zero of L, i.e.,
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{6.2}$$

We recognize that the 1D-Newton method (6.2) has the form $x_{n+1} = \Phi^{Newton}(x_n)$ of a fixed point iteration with Φ^{Newton} given by

$$\Phi^{Newton}(x) = x - \frac{f(x)}{f'(x)}. \tag{6.3}$$

Example 6.2 *$x^* = \sqrt{a}$ is the zero of $f(x) = x^2 - a$. With $f'(x) = 2x$, Newton's method is*

$$x_{n+1} = \Phi^{Newton}(x_n) = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - a}{2x_n}.$$

The rapid convergence of the method is visible in Fig. 6.1 for the choice $a = 2$ and initial value $x_0 = 2$. In fact, we observe so-called quadratic convergence *in that the error behaves like $|x^* - x_{n+1}| \approx C|x^* - x_n|^2$ for some $C > 0$.*

	Newton iterates ($x_0 = 2$)	error
x_1	1.5	8.578643762690485_{-2}
x_2	**1.41**6666666666667	2.453104293571595_{-3}
x_3	**1.41421**5686274510	2.1239014147411694_{-6}
x_4	**1.41421356237**4690	1.5947243525715749_{-12}
exact:	1.414213562373095	

Figure 6.1: Newton's method for computing $\sqrt{2}$ (cf. Example 6.2)

6.2 Convergence of fixed point iterations

The key property that ensures convergence of the fixed point iteration (6.1) is that Φ is a *contraction*:

Definition 6.3 *The function $\Phi : \mathbb{R}^d \to \mathbb{R}^d$ is a contraction (with respect to the norm $\|\cdot\|$) near the point $\mathbf{x}^*$ if there are $q \in (0,1)$ and $\varepsilon > 0$ such that*

$$\|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| \le q\|\mathbf{x} - \mathbf{y}\| \qquad \forall \mathbf{x}, \mathbf{y} \in B_\varepsilon(\mathbf{x}^*). \tag{6.4}$$

Exercise 6.4 *Consider the case $d = 1$. Show: If $\Phi \in C^1$ and $|\Phi'(x^*)| < 1$ near a point x^*, then Φ is a contraction near x^*.*

The following result shows that the contraction property implies convergence of the fixed point iteration (6.1) if the initial value $\mathbf{x}_0$ is sufficiently close to the fixed point $\mathbf{x}^*$.

Theorem 6.5 *Let Φ be a contraction with contraction constant $q \in (0,1)$ near the fixed point $\mathbf{x}^* = \Phi(\mathbf{x}^*)$. Then there is $\varepsilon > 0$ such that for $\mathbf{x}_0 \in B_\varepsilon(\mathbf{x}^*)$ the iterates $\mathbf{x}_n$ given by (6.1) converge to $\mathbf{x}^*$. Moreover,*

$$\|\mathbf{x}^* - \mathbf{x}_{n+1}\| \le q\|\mathbf{x}^* - \mathbf{x}_n\| \qquad \forall n \in \mathbb{N}_0. \tag{6.5}$$

Proof: Let $\varepsilon > 0$ be given by Def. 6.3 and $x_n \in B_\varepsilon(x^*)$. Then:

$$\|x^* - x_{n+1}\| = \|x^* - \Phi(x_n)\| \overset{x^* \text{ fixed pt}}{=} \|\Phi(x^*) - \Phi(x_n)\| \overset{\text{contraction property}}{\le} q\|x^* - x_n\|.$$

Hence, if $x_0 \in B_\varepsilon(x^*)$, then by induction all iterates $x_n \in B_\varepsilon(x^*)$ and $\|x^* - x_n\| \to 0$. □

Exercise 6.4 gives an easy condition (in the scalar case $d = 1$) when the iteration (6.1) converges:

Exercise 6.6 *Let $d = 1$ and $\Phi \in C^1$ satisfy $|\Phi'(x^*)| < 1$ at the fixed point x^* of Φ. Then the iterates x_n given by (6.1) converge to x^* provided the initial value x_0 is sufficiently close to x^*.*
Remark: *The vector-valued analog is as follows: The derivative Φ' is a $d \times d$ matrix and if there is a norm $\|\cdot\|$ such that $\|\Phi'(\mathbf{x}^*)\| < 1$ at a fixed point $\mathbf{x}^*$ of Φ, then Φ is a contraction near $\mathbf{x}^*$.*

Example 6.7 **slide 19 - Convergence of fixed point iterations**
We seek a solution of the nonlinear equation

$$2 - x^2 - e^x = 0. \tag{6.6}$$

n	$x_{n+1} = \Phi_1(x_n)$	$x_{n+1} = \Phi_2(x_n)$
0	0.592687716508341	0.559615787935423
1	0.437214425050104	0.522851128605001
2	0.672020792350124	0.546169619063046
3	0.204473907097276	0.531627015197373
4	0.879272743474883	0.540795632739194
5	stop: $(2 - e^{0.87} < 0)$	0.535053787215218
6		0.538664955236433
7		0.536399837485597
8		0.537823020842571
9		0.536929765486145

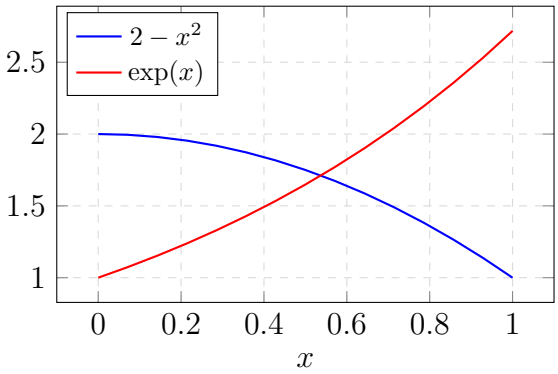


Figure 6.2: Left: fixed point iteration of Example 6.7. Right: $x \mapsto e^x$ and $x \mapsto 2 - x^2$.

Graphical considerations show that there is exactly one positive solution $x^* \approx 0.5$. For $x > 0$ equation (6.6) can be converted to a fixed point form in several ways:

$$x = \sqrt{2 - e^x} =: \Phi_1(x), \qquad x = \ln(2 - x^2) =: \Phi_2(x), \tag{6.7}$$

The fixed point iterations based on Φ_1 and Φ_2 behave differently when initialized with $x_0 = 0.5$ as is visible in Table 6.2: Whereas the iteration $x_{n+1} = \Phi_2(x_n)$ converges to the correct value $x^* = 0.5372744491738\ldots$ the iteration $x_{n+1} = \Phi_1(x_n)$ does not converge. The reason is that $|\Phi_1'(x^*)| \approx |-1.59| > 1$ whereas $|\Phi_2'(x^*)| \approx 0.31 < 1$.

Theorem 6.5 shows that if Φ is a contraction, then one has *linear* convergence, i.e., the error decreases by a factor $q \in (0, 1)$ in each step. A special situation arises if $\Phi'(x^*) = 0$. Then faster convergence is possible:

Theorem 6.8 *Let $d = 1$ and $\Phi \in C^p(\mathbb{R}^d)$, $p \geq 2$. Assume $x^* = \Phi(x^*)$ and $0 = \Phi^{(j)}(x^*)$ for $j = 1, \ldots, p-1$. Then there are C, $\varepsilon > 0$ such that for $x_0 \in B_\varepsilon(x^*)$ the iterates x_n given by (6.1) converge to x^* and*

$$|x^* - x_{n+1}| \leq C|x^* - x_n|^p \qquad \forall n \in \mathbb{N}_0.$$

Proof: By Theorem 6.5 we already know that the iterates converge to x^* if ε is sufficiently small. For the estimate, we modify the proof of Theorem 6.5. By Taylor expansion around x^* we have

$$\begin{aligned} |x^* - x_{n+1}| = |\Phi(x^*) - \Phi(x_n)| &= \left| \frac{1}{(p-1)!} \int_{x^*}^{x_n} (x_n - t)^{p-1} \Phi^{(p)}(t)\, dt \right| \\ &\leq \frac{\|\Phi^{(p)}\|_{\infty, B_\varepsilon(x^*)}}{(p-1)!} |x^* - x_n|^p. \end{aligned}$$

□

In the setting of Theorem 6.8, we say that the iteration converges with *order* p. In particular, for $p = 2$ the method converges *quadratically*. Example 6.2 shows that the Newton method applied to the problem $f(x) = x^2 - a = 0$ convergence quadratically. This is typical of the Newton method:

Corollary 6.9 *Let $d = 1$ and $f \in C^2$. Assume $f(x^*) = 0$ and $f'(x^*) \neq 0$. Then Newton's method converges quadratically. That is, there are constants C, $\varepsilon > 0$ such that if $|x^* - x_0| \leq \varepsilon$ then the sequence $(x_n)_n$ converges to x^* and*

$$|x^* - x_{n+1}| \leq C|x^* - x_n|^2 \qquad \forall n.$$

Proof: One computes (exercise!) $\frac{d\Phi^{Newton}}{dx}(x^*) = 0$. Hence, Theorem 6.8 implies (at least) quadratic convergence. □

The quadratric convergence asserted in Cor. 6.9 requires $f'(x^*) \neq 0$. This is not an artefact of the proof:

Exercise 6.10 *Apply Newton's method to find the zero of $f(x) = x^2$. Show that Newton's method converges only linearly.*

6.3 Newton's method in higher dimensions

Idee: as in 1D: linearize (= Taylor expansion up to linear terms) and find zero of linearization procedure:

- in $\mathbb{R}^n$: $\mathbf{x}_n$ = current iterate
- linearization $L(\mathbf{x}) := \mathbf{f}(\mathbf{x}_n) + \mathbf{f}'(\mathbf{x}_n)(\mathbf{x} - \mathbf{x}_n)$ = linearization of $\mathbf{f}$ at $\mathbf{x}_n$, where

$$\mathbf{f}'(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{pmatrix}$$

- determine $\mathbf{x}_{n+1}$ as the zero of L, i.e.,

$$\mathbf{x}_{n+1} := \mathbf{x}_n - \Big(\mathbf{f}'(\mathbf{x}_n)\Big)^{-1}\mathbf{f}(\mathbf{x}_n).$$

That is, the iteration function Φ is

$$\Phi^{\text{Newton}}(\mathbf{x}) = \mathbf{x} - \Big(\mathbf{f}'(\mathbf{x})\Big)^{-1}\mathbf{f}(\mathbf{x}) \tag{6.8}$$

The convergence of the method is analogous to the 1D situation:

Theorem 6.11 *Let $\mathbf{f} \in C^2(B_\delta(\mathbf{x}^*))$ for some $\delta > 0$. Assume $\mathbf{f}(x^*) = 0$ and $\mathbf{f}'(x^*)$ is an invertible matrix. Then there exist $\varepsilon > 0$ and $C > 0$ such that if $\mathbf{x}_0 \in B_\varepsilon(\mathbf{x}^*)$, then all iterates $\mathbf{x}_n$ are in $B_\varepsilon(\mathbf{x}^*)$, one has convergence $\mathbf{x}_n \to \mathbf{x}^*$, and*

$$\|\mathbf{x}^* - \mathbf{x}_{n+1}\| \leq C\|\mathbf{x}^* - \mathbf{x}_n\|^2 \qquad \forall n.$$

Theorem 6.11 states *quadratic* convergence of Newton's method (provided the starting value is sufficiently close to $\mathbf{x}^*$) provided $\mathbf{f}'(\mathbf{x}^*)$ is invertible.

Remark 6.12 *In practice the Newton step is* not *realized by computing the inverse* $(\mathbf{f}')^{-1}$ *but by solving a linear system:*

1. *compute* $\mathbf{f}'(\mathbf{x}_n)$ *and the residual* $\mathbf{f}(\mathbf{x}_n)$
2. *compute the correction by solving the linear system* $\mathbf{f}'(\mathbf{x}_n)\delta = \mathbf{f}(\mathbf{x}_n)$
3. *perform the update* $\mathbf{x}_{n+1} := \mathbf{x}_n - \delta$

Remark 6.13 *The residual* $\mathbf{f}(\mathbf{x}_n)$ *is some measure for the error* $\mathbf{x}^* - \mathbf{x}_n$*. If* $\mathbf{f}'(\mathbf{x}^*)$ *is invertible, then for* $\mathbf{x}_n$ *sufficiently close to* $\mathbf{x}^*$*, Taylor expansion indicates*

$$\mathbf{f}(\mathbf{x}_n) = \mathbf{f}(\mathbf{x}_n) - \mathbf{f}(\mathbf{x}^*) \approx \mathbf{f}'(\mathbf{x}^*)(\mathbf{x}_n - \mathbf{x}^*)$$

so that we can expect

$$\|(\mathbf{f}'(\mathbf{x}^*))^{-1}\mathbf{f}(\mathbf{x}_n)\| \approx \|\mathbf{x}^* - \mathbf{x}_n\|. \tag{6.9}$$

The residual $\mathbf{f}(\mathbf{x}_n)$ *still is a measure for the error, however, only up to a constant depending on* $\mathbf{f}'(\mathbf{x}^*)$*:*

$$\|\mathbf{f}(\mathbf{x}_n)\| \le \|\mathbf{f}'(\mathbf{x}^*)\|\|\mathbf{x}^* - \mathbf{x}_n\| + O(\|\mathbf{x}^* - \mathbf{x}_n\|^2), \tag{6.10}$$

$$\|\mathbf{x}^* - \mathbf{x}_n\| \le \|(\mathbf{f}'(\mathbf{x}^*))^{-1}\|\|\mathbf{f}(\mathbf{x}_n)\| + O(\|\mathbf{x}^* - \mathbf{x}_n\|^2). \tag{6.11}$$

6.4 Implementation aspects of Newton methods

stopping criteria

1. $\mathbf{x}_n$ close to $\mathbf{x}^*$ $\Rightarrow$ quadratic convergence $\Rightarrow$ $\|\mathbf{x}_{n+1} - \mathbf{x}_n\|$ is a good estimate for $\|\mathbf{x}_n - \mathbf{x}^*\|$:

$$\|\mathbf{x}_n - \mathbf{x}^*\| \;\le\; \|\mathbf{x}_n - \mathbf{x}_{n+1}\| + \underbrace{\|\mathbf{x}_{n+1} - \mathbf{x}^*\|}_{\substack{\le\ c\,\|x_n - x^*\|^2 \\ \ll\ \|x_n - x^*\|}}$$

   $\Rightarrow$ If each Newton step is cheap, then the stopping criterion is

$$\|\mathbf{x}_{n+1} - \mathbf{x}_n\| \;\le\; \text{given tolerance}$$

2. If Newton steps are expensive (e.g., for large systems of equations) then one can approximate $\|\mathbf{x}_{n+1} - \mathbf{x}_n\|$ as follows:

$$\|\mathbf{x}_{n+1} - \mathbf{x}_n\| = \|\Big(\mathbf{f}'(\mathbf{x}_n)\Big)^{-1}\mathbf{f}(\mathbf{x}_n)\| \approx \|\Big(\mathbf{f}'(\mathbf{x}_{n-1})\Big)^{-1}\mathbf{f}(\mathbf{x}_n)\|$$

   This expression is computable since $\mathbf{f}'(\mathbf{x}_{n-1})$ has been determined for the computation of $\mathbf{x}_n$. If an LU-factorization of $\mathbf{f}'(\mathbf{x}_{n-1})$ is available, then the computation of $\mathbf{f}'^{-1}(\mathbf{x}_{n-1})\mathbf{f}(x_n)$ is comparatively cheap.

computing $\mathbf{f}'(\mathbf{x}_n)$:

1. problem: often $\mathbf{f}'$ is not explicitly available but only $\mathbf{f}$ (e.g., if $\mathbf{f}$ is available as a C-code). Then $\mathbf{f}'(\mathbf{x}_n)$ can be approximated by difference quotients.

2. problem: Computing $\mathbf{f}'(\mathbf{x}_n)$ can be expensive (for example: for large d the $d \times d$-matrix $\mathbf{f}'$ has many entries) Then one often uses the *simplified Newton method*

$$\mathbf{x}_{n+1} \;=\; \mathbf{x}_n - \Big(\mathbf{f}'(\mathbf{x}_0)\Big)^{-1}\mathbf{f}(\mathbf{x}_n)$$

   Since one uses the same, fixed derivative (at the point $\mathbf{x}_0$), the method is only linearly convergent.

Exercise 6.14 *Let $\mathbf{B} \in \mathbb{R}^{d\times d}$ be invertible, $\widetilde{\mathbf{f}}(x) := \mathbf{B}\mathbf{f}(x)$. Then: $\mathbf{f}(x^*) = 0$ if and only if $\widetilde{\mathbf{f}}(\mathbf{x}^*) = 0$, and the Newton iterates for computing the zeros of $\mathbf{f}$ and of $\widetilde{\mathbf{f}}$ coincide.*

6.5 Damped and globalized Newton methods

Problem: Newton's method converges only *locally*, i.e., if $\mathbf{x}_0$ is sufficiently close to the zero $\mathbf{x}^*$.
goal: methods that cope (reasonably well) with poor initial values $\mathbf{x}_0$.

6.5.1 Damped Newton method

Problem: quite often, the Newton steps $\mathbf{x}_{n+1} - \mathbf{x}_n$ are too large for convergence.

slide 20 - Damped Newton method

The way to cope with this problem is the *damped Newton method* where, for chosen $\lambda_n \in (0, 1]$, the update is

$$\mathbf{x}_{n+1} := \mathbf{x}_n - \lambda_n(\mathbf{f}'(\mathbf{x}_n))^{-1}\mathbf{f}(\mathbf{x}_n) \tag{6.12}$$

For suitably small λ_n, this method converges for a larger regime of initial values $\mathbf{x}_0$. However, the convergence is only linear. One is therefore interested in methods where the parameters λ_n are selected adaptively and in particular $\lambda_n = 1$ for the iterates sufficiently close to $\mathbf{x}^*$ so as to obtain the quadratic convergence of the Newton method. An algorithm that realizes this is given in Alg. 6.16.

6.5.2 A digression: descent methods

Let $g : \mathbb{R}^d \to \mathbb{R}$ be a given function. Minima of g can be sought with *descent methods*, which are iterative methods that determine the next iterate $\mathbf{x}_{n+1}$ from a current iterate $\mathbf{x}_n$ as follows:

1. select a *search direction* $\mathbf{d}_n$

2. select a *step length* λ_n such that for $\mathbf{x}_{n+1} := \mathbf{x}_n + \lambda_n\mathbf{d}_n$ one has $g(\mathbf{x}_{n+1}) < g(\mathbf{x}_n)$.

The search direction $\mathbf{d}_n$ is called a *descent direction* if the 1D function $\widetilde{g}(t) := g(\mathbf{x}_n + t\mathbf{d}_n)$ satisfies $\widetilde{g}'(0) < 0$, i.e., is decreasing for small $t > 0$. Put differently, $\mathbf{d}_n$ needs to satisfy

$$\nabla g(\mathbf{x}_n) \cdot \mathbf{d}_n < 0.$$

The method of *steepest descent* corresponds to the choice $\mathbf{d}_n = -\nabla g(\mathbf{x}_n)$.
The second ingredient of a descent method is the choice of the step length λ_n. The "greedy" approach would be to select λ_n such that

$$\min_{t>0} \widetilde{g}(t) = \widetilde{g}(\lambda_n).$$

Since this "line search" is still quite expensive, several other options are common that realize the idea of selecting a step size with "sufficient" descent. We mention the so-called *Armijo*-rule: Given $\sigma \in (0,1)$ and $q \in (0,1)$ one selects the *largest* step length of the form q^k, $k = 0, 1, \ldots$, such that

$$\widetilde{g}(q^k) < \widetilde{g}(0) + \sigma \widetilde{g}'(0) q^k,$$

or, written in terms of g

$$g(\mathbf{x}_n + q^k \mathbf{d}_n) < g(\mathbf{x}_n) + \sigma(\nabla g(\mathbf{x}_n) \cdot \mathbf{d}_n) q^k. \tag{6.13}$$

This can be realized by trying the cases $k = 0$, 1, etc. in turn until (6.13) is satisfied. This step length choice can be interpreted as trying to make fairly large steps with a reasonable reduction of the functional g.

6.5.3 Globalized Newton method as a descent method

observe: zeros of $\mathbf{f}$ are minima of $\mathbf{x} \mapsto \|\mathbf{f}(\mathbf{x})\|_2^2 = \mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$.
idea: View the damped Newton method as a descent method with search direction $\mathbf{d}_n := -(\mathbf{f}'(\mathbf{x}_n))^{-1}\mathbf{f}(\mathbf{x}_n)$ and step length parameter λ_n.
For this idea to work, we need to know that the so-called *Newton direction*

$$\mathbf{d}_n := -(\mathbf{f}'(\mathbf{x}_n))^{-1}\mathbf{f}(\mathbf{x}_n) \tag{6.14}$$

is a descent direction for $g(\mathbf{x}) := \|\mathbf{f}(\mathbf{x})\|_2^2$.

Lemma 6.15 *Let $\mathbf{f} \in C^2(\mathbb{R}^d)$. Then: For given $\mathbf{x}$ and $\mathbf{d} := -(\mathbf{f}'(\mathbf{x}))^{-1}\mathbf{f}(\mathbf{x})$ the function $\widetilde{g}(\lambda) := g(\mathbf{x} + \lambda \mathbf{d})$ has the Taylor expansion $\widetilde{g}(\lambda) = g(\mathbf{x}) - 2\lambda g(\mathbf{x}) + O(\lambda^2)$ for small λ.*

Proof: For notational simplicity we consider the case $d = 1$. Then $g(x) = f^2(x)$ and $\widetilde{g}(\lambda) = f^2(x + \lambda \mathfrak{d}(x))$ with $\mathfrak{d}(x) = -(f'(x))^{-1} f(x)$. Then by Taylor, we have for small λ

$$\begin{aligned}\widetilde{g}(\lambda) &= \widetilde{g}(0) + \lambda g'(0) + O(\lambda^2) = f^2(x) + \lambda 2 f(x) f'(x) \mathfrak{d}(x) + O(\lambda^2) \\ &= f^2(x) - \lambda 2 f(x) f'(x) \frac{f(x)}{f'(x)} + O(\lambda^2) = f^2(x) - 2\lambda f^2(x) + O(\lambda^2).\end{aligned}$$

□

Lemma 6.15 shows that the Newton direction is a descent direction and that, for λ sufficiently small, we may achieve a descent

$$g(\mathbf{x}_n + \lambda_n \mathbf{d}_n) - g(\mathbf{x}_n) \approx 2\lambda_n g(\mathbf{x}_n) \tag{6.15}$$

$\Rightarrow$ sensible goals for selecting λ are:

- if $\mathbf{x}_n$ is close to $\mathbf{x}^*$ then select $\lambda = 1$ (so that actual Newton steps with quadratic convergence are performed!). We note that the quadratic convergence implies a descent of almost $\|\mathbf{f}(\mathbf{x}_n)\|^2$: for $\mathbf{x}_n$ near $\mathbf{x}^*$ we have

$$\|\mathbf{f}(\mathbf{x}_{n+1})\|^2 \overset{(6.10)}{\leq} C_1\|\mathbf{x}^* - \mathbf{x}_{n+1}\|_2^2 \overset{\text{quad. conv.}}{\leq} C_2\|\mathbf{x}^* - \mathbf{x}_n\|_2^4 \overset{(6.11)}{\leq} C_3\|\mathbf{f}(\mathbf{x}_n)\|_2^4.$$

  In other words: for actual Newton steps, we expect $\|\mathbf{f}(\mathbf{x}_n)\|_2^2 - \|\mathbf{f}(\mathbf{x}_{n+1})\|_2^2 \approx \|\mathbf{f}(\mathbf{x}_n)\|_2^2$.

- If $\mathbf{x}_n$ is far from $\mathbf{x}^*$, then select λ small but s.t. the descent $\|\mathbf{f}(\mathbf{x}_n)\|_2^2 - \|\mathbf{f}\big(\mathbf{x}_n + \lambda\,\mathbf{d}(x_n)\big)\|_2^2$ is large. By (6.15), a descent $\|\mathbf{f}(\mathbf{x}_n + \lambda\mathbf{d}_n)\|_2^2 - \|\mathbf{f}(\mathbf{x}_n)\|_2^2 \approx 2\lambda_n\|\mathbf{f}(\mathbf{x}_n)\|_2^2$ is possible for small λ_n

We wish to require the descent to be compatible with Newton steps. Therefore, we require a descent of $\approx \lambda_n\|\mathbf{f}(\mathbf{x}_n)\|_2^2$ rather than the "greedy" $2\lambda_n\|\mathbf{f}(\mathbf{x}_n)\|_2^2$. This is what we enforce in the following algorithm:

Algorithm 6.16 *Input: initial value* $\mathbf{x}_0$, *parameter* $\mu, q \in (0,1)$

$\lambda_0 := 1$
$n := 0$
while *(stopping criterion not satisfied)* ***do***
  $\mathbf{d}_n := -\Big(\mathbf{f}'(\mathbf{x}_n)\Big)^{-1}\mathbf{f}(\mathbf{x}_n)$
  while $\Big(\|\mathbf{f}(\mathbf{x}_n)\|_2^2 - \|\mathbf{f}(\mathbf{x}_n + \lambda_n\,\mathbf{d}_n)\|_2^2 < \mu\,\lambda_n\|\mathbf{f}(\mathbf{x}_n)\|_2^2\Big)$ ***do*** *% reduce λ until sufficient amount of descent*
    $\lambda_n := \lambda_n \cdot q$
  end while
  $\mathbf{x}_{n+1} := \mathbf{x}_n + \lambda_n\,\mathbf{d}_n$
  $\lambda_{n+1} := \min\Big(1, \frac{\lambda_n}{q}\Big)$ *% try a little large λ next time*
end while

Remark 6.17 *The $\|\cdot\|_2$-norm was selected for convenience of exposition. Especially for large systems, other norms may be more appropriate.*

6.6 Gauss-Newton

A practically relevant case is that of "nonlinear least squares problems": given a function $F : \mathbb{R}^n \to \mathbb{R}^m$ the goal is

$$\text{Find } x^\star \text{ s.t.} \qquad \|F(x^\star)\|_2 \leq \|F(x)\|_2 \qquad \forall x \in \mathbb{R}^n. \tag{6.16}$$

Such problems arise, for example when fitting parameters x to measurements of a nonlinear model.
(Local) minima $x^\star$ of the function $g(x) := \|F(x)\|_2^2$ satisfy $\nabla g(x^\star) = 0$, i.e.,

$$G(x) := (F'(x))^\top F(x) \overset{!}{=} 0.$$

The Newton iteration is then[1]

$$G'(x_n)\Delta x_n = -G(x_n), \qquad G'(x) = (F'(x))^\top F'(x) + (F''(x))^\top F(x). \tag{6.17}$$

Let us next assume that $F'(x)$ has full rank near a solution $x^\star$ so that $(F'(x))^\top F'(x)$ is invertible. Let us also assume that

$$F(x^\star) = 0.$$

Then, $F''(x)F(x)$ is small near the solution $x^\star$ so that one could replace in (6.17) the full derivative $G'(x) = (F'(x))^\top F'(x) + (F''(x))^\top F(x)$ with a simplified version $G'(x) \approx (F'(x))^\top F'(x)$. The resulting method is

$$(F'(x_n))^\top F'(x_n)\Delta x_n = -(F'(x_n))^\top F(x_n). \tag{6.18}$$

These are the normal equations for the following *linear* least squares problem:

$$\text{Find } \Delta x_n \text{ s.t.} \|F'(x_n)\Delta x_n + F(x_n)\|_2^2 \le \|F'(x_n)y + F(x_n)\|_2^2 \qquad \forall y \in \mathbb{R}^n. \tag{6.19}$$

Thus, the nonlinear least squares problem (6.16) has been reduced to a sequence of linear least squares problems. The simplification is quite significant in that the second derivative G'' does not have to be computed! Normally in Newton methods, an approximation of the derivative (here: G') leads to a convergence reduction from quadratic to linear. In the present case, the neglected term $F''(x)F(x)$ is small and even vanishes asymptotically as $x \to x^\star$. Hence, there is hope that the Gauss-Newton method still converges quadratically:

Theorem 6.18 *Assume that F is sufficiently smooth, that $F(x^*) = 0$ and that $F'(x^*)$ has full rank. Then, the Gauss-Newton method (6.19) converges locally quadratically, i.e., for x_0 sufficiently close to x^*, the sequence of iterates x_n satisfies*

$$\|x^* - x_{n+1}\|_2 \le C\|x^* - x_n\|_2^2 \quad \forall n.$$

If $F(x^*) \neq 0$ but still $F'(x^*)$ has full rank, then the Gauss-Newton method converges but only linearly for starting values sufficiently close to the solution x^*.

Exercise 6.19 *Consider the case $n = m = 1$. Formulate the Gauss-Newton method for solving $f(x^\star) = 0$. Under the assumption $f(x^\star) = 0$ and $f'(x^*) \neq 0$, show that the Gauss-Newton method reduces to the standard Newton iteration for the problem of finding $x^\star$ with $f(x^*) = 0$.*

[1]The second derivative G'' is a third order tensor but we will not formally define this object as we will not need it in the sequel. At this point, it suffices to accept that the notation is set up in such a way that what one expects from simple calculus in 1D extends to multi-d

6.7 Quasi-Newton methods (CSE)

Problem: often, the computation of $\mathbf{f}'$ is expensive.
simple solution: simplified Newton method where $\mathbf{f}'(\mathbf{x}_n)$ is replaced with $\mathbf{f}'(\mathbf{x}_0)$. Downside: *linear convergence*
goal: methods that converge superlinearly but are cheaper than full Newton method

6.7.1 Broyden method

Setting: $\mathbf{f} \in C^1(\mathbb{R}^d;\mathbb{R}^d)$, $\mathbf{f}(x^*) = 0$, $\mathbf{f}'(x^*)$ invertible
Broyden methods are iterative methods of the form $\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{H}_n^{-1}\mathbf{f}(\mathbf{x}_n)$ with suitable matrices $\mathbf{H}_n$.
idea of Broyden's method

- after computing $\mathbf{x}_{n+1}$ compute the next $\mathbf{H}_{n+1}$ from $\mathbf{H}_n$
- $\mathbf{H}_{n+1}$ is some kind of "approximation" to $\mathbf{f}'(\mathbf{x}_{n+1})$

Taylor yields $-\mathbf{f}(\mathbf{x}_{n+1}) + \mathbf{f}(\mathbf{x}_n) = \mathbf{f}'(\mathbf{x}_{n+1})(\mathbf{x}_n - \mathbf{x}_{n+1}) + O(\|\mathbf{x}_{n+1} - \mathbf{x}_n\|^2)$ so that we expect $\mathbf{f}'(\mathbf{x}_{n+1})(\mathbf{x}_{n+1} - \mathbf{x}_n) \approx \mathbf{f}(\mathbf{x}_{n+1}) - \mathbf{f}(\mathbf{x}_n)$ Hence, a resonable condition on $\mathbf{H}_{n+1}$ is the "secant condition"

$$\mathbf{H}_{n+1}(\mathbf{x}_{n+1} - \mathbf{x}_n) \overset{!}{=} \mathbf{f}(\mathbf{x}_{n+1}) - \mathbf{f}(\mathbf{x}_n) \tag{6.20}$$

Condition (6.20) does not fix $\mathbf{H}_{n+1}$ (unless $d = 1$). A reasonable further condition is that $\mathbf{H}_{n+1}$ does not deviate much from $\mathbf{H}_n$, i.e., that $\mathbf{H}_{n+1} - \mathbf{H}_n$ be small. This leads to the problem:

$$\text{Find } \mathbf{H}_{n+1} \text{ satisfying (6.20) s.t. } \|\mathbf{H}_{n+1} - \mathbf{H}_n\|_F = \min\{\|\mathbf{A} - \mathbf{H}_n\|_F \,|\, \mathbf{A}(\mathbf{x}_{n+1} - \mathbf{x}_n) = \mathbf{f}(\mathbf{x}_{n+1}) - \mathbf{f}(\mathbf{x}_n)\} \tag{6.21}$$

This constrained minimization problem has a unique solution:

$$\mathbf{H}_{n+1} = \mathbf{H}_n + \frac{1}{\|\mathbf{s}\|_2^2}(\mathbf{y} - \mathbf{H}_n\mathbf{s})\,\mathbf{s}^\top, \quad \mathbf{s} = \mathbf{x}_{n+1} - \mathbf{x}_n, \quad \mathbf{y} = \mathbf{f}(\mathbf{x}_{n+1}) - \mathbf{f}(\mathbf{x}_n). \tag{6.22}$$

The reason is the following, more general result:

Lemma 6.20 *Let $\mathbf{B} \in \mathbb{R}^{d\times d}$, $\mathbf{s}$, $\mathbf{y} \in \mathbb{R}^d$ with $\mathbf{s} \neq 0$. Then the matrix $\mathbf{B}_+ \in \mathbb{R}^{d\times d}$ given by*

$$\mathbf{B}_+ = \mathbf{B} + \frac{1}{\|\mathbf{s}\|_2^2}(\mathbf{y} - \mathbf{B}\mathbf{s})\mathbf{s}^\top \tag{6.23}$$

solves the following constrained minimization problem:

$$\textit{Find the minimizer } \mathbf{A} \textit{ of } \|\mathbf{A} - \mathbf{B}\|_F \textit{ under the constraint } \mathbf{A}\mathbf{s} = \mathbf{y} \tag{6.24}$$

Furthermore, the minimizer is unique.

Proof: We will only show that the given $\mathbf{B}_+$ solves the minimization problem. By construction, $\mathbf{B}_+\mathbf{s} = \mathbf{y}$. For arbitrary $\mathbf{A}$ with $\mathbf{A}\mathbf{s} = \mathbf{y}$, we compute

$$\|\mathbf{B}_+ - \mathbf{B}\|_F = \|\frac{1}{\|\mathbf{s}\|_2^2}(\mathbf{y} - \mathbf{B}\mathbf{s})\mathbf{s}^\top\|_F = \|\frac{1}{\|\mathbf{s}\|_2^2}(\mathbf{A}\mathbf{s} - \mathbf{B}\mathbf{s})\mathbf{s}^\top\|_F = \|(\mathbf{A} - \mathbf{B})\frac{\mathbf{s}^\top}{\|\mathbf{s}\|_2^2}\|_F$$

$$\overset{\|\mathbf{G}\mathbf{H}\|_F \le \|\mathbf{G}\|_F\|\mathbf{H}\|_2}{\le} \|\mathbf{A} - \mathbf{B}\|_F \, \underbrace{\|\frac{\mathbf{s}\mathbf{s}^\top}{\|\mathbf{s}\|_2^2}\|_2}_{=1 \text{ since } \mathbf{s}\mathbf{s}^\top \text{ is sym. with } d-1 \text{ EVs 0 and one EV 1}}$$

□

The update formula (6.20) yields the following *Broyden method*:

1. given $\mathbf{H}_n$ compute $\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{H}_n^{-1}\mathbf{f}(\mathbf{x}_n)$
2. compute $\mathbf{H}_{n+1}$ via (6.22).

Important features of this method are:

1. The method converges (locally) superlinearly, i.e., for some sequence $\varepsilon_n \to 0$ there holds
$$\|\mathbf{x}_{n+1} - \mathbf{x}_n\| \leq \varepsilon_n \|\mathbf{x}_n - \mathbf{x}_{n-1}\|$$
2. The Broyden updates are rank-1 updates. For rank-1 updates of matrices, the inverses can be computed fairly cheaply with the *Sherman-Morrison-Woodbury formula*, which asserts (exercise!) that for arbitrary invertible $\mathbf{A} \in \mathbb{R}^{d\times d}$ and vectors $\mathbf{u}$, $\mathbf{v}$ (with $\mathbf{v}^\top \mathbf{A}^{-1}\mathbf{u} \neq -1$) there holds
$$(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)^{-1} = \mathbf{A}^{-1} - \frac{1}{1 + \mathbf{v}^\top \mathbf{A}^{-1}\mathbf{u}} \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\top \mathbf{A}^{-1}. \tag{6.25}$$

Example 6.21 **slide 20a - Broyden method**
We seek the zero $\mathbf{x}^* = (0,1)^T$ *of*

$$F(\mathbf{x}) = \begin{pmatrix} (x_1+3)(x_2^3 - 7) + 18 \\ \sin(x_2 e^{x_1} - 1) \end{pmatrix} = 0$$

with initial value $\mathbf{x}_0 = (-0.5, 1.4)^T$. *The classical Broyden method is started with* $\mathbf{H}_0 = F'(\mathbf{x}_0)$. *One observes in Fig. 6.3 in particular superlinear convergence of the Broyden method. For comparison purposes also the gradient method (steepest descent) for* $f(x) := \|F(x)\|_2^2$ *with* $\sigma = 0.9$ *and* $q = 0.5$ *(see Sec. 6.8.1) is shown.*

Remark 6.22 *There are many important variations of the Broyden method. Consider for example the case that Newton's method is applied to find the minimum of a function f (see Section 6.8.1). Then the Hessian of f is symmetric and — at least in the vicinity of the sought minimum — positive definite. One would like to make Broyden-like updates that preserve symmetric and positive definiteness. Such methods exist: see PSB ("Powell symmetric Broyden"), DFP ("Davidson-Fletcher-Powell"), BFGS ("Broyden-Fletcher-Goldfarb-Shanno").*

Remark 6.23 *Just like globalized Newton methods, Broyden and Broyden-like methods are in practice combined with algorithms that select the step length.*

6.8 Unconstrained minimization problems (CSE)

goal: minimize a function $f : \mathbb{R}^d \to \mathbb{R}$
This problem can be approached in several ways, for example:

1. The minimizer satisfies $\nabla f(\mathbf{x}^*) = 0$ so that a (globalized) Newton method could be used. We note that then the Hessian of f is required.

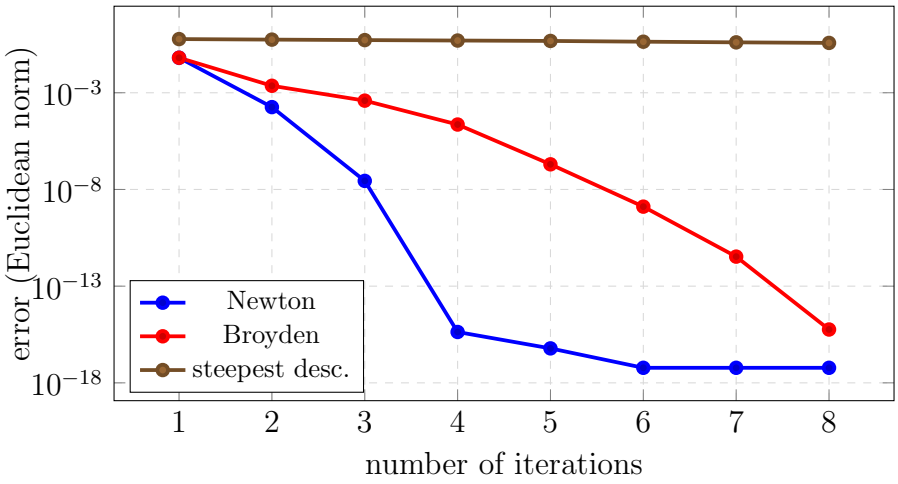


Figure 6.3: Comparison of Newton method, Broyden method, and gradient method (See Example 6.21).

2. Descent method: These methods identify a descent direction for f (e.g., $-\nabla f(\mathbf{x}_n)$) and then make a step that reduces f. These methods typically require only ∇f and are discussed in Sec. 6.8.1.

3. Trust region methods: these methods approximate f locally by a quadratic function that is minimized in a region where the quadratic approximation is deemed reliable. This is sketched in Sec. 6.8.3.

6.8.1 Gradient methods

The simplest minimization strategy is the following iteration, starting with an initial point $\mathbf{x}_0$:

1. select a *search direction* $\mathbf{d}_n$ with $\nabla f(\mathbf{x}_n) \cdot \mathbf{d}_n < 0$

2. select a *step length* λ_n such that $f(\mathbf{x}_n + \lambda_n \mathbf{d}_n) < f(\mathbf{x}_n)$

Concerning the search direction $\mathbf{d}_n$, the simplest one is the negative gradient: $\mathbf{d}_n = -\nabla f(\mathbf{x}_n)$. This is called the *steepest descent direction*.
There are many choices for the step length λ_n. The "greedy" approach is to take λ_n as the minimizer of 1D optimization problem:

$$\text{minimize } t \mapsto \varphi(t) := f(\mathbf{x}_n + t\mathbf{d}_n). \tag{6.26}$$

Since this minimization problem is typically still difficult to solve, various simplified versions are employed. A typical condition imposed on the step length λ_n is that each step make sufficient descent, namely,

$$f(\mathbf{x}_n + \lambda_n \mathbf{d}_n) < f(\mathbf{x}_n) + \lambda_n \sigma \nabla f(\mathbf{x}_n) \cdot \mathbf{d}_n \tag{6.27}$$

for some user chosen parameter σ. That is, the reduction in f should be proportional to the step size as well as the directional derivative $\nabla f(\mathbf{x}_n) \cdot \mathbf{d}_n$. One popular technique to ensure this is the *Armijo*-rule: Given $q \in (0, 1)$, one selects λ_n as the largest number of the form $\lambda_n = q^k$, $k \in \mathbb{N}_0$, such that the condition (6.27) is satisfied.

6.8.2 Gradient method with quadratic cost function

We consider the special case of a quadratric function f:

$$f(\mathbf{x}) = \gamma + \mathbf{c}^\top \mathbf{x} + \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} \tag{6.28}$$

where $\gamma \in \mathbb{R}$, $\mathbf{c} \in \mathbb{R}^d$, $\mathbf{Q}$ is SPD. ⟦ note: in the vicinity of a minimum of f, one expects f to be close to a quadratic polynomial of this form by Taylor ⟧. We employ as the search direction $\mathbf{d}_n := -\nabla f(\mathbf{x}_n)$. Rather than using the Armijo rule, we use the minimum rule since the minimum can be computed: The minimum of $\varphi : t \mapsto f(\mathbf{x}_n + t\mathbf{d}_n)$ is explicitly given by

$$t = -\frac{f(\mathbf{x}_n) \cdot \mathbf{d}_n}{\mathbf{d}_n^\top \mathbf{Q}\mathbf{d}_n}$$

since

$$\begin{aligned}
\varphi(t) &= f(\mathbf{x}_n + t\mathbf{d}_n) = f(\mathbf{x}_n) + t\nabla f(\mathbf{x}_n) \cdot \mathbf{d}_n + \frac{1}{2}t^2 \mathbf{d}_n^\top \mathbf{Q}\mathbf{d}_n, \\
\varphi'(t) &= \nabla f(\mathbf{x}_n) \cdot \mathbf{d}_n + t\mathbf{d}_n^\top \mathbf{Q}\mathbf{d}_n;
\end{aligned}$$

therefore, one step of the gradient method is

$$\mathbf{x}_{n+1} = \mathbf{x}_n + t\mathbf{d}_n = \mathbf{x}_n - \frac{\nabla f(\mathbf{x}_n) \cdot \mathbf{d}_n}{\mathbf{d}_n^\top \mathbf{Q}\mathbf{d}_n}\mathbf{d}_n$$

The convergence can be estimate:

Lemma 6.24 *Let f be given by (6.28) with an SPD matrix $\mathbf{Q}$. Consider steepest descent, i.e., $\mathbf{d}_n := -\nabla f(\mathbf{x}_n)$. Then:*

$$\begin{aligned}
f(\mathbf{x}_{n+1}) - f(\mathbf{x}^*) &\le \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right)^2 (f(\mathbf{x}_n) - f(\mathbf{x}^*)) = \left(\frac{\kappa - 1}{\kappa + 1}\right)^2 (f(\mathbf{x}_n) - f(\mathbf{x}^*)), \\
\|\mathbf{x}_{n+1} - \mathbf{x}^*\|_{\mathbf{Q}}^2 &\le \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right)^2 \|\mathbf{x}_n - \mathbf{x}^*\|_{\mathbf{Q}}^2 = \left(\frac{\kappa - 1}{\kappa + 1}\right)^2 \|\mathbf{x}_n - \mathbf{x}^*\|_{\mathbf{Q}}^2,
\end{aligned}$$

where $\|\mathbf{z}\|_Q^2 = \mathbf{z}^\top \mathbf{Q}\mathbf{z}$ and $\kappa = \lambda_{max}/\lambda_{min}$ is the condition number of $\mathbf{Q}$.

Proof: Literature. □

Lemma 6.24 shows that the steepest descent method degrades if $\mathbf{Q}$ has widely differing eigenvalues (i.e., large condition number κ). This problem can be solved or at least mitigated by selecting the search directions in a different way. In fact, if one takes an SPD matrix $\mathbf{H}$ (as a "preconditioner") and considers as the search direction

$$\mathbf{d}_n = -\mathbf{H}\nabla f(\mathbf{x}_n)$$

then, one can show that

$$f(\mathbf{x}_{n+1}) - f(\mathbf{x}^*) \le \left(\frac{\lambda_{max}(\mathbf{H}^{-1}\mathbf{Q}) - \lambda_{min}(\mathbf{H}^{-1}\mathbf{Q})}{\lambda_{max}(\mathbf{H}^{-1}\mathbf{Q}) + \lambda_{min}(\mathbf{H}^{-1}\mathbf{Q})}\right)^2 (f(\mathbf{x}_n) - f(\mathbf{x}^*)),$$

so that the contraction factor can be much smaller than in the unpreconditioned case. The extreme case $\mathbf{H} = \mathbf{Q}$ leads to convergence in one step.

Remark 6.25 *The minimization of the quadratic function f can be done explicitly with solution $x^* = -\mathbf{Q}^{-1}\mathbf{c}$ so that a (steepest) descent method seems useless. Nevertheless, the discussion of quadratic functions f is of interest as it indicates weaknesses of the steepest descent methods for general f: one should expect slow convergence if, for example, the Hessian of f has a large condition number.*

Returning to the quadratic problem, it is of interest to note that the minimum can also be found as the zero of the function $\mathbf{x} \mapsto \nabla f(\mathbf{x})$. This is a linear function. The Hessian of f is $\mathbf{H} = \mathbf{Q}$. Applying the Newton method yields convergence in one step. The Newton step is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{H}^{-1}\nabla f(\mathbf{x}_n).$$

This is precisely the preconditioned gradient method with the above identified optimal preconditioner $\mathbf{H} = \mathbf{Q}$.

6.8.3 Trust region methods

starting point: many minimization techniques are based on "sequential quadratic programming", i.e., the function f is approximated locally by a quadratric "model" of the form

$$q_k(x) = f(x_k) + g_k \cdot (x - x_k) + \frac{1}{2}(x - x_k)^T B_k (x - x_k), \tag{6.29}$$

that is then minimized instead. Examples are:

- $g_k = \nabla f(x_k)$ and $B_k = \mathbf{H}(x_k)$, where $\mathbf{H}(x_k)$ is the Hessian of f at x_n: $\rightarrow$ Newton's method if $\mathbf{H}(x_k)$ SPD
- $g_k = \nabla f(x_k)$ and $B_k = \mathrm{Id}$: $\rightarrow$ gradient method (with step length $t_k = 1$)

Problems:

- the quadratic model is only valid in a small region near x_k. Too large steps of the minimization algorithm may lead to leaving the region of validity of the model.
- If B_k is not SPD, then the minimization problem is not meaningful.

In *trust region methods* the model q_k is not minimized over $\mathbb{R}^d$ but merely on a ball $B_{\Delta_k}(x_k)$ for given Δ_k:

$$\text{Minimize} \quad q_k(x) \quad \text{under the constraint } \|x_{k+1} - x_k\| \leq \Delta_k. \tag{6.30}$$

- (6.30) has a solution
- key ingredient of the algorithm is the steering of the Δ_k.
- in order to assess whether the quadratic model is "good", one defines

$$\rho_k := \frac{f(x_k) - f(x_{k+1})}{q_k(x_k) - q_k(x_{k+1})}. \tag{6.31}$$

⟦ = ratio of actual descent and descent predicted by the model ⟧
⟦ denominator is always non-negative ⟧ If the model is "good", then $\rho_k \approx 1$ will be close to 1. In particular, for $\rho_k \leq 0$ no descent is achieved (since the denominator is positive!).
In trust region methods, the search directions and the step lengths are not selected separately. Rather, they are selected in some sense simultaneously.

Algorithm 6.26 (Trust region method) *%input* $\widehat{\Delta}$, $\Delta_0 \in (0, \widehat{\Delta})$, $\eta \in [0, 1/4)$

for $k = 0, 1, \ldots$ **do** {
minimize q_k *with minimizer* $\widehat{x}_{k+1}$
$\rho_k = (f(\widehat{x}_{k+1}) - f(x_k))/(q_k(\widehat{x}_{k+1}) - q_k(x_k))$
if $\rho_k < 1/4$ **then** $\Delta_{k+1} := \frac{1}{4}\Delta_k$ *% Model "bad" → reduce trust region*
else if *(*$\rho_k > 3/4$ **and** $\|\widehat{x}_{k+1} - x_k\| = \Delta_k$*)* **then** $\Delta_{k+1} = \min(2\Delta_k, \widehat{\Delta})$
% model "good", minimizer at boundary → trust region apparently too small
else $\Delta_{k+1} = \Delta_k$
if $\rho_k > \eta$ **then** $x_{k+1} := \widehat{x}_{k+1}$ *% model OK, → accept step*
else $x_{k+1} := x_k$ *% model not OK → reject the step*
}

Remark 6.27 *The actual realization of a trust region method is non-trivial as the constrained minimization problem of finding* $\widehat{x}_{k+1}$ *has to be (approximately) solved. For actual realizations of trust region methods: see literature.*

7 Eigenvalue Problems

goal: compute some or all eigenvalues of $\mathbf{A} \in \mathbb{R}^{n\times n}$

7.1 The power method

goal: compute largest (in absolute value) eigenvalue and corresponding eigenvector

Algorithm 7.1 (power method)

$$\begin{array}{rcl} \%\textit{input} & : & \mathbf{A} \in \mathbb{R}^{n\times n},\ 0 \neq \mathbf{x}_0 \in \mathbb{R}^n \\ \ell := 0 & ; & \mathbf{x}_0 := \dfrac{\mathbf{x}_0}{\|\mathbf{x}_0\|_2} \quad , \quad \tilde{\lambda}_0 := \mathbf{x}_0^H \mathbf{A}\mathbf{x}_0 \\ \textit{repeat} & \{ & \mathbf{x}_{\ell+1} := \dfrac{\mathbf{A}\mathbf{x}_\ell}{\|\mathbf{A}\mathbf{x}_\ell\|_2} \qquad \%\ \textit{approx. eigenvector} \\ & & \tilde{\lambda}_{\ell+1} := \mathbf{x}_{\ell+1}^H \mathbf{A}\mathbf{x}_{\ell+1} \quad \%\ \textit{approx. eigenvalue} \\ & & \ell := \ell + 1 \\ & & \}\ \textit{until sufficiently accurate} \end{array} \tag{7.1}$$

Theorem 7.2 *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ have a basis of eigenvectors (i.e., $\mathbf{A}$ is diagonalizable) $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ with eigenvalues $\lambda_1, \ldots, \lambda_n$ satisfying $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$. Let $\mathbf{x}_0 = \sum_{i=1}^n \alpha_i \mathbf{v}_i$ with $\alpha_1 \neq 0$ Then:*

(i) The $\mathbf{x}_\ell$ of Alg. 7.1 are well-defined.

(ii) $\exists C > 0$ s.t. $|\tilde{\lambda}_\ell - \lambda_1| \leq C|\frac{\lambda_2}{\lambda_1}|^\ell,\ \ell = 0, 1, \ldots$

Proof: $\mathbf{x}_0 = \sum_i \alpha_i \mathbf{v}_i \ \Rightarrow\ \mathbf{A}^\ell \mathbf{x}_0 = \sum_i \alpha_i \lambda_i^\ell \mathbf{v}_i$. The assumption $\alpha_1 \neq 0 \wedge \lambda_1 \neq 0$ implies $\mathbf{A}^\ell \mathbf{x}_0 \neq 0 \forall \ell$. Inductively, this implies that $\mathbf{x}_\ell \neq 0$ for all ℓ and that $\mathbf{x}_\ell = c_\ell \mathbf{A}^\ell \mathbf{x}_0$ for $c_\ell := 1/\|\mathbf{A}^\ell \mathbf{x}_0\|_2 \neq 0$. Therefore:

$$\mathbf{x}_\ell = c_\ell \alpha_1 \lambda_1^\ell \left(\mathbf{v}_1 + \underbrace{\sum_{i=2}^n \frac{\alpha_i}{\alpha_1} \left(\frac{\lambda_i}{\lambda_1}\right)^\ell \mathbf{v}_i}_{=:\epsilon_\ell} \right). \tag{7.2}$$

The assumption $|\lambda_i| \leq |\lambda_2| < |\lambda_1|\ \forall i = 2, \ldots, n$ then implies

$$\|\epsilon_\ell\|_2 \leq \sum_{i=2}^n \left|\frac{\alpha_i}{\alpha_1}\right| \left|\frac{\lambda_i}{\lambda_1}\right|^\ell \|\mathbf{v}_i\|_2 \leq C \left|\frac{\lambda_2}{\lambda_1}\right|^\ell \text{ for suitable } C > 0. \tag{7.3}$$

For ℓ large, we have that $\|\epsilon_\ell\|_2$ is small $\Rightarrow$

$$\begin{aligned} \tilde{\lambda_\ell} &= \mathbf{x}_\ell^H \mathbf{A}\mathbf{x}_\ell \overset{\|\mathbf{x}_\ell\|_2=1}{=} \frac{\mathbf{x}_\ell^H \mathbf{A}\mathbf{x}_\ell}{\|\mathbf{x}_\ell\|_2^2} = \frac{(\mathbf{v}_1+\epsilon_\ell)^H \mathbf{A}(\mathbf{v}_1+\epsilon_\ell)}{\|\mathbf{v}_1+\epsilon_\ell\|_2^2} = \frac{\mathbf{v}_1^H\mathbf{A}\mathbf{v}_1 + \mathbf{v}_1^H\mathbf{A}\epsilon_\ell + \epsilon_\ell^H\mathbf{A}\mathbf{v}_1 + \epsilon_\ell^H\mathbf{A}\epsilon_\ell}{||\mathbf{v}_1+\epsilon_\ell||_2^2} = \\ &= \frac{\lambda_1\|\mathbf{v}_1\|_2^2 + O(\|\epsilon_\ell\|_2)}{\|\mathbf{v}_1\|_2^2 + O(\|\epsilon_\ell\|_2)} = \lambda_1 + O(\|\epsilon_\ell\|_2) \end{aligned}$$

Hence, $|\lambda_1 - \tilde{\lambda}_\ell| \;\leq\; C\|\epsilon_\ell\|_2 \;\leq\; C\left|\frac{\lambda_2}{\lambda_1}\right|^\ell$. $\square$

Remark 7.3 *1. Since $\mathbf{v}_1$ is not known, the requirement $\alpha_1 \neq 0$ cannot be checked. In practice, this is not a problem since:*

- *a randomly chosen $\mathbf{x}_0$ satisfies $\alpha_1 \neq 0$ with probability 1*
- *rounding errors create a component in the direction of $\mathbf{v}_1$*

2. analogous result holds for the eigenvalue converge if λ_1 is a multiple eigenvalue

3. Algorithm 7.1 does not converge, if $\lambda_1 \neq \lambda_2$ but $|\lambda_1| = |\lambda_2|$. This case arises, e.g., when $\mathbf{A} \in \mathbb{R}^{n\times n}$ but $\mathbf{A}$ has complex eigenvalues.

4. greatest weakness of Algorithm 7.1: slow convergence if λ_1 is not well-separated from $\sigma(\mathbf{A}) \setminus \{\lambda_1\}$, i.e., $\left|\frac{\lambda_2}{\lambda_1}\right|$ is close to 1.

5. common application: estimate $\|\mathbf{A}\|_2^2 = \lambda_{max}(\mathbf{A}^H\mathbf{A})$

In addition to providing approximations to the largest eigenvalue, Algorithm 7.1 also yields an approximation to the corresponding eigenvector. To capture this convergence mathematically, we introduce the notion of "distance" between the spaces spanned by two vectors:

Definition 7.4 *Let $\{0\} \neq \mathcal{S} = \text{span}\{\mathbf{x}\}$ and $\{0\} \neq \mathcal{T} = \text{span}\{\mathbf{y}\}$. We define*

$$d(\mathcal{S},\mathcal{T}) := |\sin\varphi| = \sqrt{1-\cos^2\varphi}, \qquad \cos\varphi = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|_2\|\mathbf{y}\|_2}.$$

Remark 7.5 (geometric intepretation) *φ is the angle between the vectors $\mathbf{x}$ and $\mathbf{y}$. If $\mathbf{x} \parallel \mathbf{y}$, then $\varphi = 0$, i.e., $\mathcal{S} = \mathcal{T}$ and indeed $d(\mathcal{S},\mathcal{T}) = 0$. If $\mathbf{x} \perp \mathbf{y}$, then $d(\mathcal{S},\mathcal{T}) = 1$.*

The following Theorem 7.6 shows that $|\sin\angle(\mathbf{v}_1, \mathbf{x}_\ell)| \to 0$:

Theorem 7.6 *Assumptions as in Theorem 7.2. Then $\exists\, C > 0$ such that*

$$d(\text{span}\{\mathbf{v}_1\},\ \text{span}\{\mathbf{x}_\ell\}) \;\leq\; C\left|\frac{\lambda_2}{\lambda_1}\right|^\ell,\ \ell = 0, 1, \ldots$$

Proof: From (7.2), we get $\text{span}\{\mathbf{x}_\ell\} = \text{span}\{\mathbf{v}_1 + \epsilon_\ell\}$. Hence from (7.3) and a calculation

$$d(\text{span}\{\mathbf{x}_\ell\}, \text{span}\{\mathbf{v}_1\}) \leq \frac{\|\epsilon_\ell\|_2}{\|\mathbf{v}_1 + \epsilon_\ell\|_2} \leq C\left|\frac{\lambda_2}{\lambda_1}\right|^\ell$$

$\square$

7.2 Inverse Iteration

goal: eigenvalue other than the largest one
observation: if $\mathbf{A}$ is invertible and $\sigma(\mathbf{A}) = \{\lambda_i \,|\, i = 1, \ldots, n\}$ then $\sigma(\mathbf{A}^{-1}) = \{\frac{1}{\lambda_i} \,|\, i = 1, \ldots, n\}$ i.e., the largest (in absolute value) eigenvalue of $\mathbf{A}^{-1}$ is the reciprocal of the smallest one (in absolute value) of $\mathbf{A}$.

Algorithm 7.7 (inverse Iteration) $\ell := 0, \qquad \mathbf{x}_0 := \frac{\mathbf{x}_0}{\|\mathbf{x}_0\|_2}$
repeat {

- *solve* $\mathbf{A}\tilde{\mathbf{x}}_{\ell+1} = \mathbf{x}_\ell$
- $\mathbf{x}_{\ell+1} := \frac{\tilde{\mathbf{x}}_{\ell+1}}{\|\tilde{\mathbf{x}}_{\ell+1}\|_2}$
- $\tilde{\lambda}_{\ell+1} := \mathbf{x}_{\ell+1}^H \mathbf{A}\mathbf{x}_{\ell+1}$
- $\ell := \ell + 1$

} *until sufficiently accurate*

Remark 7.8 *1. If* $0 < |\lambda_n| < |\lambda_{n-1}| \leq \cdots \leq |\lambda_1|$*, then, analogous to Theorem 7.2, one has*
$|\lambda_n - \tilde{\lambda}_l| \leq C \left|\frac{\lambda_n}{\lambda_{n-1}}\right|^\ell$ $\qquad [\![$ *exercise* $]\!]$

2. since a linear system is solved in each step $\rightarrow$ *perform an LU-factorization of* $\mathbf{A}$ *at the beginning*

The inverse iteration is a special case of an inverse iteration with shift:

Algorithm 7.9 (inverse iteration with shift) *% input* $\mathbf{A} \in \mathbb{R}^{n\times n}$*, shift* $\lambda \in \mathbb{R}$*,* $\mathbf{x}_0 \in \mathbb{R}^n \backslash \{0\}$
$\ell := 0$ *;* $\mathbf{x}_0 := \frac{\mathbf{x}_0}{\|\mathbf{x}_0\|_2}$

repeat {

- *solve* $(\mathbf{A} - \lambda)\tilde{\mathbf{x}}_{\ell+1} = \mathbf{x}_\ell$
- $\mathbf{x}_{\ell+1} := \frac{\tilde{\mathbf{x}}_{\ell+1}}{\|\tilde{\mathbf{x}}_{\ell+1}\|_2}$
- $\tilde{\lambda}_{\ell+1} := \mathbf{x}_{\ell+1}^H \mathbf{A}\mathbf{x}_{\ell+1}$
- $\ell := \ell + 1$

} *until sufficiently accurate*

Theorem 7.10 *Let* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *be diagonalizable;* $\lambda \in \mathbb{R}$*. Let the eigenvalues of* $\mathbf{A}$ *be numbered such that* $|\lambda_1 - \lambda| \geq |\lambda_2 - \lambda| \geq \cdots \geq |\lambda_{n-1} - \lambda| > |\lambda_n - \lambda| > 0$*.*
Then: $\exists\, C > 0$ *such that the approximation* $\tilde{\lambda}_\ell$ *computed by Algorithmus 7.9 satisfies:*

$$|\lambda_n - \tilde{\lambda}_\ell| \leq C \left|\frac{\lambda_n - \lambda}{\lambda_{n-1} - \lambda}\right|^\ell$$

Proof: analogous to that of Theorem 7.2. □

observation:

- inverse iteration with shift converges to the eigenvalue closest to the shift parameter $\lambda \rightarrow$ it is possible to seek specific eigenvalues
- the closer λ is to an eigenvalue, the faster the convergence

idea: use, in each step of the iteration, as a shift parameter λ the best available approximation to an eigenvalue $\rightarrow$ Rayleigh quotient iteration with shift $\lambda_\ell = \mathbf{x}_\ell^H \mathbf{A}\mathbf{x}_\ell$

Algorithm 7.11 (Rayleigh quotient iteration) *% input* $\mathbf{A} \in \mathbb{R}^{n\times n}$, $0 \neq \mathbf{x}_0 \in \mathbb{R}^n$, *(=initial guess for eigenvector corresponding to sought eigenvalue)*
$\ell := 0$; $\mathbf{x}_0 := \frac{\mathbf{x}_0}{\|\mathbf{x}_0\|_2}$
repeat {

- $\tilde{\lambda}_\ell := \mathbf{x}_\ell^H \mathbf{A} x_\ell$
- *solve* $(\mathbf{A} - \tilde{\lambda}_\ell)\tilde{\mathbf{x}}_{\ell+1} = \mathbf{x}_\ell$
- $\mathbf{x}_{\ell+1} := \frac{\tilde{\mathbf{x}}_{\ell+1}}{\|\tilde{\mathbf{x}}_{\ell+1}\|_2}$

} *until sufficiently accurate*

One expects better convergence of the Rayleigh quotient iteration than in the case of a fixed shift. One has, for example:

Theorem 7.12 *Let* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *be symmetric,* $\lambda \in \sigma(\mathbf{A})$ *be a simple eigenvalue with corresponding eigenspace* $\mathrm{span}\{\mathbf{v}\}$. *Then:* $\exists\, C > 0,\ \epsilon_0 > 0$ *such that* $\forall \epsilon \in (0, \epsilon_0)$*: If* $\mathbf{x}_0 \in \mathbb{R}^n \backslash \{0\}$ *satisfies the condition* $d(\mathrm{span}\{\mathbf{x}_0\},\ \mathrm{span}\{\mathbf{v}\}) < \epsilon$, *then* $\mathbf{x}_1$ *(= one step of Algorithm 7.11) satisfies*

$$d(\mathrm{span}\{\mathbf{x}_1\},\ \mathrm{span}\{\mathbf{v}\}) \leq C\epsilon^3 \qquad \textit{and} \qquad \left| \frac{\mathbf{x}_0^H \mathbf{A}\mathbf{x}_0}{\|\mathbf{x}_0\|_2^2} - \lambda \right| \leq C\epsilon^2.$$

Proof: See literature. Note in particular, that the result implies $\left| \frac{\mathbf{x}_1^H \mathbf{A}\mathbf{x}_1}{\|\mathbf{x}_1\|_2^2} - \lambda \right| \leq C\epsilon^6$. □

Remark 7.13
1. *Analogous result holds also for general diagonalizable matrices: One then has locally quadratic (instead of cubic) convergence.*
2. *Iterations with variable shift are more expensive than those with fixed shift for which a factorization can be amortized over several iterations.*

slide 21 - Vector and QR iteration

7.3 error estimates–stopping criteria

7.3.1 Bauer-Fike

Question:
Relation of $\sigma(\mathbf{A})$ and $\sigma(\mathbf{A}+\Delta\mathbf{A})$?

Theorem 7.14 (Bauer–Fike) *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be diagonalizable, i.e., $\exists\, \mathbf{T} \in \mathbb{R}^{n\times n}$ with $\mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \operatorname{diag}(\lambda_1,\dots,\lambda_n) =: \mathbf{D}$. Then: Let $\Delta\mathbf{A} \in \mathbb{R}^{n\times n}$. Then for any $\mu \in \sigma(\mathbf{A}+\Delta\mathbf{A})$ there holds $\min|\mu-\lambda_i| \le \operatorname{cond}_p(\mathbf{T})\|\Delta\mathbf{A}\|_p$, where $\operatorname{cond}_p(\mathbf{T}) = \|\mathbf{T}\|_p\|\mathbf{T}^{-1}\|_p$ and $p \in [1,\infty]$ arbitrary.*

Proof: Without loss of generality let $\mu \in \sigma(\mathbf{A}+\Delta\mathbf{A}) \setminus \sigma(\mathbf{A})$. Let $\mathbf{v}$ be an eigenvector with eigenvalue μ. Then:

$$((\mathbf{A}+\Delta\mathbf{A}) - \mu I)\,\mathbf{v} = 0 \quad \Rightarrow \quad ((\mathbf{A}-\mu\mathbf{I}) + \Delta\mathbf{A})\,\mathbf{v} = 0 \quad \Rightarrow \quad \left(\mathbf{I} + (\mathbf{A}-\mu)^{-1}\Delta\mathbf{A}\right)\mathbf{v} = 0 \quad \Rightarrow$$

$$\begin{aligned}
1 \quad &= \quad \frac{\|\mathbf{I}\mathbf{v}\|_p}{\|\mathbf{v}\|_p} = \frac{\|(\mathbf{A}-\mu)^{-1}\Delta\mathbf{A}\mathbf{v}\|_p}{\|\mathbf{v}\|_p} \;\le\; \|(\mathbf{A}-\mu)^{-1}\|_p \frac{\|\Delta\mathbf{A}\mathbf{v}\|_p}{\|\mathbf{v}\|_p} \\
&\overset{\mathbf{A}=\mathbf{T}^{-1}\mathbf{D}\mathbf{T}}{\le} \quad \| \left(\mathbf{T}^{-1}(\mathbf{D}-\mu)\mathbf{T}\right)^{-1} \|_p \|\Delta\mathbf{A}\|_p \;\le\; \|\mathbf{T}^{-1}\|_p \|(\mathbf{D}-\mu)^{-1}\|_p \|\mathbf{T}\|_p \|\Delta\mathbf{A}\|_p \\
&= \quad \|\Delta\mathbf{A}\|_p \operatorname{cond}_p(\mathbf{T}) \| \underbrace{(\mathbf{D}-\mu)^{-1}}_{diag.} \|_p = \|\Delta\mathbf{A}\|_p \operatorname{cond}_p(\mathbf{T}) \max_{i=1,\dots,n} \frac{1}{|\lambda_i - \mu|} \\
&= \quad \frac{1}{\min_i(\lambda_i-\mu)} \|\Delta\mathbf{A}\|_p \operatorname{cond}_p(\mathbf{T})
\end{aligned}$$

□

Remark 7.15 $\operatorname{cond}_p(\mathbf{T})$ *can be large if the eigenvectors of $\mathbf{A}$ are close to being linearly dependent. This does* not *happen in the self-adjoint (symmetric) case:*

Corollary 7.16 *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be self-adjoint (symmetric), $\Delta\mathbf{A} \in \mathbb{R}^{n\times n}$. Then:*

$$\forall\, \mu \in \sigma(\mathbf{A}+\Delta\mathbf{A}) \quad : \quad \min_{\lambda\in\sigma(\mathbf{A})} |\mu-\lambda| \;\le\; \|\Delta\mathbf{A}\|_2$$

Proof: $\mathbf{A}$ selfadjoint $\Rightarrow \mathbf{A} = \mathbf{Q}^H\mathbf{D}\mathbf{Q}$ with $\mathbf{Q}$ orthogonal, i.e., $\operatorname{cond}_2(\mathbf{Q}) = 1$ □

7.3.2 remarks on stopping criteria

A pair $(\mathbf{x},\tilde\lambda) \in \mathbb{R}^n \setminus \{0\} \times \mathbb{R}$ is an eigenpair, if $\mathbf{A}\mathbf{x} - \tilde\lambda\mathbf{x} = 0$
hope: For $(\mathbf{x},\tilde\lambda)$ not necessarily an eigenpair, the residual $\mathbf{A}\mathbf{x} - \tilde\lambda\mathbf{x}$ is a useful measure for the deviation from an eigenpair. We have

Theorem 7.17 *$\mathbf{A} \in \mathbb{R}^{n\times n}$ diagonalizable, $(\mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \mathbf{D})$, $\|\mathbf{x}\|_2 = 1, \tilde\lambda \in \mathbb{R}$. Set $\mathbf{r} := \mathbf{A}\mathbf{x} - \tilde\lambda\mathbf{x}$. Then:*

(i) $\min_{\lambda\in\sigma(\mathbf{A})} |\lambda - \tilde{\lambda}| \leq \text{cond}_2(T)\|\mathbf{r}\|_2$

(ii) $\min_{\lambda\in\sigma(\mathbf{A})} |\lambda - \tilde{\lambda}| \leq \|\mathbf{r}\|_2$ *if* $\mathbf{A}$ *is selfadjoint (symmetric).*

(iii) *If* $\tilde{\lambda} = \mathbf{x}^H\mathbf{A}\mathbf{x}$ *and* $\mathbf{A}$ *is selfadjoint and* $\tilde{\lambda}$ *sufficiently close to a simple eigenvalue of* $\mathbf{A}$*, then*

$$\min_{\lambda\in\sigma(\mathbf{A})} |\lambda - \tilde{\lambda}| \leq C\|\mathbf{r}\|_2^2$$

Proof: *ad* (i): (perturbation argument)
The matrix $\mathbf{A} + \Delta\mathbf{A} := \mathbf{A} - \mathbf{r}\mathbf{x}^H$ satisfies

- $\|\Delta\mathbf{A}\|_2 = \|\mathbf{r}\|_2$
- $\tilde{\lambda} \in \sigma(\mathbf{A} + \Delta\mathbf{A})$, since $(\mathbf{A} + \Delta\mathbf{A})\mathbf{x} = \mathbf{A}\mathbf{x} - \mathbf{r}\underbrace{\mathbf{x}^H\mathbf{x}}_{=1} = \tilde{\lambda}\mathbf{x}$

The claim follows from Bauer-Fike (Theorem 7.14).
ad (ii): follows from (i)
ad (iii): see literature. □

7.4 orthogonal Iteration

recall: the power iteration generates a sequence $(\mathbf{A}^\ell \text{span}\{\mathbf{x}_0\})_{l=0}^\infty$ of 1-D spaces that converge to an invariant subspace of the matrix $\mathbf{A}$ (in fact, the eigenspace corresponding to the largest eigenvalue).

Idea: Perform power iteration on a k-dimensional space (described by $\mathbf{X}_0 \in \mathbb{R}^{n\times k}$)
Hope: The sequence $(\mathbf{A}^\ell\mathbf{X}_0)_{\ell=0}^\infty$ of k-dimensional spaces converges[1] to the invariant subspace that is spanned by the k dominant eigenvectors.
essential for the numerical realization:
The power iteration in Sec. 7.1 used a normalization of the vector in each space (i.e., an ONB of the space spanned by $\mathbf{A}^\ell\mathbf{x}_0$ was created). Here, an ONB of the space spanned by the columns of $\mathbf{A}^\ell\mathbf{X}_0$ is created.

Algorithm 7.18 (orthogonal iteration) *% input:* $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{X}_0 \in \mathbb{R}^{n\times k}$ *with linearly independent columns.*
$\ell := 0$
$\mathbf{X}_0 =: \mathbf{Q}_0\mathbf{R}_0$*, where* $\mathbf{Q}_0 \in \mathbb{R}^{n\times k}$ *has orthogonal columns,* $\mathbf{R} \in \mathbb{R}^{k\times k}$ *upper triangular.*
repeat {

$\mathbf{X}_{\ell+1} := \mathbf{A}\mathbf{Q}_\ell$

$\mathbf{X}_{\ell+1} =: \mathbf{Q}_{\ell+1}\mathbf{R}_{\ell+1}$ *% reduced QR-decomposition of* $\mathbf{X}_{\ell+1}$ *:* $\mathbf{Q}_{\ell+1} \in \mathbb{R}^{n\times k}$ *has orthogonal columns* $\mathbf{R}_{\ell+1} \in \mathbb{R}^{k\times k}$ *is upper triangular*

[1] actually, we haven't introduced the notion of distance on the space of k-dimensional spaces, so that this statement has to remain vague

$\ell := \ell + 1$

} **until** *sufficiently accurate*

Remark 7.19 *1. The columns of $\mathbf{Q}_\ell$ form an ONB of the space $\mathbf{A}^\ell \mathcal{S}^0$ where $\mathcal{S}^0$ is the space spanned by the columns of $\mathbf{X}_0$.*

2. Orthogonalization is numerically essential: without orthogonalization one performs only k independent vector iterations that all converge to the same dominant eigenspace.

Theorem 7.20 *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be diagonalizable, $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ basis of $\mathbb{R}^n$ of eigenvectors with corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$. Let $|\lambda_1| \geq |\lambda_2| \geq \ldots |\lambda_k| > |\lambda_{k+1}| \geq \cdots \geq |\lambda_n|$.*
Let $\mathcal{S}^0 \subset \mathbb{R}^n$ be the k-dimensional subspace spanned by the columns of $\mathbf{X}_0 \in \mathbb{R}^{n\times k}$ and assume $\mathcal{S}^0 \cap \operatorname{span}\{\mathbf{v}_{k+1}, \ldots, \mathbf{v}_n\} = \{0\}$. Then, there exists $C > 0$ such that the k eigenvalues $\widetilde{\lambda}_{i,\ell}$, $i = 1, \ldots, k$, of $\mathbf{Q}_\ell^H \mathbf{A} \mathbf{Q}_\ell$ satisfy

$$\min_{\lambda\in\sigma(\mathbf{A})} |\widetilde{\lambda}_{i,\ell} - \lambda| \leq C \left|\frac{\lambda_{k+1}}{\lambda_k}\right|^\ell, \qquad i = 1, \ldots, k, \quad \ell = 0, 1, \ldots,$$

Furthermore, for any matrix $\mathbf{Q}'_\ell \in \mathbb{R}^{n\times(n-k)}$ such that $(\mathbf{Q}_\ell, \mathbf{Q}'_\ell)$ is an orthogonal matrix, one has for the block matrix

$$\mathbf{A}_\ell := (\mathbf{Q}_\ell, \mathbf{Q}'_\ell)^H \mathbf{A} (\mathbf{Q}_\ell, \mathbf{Q}'_\ell) = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix}$$

that

$$\|\mathbf{A}_{21}\|_2 \leq C \left|\frac{\lambda_{k+1}}{\lambda_k}\right|^\ell.$$

Proof: see literature. □

Remark 7.21 *The matrix $(\mathbf{Q}_\ell, \mathbf{Q}'_\ell)^H \mathbf{A} (\mathbf{Q}_\ell, \mathbf{Q}'_\ell)$ is similar to the matrix $\mathbf{A}$. Hence, its eigenvalues are the same as those of $\mathbf{A}$. Theorem 7.20 states that the eigenvalues of the block $\mathbf{A}_{11}$ are close to the k largest eigenvalues of $\mathbf{A}$. Theorem 7.20 also states that the block $\mathbf{A}_{21}$ tends to zero as $\ell \to \infty$. That is, the sequence of matrices $\mathbf{A}_\ell$ tends to block diagonal form.*

7.5 Basic QR-algorithm

A first way to understand the classical QR-algorithm (without refinements such as shift strategies) is to view it as the orthogonal iteration with starting matrix $\mathbf{X}_0 = \mathrm{I} \in \mathbb{R}^{n\times n}$:

Algorithm 7.22 (orthogonal iteration with $\mathbf{X}_0 = \mathrm{I}$) *% input:* $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{X}_0 := \mathrm{I} \in \mathbb{R}^{n\times n}$
$\ell := 0$
$\mathbf{X}_0 =: \mathbf{Q}_0 \mathbf{R}_0$, *where $\mathbf{Q}_0 \in \mathbb{R}^{n\times n}$ has orthogonal columns, $\mathbf{R} \in \mathbb{R}^{n\times n}$ upper triangular.*
repeat {

$\mathbf{X}_{\ell+1} := \mathbf{A}\mathbf{Q}_\ell$

$\mathbf{X}_{\ell+1} =: \mathbf{Q}_{\ell+1}\mathbf{R}_{\ell+1}$ *% QR-decomposition of* $\mathbf{X}_{\ell+1}$: $\mathbf{Q}_{\ell+1} \in \mathbb{R}^{n\times n}$ *has orthogonal columns* $\mathbf{R}_{\ell+1} \in \mathbb{R}^{n\times n}$ *is upper triangular*

$\ell := \ell + 1$

} **until** *sufficiently accurate*

Remark 7.23 *Algorithm 7.22 actually performs n orthogonal iterations simultaneously. That is, for each* $k \in \{1, \dots, n\}$*, the first k columns of* $\mathbf{Q}_\ell$ *are those that would be created by the orthogonal iteration Alg. 7.18 started with* $\mathbf{X}_0 = [\mathbf{e}_1, \dots, \mathbf{e}_k]$*. To see this, we compute with* $\mathbf{X}_0 = \mathrm{I}$

$$\mathbf{A}^\ell \mathrm{I} = \mathbf{A}^\ell \mathbf{X}_0 = \mathbf{A}^{\ell-1}\mathbf{A}\mathbf{X}_0 = \mathbf{A}^{\ell-1}\mathbf{Q}_1\mathbf{R}_1 = \mathbf{A}^{\ell-2}\mathbf{A}\mathbf{Q}_1\mathbf{R}_1 = \mathbf{A}^{\ell-2}\mathbf{Q}_2\mathbf{R}_2\mathbf{R}_1 = \cdots = \mathbf{Q}_\ell\mathbf{R}_\ell\cdots\mathbf{R}_1$$

Since the product $\mathbf{R}_\ell \cdots \mathbf{R}_1$ *is upper triangular as a product of upper triangular matrices, we see that the columns of* $\mathbf{A}^\ell[\mathbf{e}_1, \dots, \mathbf{e}_k]$ *are linear combinations of the first k columns of* $\mathbf{Q}_\ell$*. Hence, for invertible* $\mathbf{A}$*, the first k columns of* $\mathbf{Q}_\ell$ *form an ONB of the space* $\mathbf{A}^\ell\mathcal{S}^0$*, where* $\mathcal{S}^0$ *is the space spanned by* $\mathbf{X}_0 = [\mathbf{e}_1, \dots, \mathbf{e}_k]$*. See also Remark 7.19.*

Since Alg. 7.22 performs n simultanenous orthogonal iterations (by Remark 7.23) Theorem 7.20 suggests that the sequence of matrices

$$\mathbf{A}_\ell := \mathbf{Q}_\ell^H \mathbf{A}\mathbf{Q}_\ell$$

converges to upper triangular form. Indeed, if $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ (and the technical conditions $\operatorname{span}\{\mathbf{e}_1, \dots, \mathbf{e}_k\} \cap \operatorname{span}\{\mathbf{v}_{k+1}, \dots, \mathbf{v}_n\} = \{0\}$ for every $k \in \{1, \dots, n\}$) then Theorem 7.20 asserts that each block $\mathbf{A}_\ell([1:k],[k+1:n])$ of $\mathbf{A}_\ell$ tend to zero. Since the matrices $\mathbf{A}_\ell$ are similar to $\mathbf{A}$, the eigenvalues of $\mathbf{A}_\ell$ and $\mathbf{A}$ coincide. Thus, the diagonal entries of the matrices $\mathbf{A}_\ell$ converge to the eigenvalues of $\mathbf{A}$.
The basic QR-algorithm creates the matrices $\mathbf{A}_\ell$ in a more efficient way than computing $\mathbf{Q}_\ell^H\mathbf{A}\mathbf{Q}$ directly. One makes the following observations:

$$\begin{aligned}
&\mathbf{X}_{\ell+1} = \mathbf{A}\mathbf{Q}_\ell = \mathbf{Q}_{\ell+1}\mathbf{R}_{\ell+1},\\
&\mathbf{A}_\ell = \mathbf{Q}_\ell^H\mathbf{A}\mathbf{Q}_\ell = \underbrace{\mathbf{Q}_\ell^H\mathbf{Q}_{\ell+1}}_{=:\widehat{\mathbf{Q}}_{\ell+1}}\mathbf{R}_{\ell+1} \qquad \text{is "the" } QR\text{-decomposition of } \mathbf{A}_\ell,\\
&\mathbf{A}_{\ell+1} = \mathbf{Q}_{\ell+1}^H\mathbf{A}\mathbf{Q}_{\ell+1} = (\mathbf{Q}_\ell\widehat{\mathbf{Q}}_{\ell+1})^H\mathbf{A}\mathbf{Q}_{\ell+1} = \widehat{\mathbf{Q}}_{\ell+1}^H\mathbf{Q}_\ell^H\mathbf{A}\mathbf{Q}_\ell\widehat{\mathbf{Q}}_{\ell+1} = \widehat{\mathbf{Q}}_{\ell+1}^H\mathbf{A}_\ell\widehat{\mathbf{Q}}_{\ell+1} = \mathbf{R}_{\ell+1}\widehat{\mathbf{Q}}_{\ell+1}.
\end{aligned}$$

(The QR-decomposition is indeed unique for invertible matrices if one additionally fixes the sign of the diagonal entries of the R-factor—cf. Theorem 4.44.) We conclude that $\mathbf{A}_{\ell+1}$ is obtained from $\mathbf{A}_\ell$ by computing "the" QR-factorization of $\mathbf{A}_\ell$ and then multiplying the factors in reverse order. This is the classical QR-algorithm:

Algorithm 7.24 (basic form of classical QR-algorithm without shift and deflation)
% input: $\mathbf{A} \in \mathbb{R}^{n\times n}$
$\ell := 0$; $\mathbf{A}_0 := \mathbf{A}$
repeat {

$\mathbf{A}_\ell =: \mathbf{Q}_\ell \mathbf{R}_\ell$ *% QR-decomposition of* $\mathbf{A}_\ell$

$\mathbf{A}_{\ell+1} := \mathbf{R}_\ell \mathbf{Q}_\ell$

$\ell := \ell + 1$

} **until** *sufficiently accurate*

Remark 7.25 *Computationally, Alg. 7.24 is still too expense as each QR-decomposition costs* $O(n^3)$*. In practice,* $\mathbf{A}$ *is brought to Hessenberg form (with cost* $O(n^3)$*) and then each QR-decomposition is only* $O(n^2)$*, see Example 4.53. This is comptuationally essential: assuming that* $O(n)$ *QR-steps are needed to compute the* n *eigenvalues, the total cost are then* $O(n^3) + O(n)O(n^2) = O(n^3)$*. If, instead, cost* $O(n^3)$ *are incurred for each QR-step, then one expects the total cost to be* $O(n)O(n^3) = O(n^4)$*.*

Remark 7.26 *In practice, the QR-algorithm is combined with the Rayleighquotient iteration idea, i.e., with suitable shifts. This improves the convergence of the algorithm.*

7.6 Jacobi method(CSE)

7.6.1 Schur representation

goal: eigenvalue-revealing representation of $\mathbf{A}$
Diagonalizable matrices $\mathbf{A}$ can be written as $\mathbf{A} = \mathbf{T}\mathbf{D}\mathbf{T}^{-1}$ with the diagonal matrix $\mathbf{D}$. This is an eigenvalue-revealing representation. However, if $\text{cond}(\mathbf{T})$ is large, then this representation is numerically not advisable. In this case, an alternative is the Schur form

Theorem 7.27 (Schur form) *Let $\mathbf{A} \in \mathbb{C}^{n\times n}$. Then there is a unitary[2] matrix $\mathbf{Q} \in \mathbb{C}^{n\times n}$ and an upper triangular matrix $\mathbf{R}$ with $\mathbf{A} = \mathbf{Q}\mathbf{R}\mathbf{Q}^H$. The diagonal entries of $\mathbf{R}$ are the eigenvalues (according to multiplicity) of $\mathbf{A}$.*

Proof: We prove the theorem by induction on n. For $n = 1$ the theorem is obviously true. Suppose it is true for all matrices in $\mathbb{R}^{(n-1)\times(n-1)}$. Let $\mathbf{v} \in \mathbb{R}^n$ be an eigenvector of $\mathbf{A}$, i.e., $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$. Let the columns of $\mathbf{V}' \in \mathbb{C}^{n\times(n-1)}$ be such that $\mathbf{V} := (\mathbf{v}, \mathbf{V}')$ is unitary (i.e., the columns of $\mathbf{V}'$ are an ONB of the orthogonal complement of $\text{span}\{\mathbf{v}\}$).

$$\mathbf{V}^H\mathbf{A}\mathbf{V} = \begin{pmatrix} \lambda & \mathbf{w}^\top \\ 0 & \mathbf{C} \end{pmatrix}, \qquad \mathbf{w} \in \mathbb{R}^{n-1}, \quad \mathbf{C} \in \mathbb{R}^{(n-1)\times(n-1)}.$$

By the induction hypothesis, there is a unitary $\mathbf{Q} \in \mathbb{R}^{(n-1)\times(n-1)}$ such that $\mathbf{Q}^H\mathbf{C}\mathbf{Q} = \mathbf{R}'$ is upper triangular. Then

$$\begin{pmatrix} 1 & 0 \\ 0 & \mathbf{Q} \end{pmatrix}^H \mathbf{V}^H\mathbf{A}\mathbf{V} \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{Q} \end{pmatrix}$$

is upper triangular. Thus we have obtained the desired Schur decomposition for $\mathbf{A} \in \mathbb{C}^{n\times n}$. □

Remark 7.28 *In* MATLAB*, the Schur form of a matrix can be computed with* `schur`*.*

7.6.2 Jacobi method

The QR-method for eigenvalue computations is based on the idea of finding a sequence of orthogonal matrices $\mathbf{Q}_n$ such that the $\mathbf{Q}_n^\top\mathbf{A}\mathbf{Q}_n$ converge to upper triangular form. Since these are similarity transformations of $\mathbf{A}$, the diagonal entries of the upper triangular matrix contains the eigenvalues. If the entries in the lower part are small, then these diagonal entries are indeed good approximations to the eigenvalues:

Exercise 7.29 *Consider a matrix $\mathbf{R} + \Delta\mathbf{A}$ where $\mathbf{R}$ is upper triangular. Show, using Theorem 7.14 that for each $\lambda \in \sigma(\mathbf{R} + \Delta\mathbf{A})$ there is a diagonal entry $\mathbf{R}_{ii}$*

$$|\lambda - \mathbf{R}_{ii}| \le C\|\Delta\mathbf{A}\|_2,$$

where the constant C depends on $\mathbf{R}$ but is independent of $\Delta\mathbf{A}$.

[2] i.e., $\mathbf{Q}^H\mathbf{Q} = \mathrm{I}$

A simpler form than the QR-method is Jacobi's method, which constructs the $\mathbf{Q}_n$ by Givens rotations. Recall the definition of Givens rotations $\mathbf{G}(i,j,\theta)$ of Section 4.6.4. We also introduce for a matrix $\mathbf{A}$

$$\mathrm{off}(\mathbf{A})^2 := \sum_{\substack{i,j \\ i \neq j}} |\mathbf{A}_{ij}|^2 = \|\mathbf{A}\|_F^2 - \sum_{i=1}^{n} |\mathbf{A}_{ii}|^2. \tag{7.4}$$

We consider *symmetric* matrices $\mathbf{A}$. The basic step of the Jacobi eigenvalue procedure consists of three steps:

1. select a pair (i,j) with $1 \leq i < j \leq n$
2. select θ such that (we write again $c = \cos\theta$, $s = \sin\theta$)

$$\begin{pmatrix} \mathbf{B}_{ii} & \mathbf{B}_{ij} \\ \mathbf{B}_{ji} & \mathbf{B}_{jj} \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^\top \begin{pmatrix} \mathbf{A}_{ii} & \mathbf{A}_{ij} \\ \mathbf{A}_{ji} & \mathbf{A}_{jj} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \tag{7.5}$$

   is diagonal
3. overwrite $\mathbf{A}$ with $\mathbf{B} = \mathbf{G}(i,j,\theta)^\top \mathbf{A} \mathbf{G}(i,j,\theta)$

In other words: one makes a similarity transformation of $\mathbf{A}$ with a Givens rotation in such a way that the entries (i,j) and (j,i) of $\mathbf{A}$ are annihilated. We now show that the transformed matrix has a smaller off-diagonal part:

Lemma 7.30 *Let $\mathbf{A}$ be symmetric. Let $\mathbf{B} := \mathbf{G}(i,j,\theta)^\top \mathbf{A} \mathbf{G}(i,j,\theta)$, where θ is chosen such that $\mathbf{B}_{ij} = \mathbf{B}_{ji} = 0$. Then*

$$\mathrm{off}(\mathbf{B})^2 = \mathrm{off}(\mathbf{A})^2 - 2\mathbf{A}_{ij}^2.$$

Proof: We consider the transformation (7.5). Since the Frobenius norm is invariant under orthogonal transformations, we have

$$\mathbf{A}_{ii}^2 + \mathbf{A}_{jj}^2 + 2\mathbf{A}_{ij}^2 = \mathbf{B}_{ii}^2 + \mathbf{B}_{jj}^2 + 2\mathbf{B}_{ij}^2 = \mathbf{B}_{ii}^2 + \mathbf{B}_{jj}^2. \tag{7.6}$$

Hence, we get

$$\begin{aligned}
\mathrm{off}(\mathbf{B})^2 &= \|\mathbf{B}\|_F^2 - \sum_{k=1}^{n} |\mathbf{B}_{kk}|^2 \\
&= \|\mathbf{A}\|_F^2 - \sum_{k \notin \{i,j\}} |\mathbf{B}_{kk}|^2 - |\mathbf{B}_{ii}|^2 - |\mathbf{B}_{jj}|^2 \\
&\overset{\text{only rows/colums } i,\, j \text{ are touched}}{=} \|\mathbf{A}\|_F^2 - \sum_{k \notin \{i,j\}} |\mathbf{A}_{kk}|^2 - |\mathbf{B}_{ii}|^2 - |\mathbf{B}_{jj}|^2 \\
&= \|\mathbf{A}\|_F^2 - \sum_{k=1}^{n} |\mathbf{A}_{kk}|^2 + |\mathbf{A}_{ii}|^2 + |\mathbf{A}_{jj}|^2 - |\mathbf{B}_{ii}|^2 - |\mathbf{B}_{jj}|^2 \\
&\overset{(7.6)}{=} \mathrm{off}(\mathbf{A})^2 - 2|\mathbf{A}_{ij}|^2.
\end{aligned}$$

□

Lemma 7.30 suggests that one should select the pair (i, j) such that $|\mathbf{A}_{ij}|$ is as large as possible. That is, taking the largest possible off-diagonal entry yields, with $N = n(n-1)/2$

$$\operatorname{off}(\mathbf{A})^2 \leq N(\mathbf{A}_{ij}^2 + \mathbf{A}_{ji}^2) \tag{7.7}$$

and therefore

$$\operatorname{off}(\mathbf{B})^2 \overset{\text{Lemma 7.30}}{=\!=} \operatorname{off}(\mathbf{A})^2 - \left(|\mathbf{A}_{ij}|^2 + |\mathbf{A}_{ji}|^2\right) \overset{(7.7)}{\leq} \left(1 - \frac{1}{N}\right) \operatorname{off}(\mathbf{A})^2.$$

Thus, the Jacobi method converges to upper triangular form[3]

Remark 7.31 *1. Searching the largest off-diagonal entry incurs large costs. Practically, one therefore simply loops through the off-diagonal entries of* $\mathbf{A}$. $\rightarrow$ *"cyclic Jacobi" method.*

2. The convergence is linear. However, the asymptotic convergence is actually quadratic, i.e., for the k-th matrix $\mathbf{A}^{(k)}$ *one has* $\operatorname{off}(\mathbf{A}^{(k+N)})^2 \leq C \operatorname{off}(\mathbf{A}^{(k)})^4$.

3. The Jacobi method is not competitive with the QR-algorithm in general. However, if $\mathbf{A}$ *is already close to diagonal, then it is an option.*

4. Variants exist that produce the SVD of $\mathbf{A}$.

7.7 QR-algorithm with using Hessenberg form (CSE)

Computationally, each QR-factorization in the basic QR-algorithm (Algorithm 7.24) incurs cost $O(n^3)$. The situation changes if $\mathbf{A}$ has Hessenberg form[4]. As discussed in Example 4.53 it is possible to compute the QR-factorization of a Hessenberg matrix with cost $O(n^2)$ (using Givens rotations). Moreover, the multiplication RQ is also achieved with cost $O(n^2)$ and the resulting matrix RQ has again upper Hessenberg form. In conclusion, it is computationally advantageous to bring a matrix $\mathbf{A}$ to Hessenberg form prior to applying the QR-algorithm (or the variants that we will describe below) to it.

7.8 Deflation (CSE)

Suppose that the matrix $\mathbf{A}$ has the form

$$\mathbf{A} = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ 0 & 0 & 0 & \mu \end{pmatrix},$$

i.e., $\mathbf{A}(n,:) = (0, 0, \ldots, 0, \mu)$. Then μ is an eigenvalue of $\mathbf{A}$ and the remaining eigenvalues of $\mathbf{A}$ are the eigenvalues of the matrix $\mathbf{A}([1 : n-1], [1 : n-1])$. This can obviously be exploited algorithmically. In practice, two things are done:

[3] the matrix $\mathbf{A}$ is symmetric and the multiplications with the Givens rotations produce symmetric matrices. Since symmetric upper triangular matrices are in fact diagonal, the Jacobi method applied to symmetric matrices converges to diagonal form.

[4] upper triangular and one subdiagonal is allowed to be nonzero, i.e., $\mathbf{A}_{ij} = 0$ for $j > i + 1$

1. if the "off-diagonal" part of $\mathbf{A}(n,:)$ is small, i.e., if $\|\mathbf{A}(n,[1:n-1])\|$ is small (compared to $\varepsilon\|\mathbf{A}(n,:)\|$ for some ε close to machine precision), then $\mathbf{A}(n,n)$ will be a good approximation to a true eigenvalue of $\mathbf{A}$ (cf. Theorem 7.14). In the context of QR-iterations, one will therefore monitor the size of the $\mathbf{A}_\ell(n,[1:n-1])$, identify $\mathbf{A}_\ell(n,n)$ as an eigenvalue if possible and continue the search for eigenvalues with $\mathbf{A}_\ell([1:n-1],[1:n-1])$. Computational savings are achieved because of the reduction in size the matrix.[5]

2. The QR-algorithm with shift is designed such that deflation happens quickly: the shift parameters (later!) are chosen cleverly in such a way that $\|\mathbf{A}_\ell(n,[1:n-1])\|$ converges quickly (in fact, quadratically) to zero. Hence, quickly, one eigenvalue is identified and the iteration can be continued with a matrix whose size is reduced by 1.

7.9 QR-algorithm with shift (CSE)

goal: convergence acceleration of QR-algorithm using shifts.
mathematical background: Implicitly the QR-algorithm with shift performs an inverse iteration for $\mathbf{A}^H$ so that choosing Rayleigh quotients as shift leads to rapid convergence.
So far, we assumed $\mathbf{A}$ to be real (although this is by no means essential). Since we want to allow complex shifts, we allow $\mathbf{A}$ to be complex. We note that the concept of QR-factorizations also holds for complex matrices.[6]
A generalization of the basic QR-algorithm is the QR-algorithm with shift:

Algorithm 7.32 (QR-Algorithm with shift) $\ell := 0 \qquad \mathbf{A}_0 := \mathbf{A}$
repeat {
- *choose shift* $\mu^{(\ell)}$
- $\mathbf{A}_\ell - \mu^{(\ell)} := \mathbf{Q}_{\ell+1}\mathbf{R}_{\ell+1}$
- $\mathbf{A}_{\ell+1} := \mathbf{R}_{\ell+1}\mathbf{Q}_{\ell+1} + \mu^{(\ell)}$

} *until* $\mathbf{A}_\ell$ *is sufficiently close to upper triangular form*

Exercise 7.33 *Check that* $\mathbf{A}_\ell$ *and* $\mathbf{A}_{\ell+1}$ *are similar and hence have the same eigenvalues.*

Lemma 7.34 *Let the shifts* $\mu^{(\ell)}$ *be such that* $\mu^{(\ell)} \notin \sigma(\mathbf{A})\ \forall \ell$.

orthogonal iteration with shift	*QR-iteration with shift*
$\widehat{\mathbf{Q}}_0 := I$	$\mathbf{A}_0 := \mathbf{A}$
$\left(\mathbf{A}-\mu^{(\ell)}\right)\widehat{\mathbf{Q}}_\ell =: \widehat{\mathbf{Q}}_{\ell+1}\mathbf{R}_{\ell+1}$	$\mathbf{A}_\ell - \mu^{(l)} =: \mathbf{Q}_{\ell+1}\mathbf{R}_{\ell+1}$
	$\mathbf{A}_{\ell+1} := \mathbf{R}_{\ell+1}\mathbf{Q}_{\ell+1} + \mu^{(\ell)}$

Then: $\forall \ell$:

(i) $\left(\mathbf{A}-\mu^{(\ell)}\right)\left(\mathbf{A}-\mu^{(\ell-1)}\right)\ldots\left(\mathbf{A}-\mu^{(0)}\right)\widehat{\mathbf{Q}}_0 = \widehat{\mathbf{Q}}_{\ell+1}\widehat{\mathbf{R}}_{\ell+1}$ with $\widehat{\mathbf{R}}_{\ell+1} = \mathbf{R}_{\ell+1}\ldots\mathbf{R}_1$

(ii) $\mathbf{A}_\ell = \widehat{\mathbf{Q}}_\ell^H \mathbf{A}\widehat{\mathbf{Q}}_\ell$

[5] Significant savings arise in practice, because in the course of the iteration, one repeatedly reduces the size
[6] in fact, the eigenvalue algorithms are probably better understood by viewing $\mathbf{A} \in \mathbb{C}^{n\times n}$ and specializing to real matrices if necessary

(iii) $\widehat{\mathbf{Q}}_\ell = \mathbf{Q}_1 \cdots \mathbf{Q}_\ell$

Proof: Exercise. Define the matrices $\mathbf{Q}_\ell$ and $\mathbf{R}_\ell$ by the QR-iteration with shift, i.e., by $\mathbf{A}_\ell - \mu^{(\ell)} = \mathbf{Q}_\ell \mathbf{R}_\ell$. *Define* the matrices $\widehat{\mathbf{Q}}_\ell := \mathbf{Q}_1 \cdots \mathbf{Q}_\ell$ and $\widehat{\mathbf{R}}_\ell := \mathbf{R}_\ell \cdots \mathbf{R}_1$. Then (iii) is satisfied by definition. To see (ii) we compute

$$\mathbf{A}_{\ell+1} = \mathbf{R}_{\ell+1}\mathbf{Q}_{\ell+1} + \mu^{(\ell+1)} = \mu^{(\ell+1)} + \mathbf{Q}_{\ell+1}^H(\mathbf{A}_\ell - \mu^{(\ell+1)})\mathbf{Q}_{\ell+1} = \mathbf{Q}_{\ell+1}^H \mathbf{A}_\ell \mathbf{Q}_{\ell+1}.$$

Hence, an induction argument will show (ii).
We now show that matrices $\widehat{\mathbf{Q}}_\ell$ defined above satisfy (i). To that end, we compute

$$\widehat{\mathbf{Q}}_{\ell+1}\widehat{\mathbf{R}}_{\ell+1} = \widehat{\mathbf{Q}}_\ell \mathbf{Q}_{\ell+1}\mathbf{R}_{\ell+1}\widehat{\mathbf{R}}_\ell = \mathbf{Q}_\ell(\mathbf{A}_\ell - \mu^{(\ell)})\widehat{\mathbf{R}}_\ell \overset{(ii)}{=} \widehat{\mathbf{Q}}_\ell(\widehat{\mathbf{Q}}_\ell^H \mathbf{A}\widehat{\mathbf{Q}}_\ell - \mu^{(\ell)})\widehat{\mathbf{R}}_\ell = (\mathbf{A} - \mu^{(\ell)})\widehat{\mathbf{Q}}_\ell\widehat{\mathbf{R}}_\ell \tag{7.8}$$

Hence, an induction argument shows (i).
It remains to see that the $\widehat{\mathbf{Q}}_\ell$ actually satisfy

$$(\mathbf{A} - \mu^{(\ell)})\widehat{\mathbf{Q}}_\ell = \mathbf{Q}_{\ell+1}\mathbf{R}_{\ell+1}.$$

Since the $\mathbf{R}_i$ are invertible, this follows from (7.8) by multiplying both sides with $\widehat{\mathbf{R}}_\ell^{-1}$, which gives

$$\widehat{\mathbf{Q}}_{\ell+1}\mathbf{R}_{\ell+1} = \widehat{\mathbf{Q}}_{\ell+1}\widehat{\mathbf{R}}_{\ell+1}\widehat{\mathbf{R}}_\ell^{-1} \overset{(7.8)}{=} (\mathbf{A} - \mu^{(\ell)})\widehat{\mathbf{Q}}_\ell\widehat{\mathbf{R}}_\ell\widehat{\mathbf{R}}_\ell^{-1} = (\mathbf{A} - \mu^{(\ell)})\widehat{\mathbf{Q}}_\ell.$$

□

We have observed in Remark 7.23 that the orthogonal iteration (with $\mathbf{X}_0 = I$) performs several orthogonal iterations simultaneously. That is, the first k columns of $\widehat{\mathbf{Q}}_\ell$ are an ONB of the space $\mathbf{A}^\ell[\mathbf{e}_1, \ldots, \mathbf{e}_k]$. More generally, Lemma 7.34 shows that the first k columns of $\widehat{\mathbf{Q}}_\ell$ are an ONB of $(\mathbf{A} - \mu^{(\ell)}) \cdots (\mathbf{A} - \mu^{(0)})[\mathbf{e}_1, \ldots, \mathbf{e}_k]$.
The following Lema 7.35 shows that in the case without shift that $(\mathbf{A}^H)^{-\ell}\mathbf{e}_n$ is a multiple of the last column of $\widehat{\mathbf{Q}}_\ell$:

Lemma 7.35 *Let $\mathbf{A} \in \mathbb{C}^{n\times n}$ be invertible. Define the permutation matrix*

$$\mathbf{P} = \begin{pmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{pmatrix}$$

$[\![\ \mathbf{BP} = \mathbf{B}(:, [n : -1 : 1]); \mathbf{P}^H\mathbf{B} = \mathbf{PB} = \mathbf{B}([n : -1 : 1], :)\ ]\!]$
Let $\mathbf{A}^\ell = \widehat{\mathbf{Q}}_\ell\widehat{\mathbf{R}}_\ell$ with $\widehat{\mathbf{Q}}_\ell$ unitary and $\widehat{\mathbf{R}}_\ell$ upper triangular. Then:

$$(\mathbf{A}^H)^{-\ell}\mathbf{e}_n = (\mathbf{A}^H)^{-\ell}\mathbf{P}\mathbf{e}_1 = \underbrace{\widehat{\mathbf{Q}}_\ell(\mathbf{P}\underbrace{(\mathbf{P}^H\widehat{\mathbf{R}}_\ell^{-H}\mathbf{P})\mathbf{e}_1}_{\|\mathbf{e}_1})}_{\|\mathbf{e}_n} = \text{ multiple of last column of } \widehat{\mathbf{Q}}_\ell$$

Proof: direct calculation. □

Lemma 7.35 shows that the *last* columns of the matrices $\mathbf{Q}_\ell$ correspond to an inverse iteration for $\mathbf{A}^H$. More generally, one can show for the case with shifts:

Lemma 7.36

$$(\mathbf{A}^H - \overline{\mu^{(\ell)}})^{-1} \cdots (\mathbf{A}^H - \overline{\mu^{(0)}})^{-1}\mathbf{e}_n = \text{multiple of } \widehat{\mathbf{Q}}_\ell(:, n).$$

with $\widehat{\mathbf{Q}}_\ell$ given by Lemma 7.34.

Proof: Computation/literature. □

Exercise 7.37 *Show that if μ is an eigenvalue of $\mathbf{A}$, then $\overline{\mu}$ is an eigenvalue of $\mathbf{A}^H$.*

Lemma 7.36 shows the last column of $\widehat{\mathbf{Q}}_\ell$ corresponds to an inverse iteration for $\mathbf{A}^H$ with shifts related to the shifts of the QR-iteration. Hence it is sensible to select the shifts $\mu^{(\ell)}$ of the QR-iteration such that $\overline{\mu^{(\ell)}}$ is the Rayleigh quotient for $\mathbf{q}_n := \widehat{\mathbf{Q}}_\ell(:, n)$:

$$\overline{\mu^{(\ell)}} := \frac{\mathbf{q}_n^H \mathbf{A}^H \mathbf{q}_n}{\|\mathbf{q}_n\|_2^2} = \mathbf{q}_n^H \mathbf{A}^H \mathbf{q}_n = (\widehat{\mathbf{Q}}_\ell \mathbf{e}_n)^H \mathbf{A}^H (\widehat{\mathbf{Q}}_\ell \mathbf{e}_n).$$

Hence,

$$\mu^{(\ell)} := \left((\widehat{\mathbf{Q}}_\ell \mathbf{e}_n)^H \mathbf{A}^H (\widehat{\mathbf{Q}}_\ell \mathbf{e}_n) \right)^H = \mathbf{e}_n^H \widehat{\mathbf{Q}}_\ell^H \mathbf{A} \widehat{\mathbf{Q}}_\ell \mathbf{e}_n = \mathbf{e}_n^H \mathbf{A}_{\ell+1} \mathbf{e}_n = \mathbf{A}_{\ell+1}(n, n).$$

That is, the shift should be taken as the bottom lower entry $\mathbf{A}_{\ell+1}(n, n)$ of $\mathbf{A}_{\ell+1}$. We have arrive at the classical QR-algorithm with (single) shift:

Algorithm 7.38 (classical QR-Algorithm with (Rayleigh) shift and Hessenberg form)
$\ell := 0, \qquad \mathbf{A}_0 := \texttt{hessenberg}(\mathbf{A})$
repeat {
- *choose shift* $\mu^{(\ell)} := \mathbf{A}_\ell(n, n)$
- $\mathbf{A}_\ell - \mu^{(\ell)} := \mathbf{Q}_{\ell+1} \mathbf{R}_{\ell+1}$
- $\mathbf{A}_{\ell+1} := \mathbf{R}_{\ell+1} \mathbf{Q}_{\ell+1} + \mu^{(\ell)}$

} *until* $\mathbf{A}_\ell$ *is sufficiently close to upper triangular form*

A few comments are in order:

1. The general behavior of the QR-algorithm with (Rayleigh) shift is that one has rapid convergence (quadratic convergence!) towards one eigenvalue since it behaves like a Rayleigh quotient method. Furthermore, one has linear convergence towards the remaining eigenvalues.

2. The rapid convergence towards one eigenvalue make deflation possible → iterate on a smaller matrix!

3. For *deflation*, monitor $\mathbf{A}_\ell(n-1, n)$: Since one will perform the QR-algorithm for $\mathbf{A}_0$ in Hessenberg form (so that all $\mathbf{A}_\ell$ have Hessenberg form — cf. Remark 7.25) $\mathbf{A}_\ell$ is Hessenberg, and it has only two non-zero entries in the nth row, namely, $\mathbf{A}_\ell(n, n-1)$ and $\mathbf{A}_\ell(n, n)$. Hence, deflation can be done when $\mathbf{A}_\ell(n, n-1)$ is sufficiently small (e.g., a small multiple of machine precision). That is, if $\mathbf{A}_\ell(n-1, n)$ is deemed sufficiently small, the entry $\mathbf{A}_\ell(n, n)$ is recognized as an eigenvalue and the search for further eigenvalues is done by applying the QR-method to the $(n-1) \times (n-1)$ submatrix $\mathbf{A}(1 : n-1, 1 : n-1)$. This reduction has two positive effects: a) one reduces the size of the matrix one operators one (i.e., reduction in computational cost) and b) the shift strategy (leading to quadratric convergence!) focuses on the next eigenvalue.

Given the importance of the potential of deflation, we reformulate Algorithm 7.38 to include deflation

Algorithm 7.39 (classical QR-Algorithm with (Rayleigh) shift, Hessenberg form, and deflation
% `signature:` $[ev] = \texttt{qr}(\mathbf{A})$
% *input: matrix* $\mathbf{A}$ in Hessenberg form (i.e., call $\texttt{hess}(\mathbf{A})$ prior to calling `qr`)
% *output: list of eigenvalues ev*
$\ell := 0$
`repeat` {
- *choose shift* $\mu^{(\ell)} := \mathbf{A}_\ell(n,n)$
- $\mathbf{A}_\ell - \mu^{(\ell)} := \mathbf{Q}_{\ell+1}\mathbf{R}_{\ell+1}$ % *QR-decomposition of* $\mathbf{A}_\ell - \mu^{(\ell)}$
- $\mathbf{A}_{\ell+1} := \mathbf{R}_{\ell+1}\mathbf{Q}_{\ell+1} + \mu^{(\ell)}$

} `until` $|\mathbf{A}_\ell(n-1,n)|$ *is sufficiently small*
$[ev'] = \texttt{qr}(\mathbf{A}_\ell(1:n-1,1:n-1))$ % *recursive call with submatrix* $\mathbf{A}_\ell(1:n-1,1:n-1)$
`return` $[ev', \mathbf{A}_\ell(n,n)]$ % *return* $\mathbf{A}_\ell(n,n)$ *and the eigenvalues of* $\mathbf{A}_\ell(1:n-1,1:n-1)$

7.9.1 further comments on QR

Problem: in particular, for real matrices with eigenvalues appearing in complex conjugate pairs, it is possible for the Rayleigh quotient method to fail: $\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Then the QR-iteration (with shift) yields $\mathbf{A}_\ell = \mathbf{A} \forall \ell$.
solution:(Wilkinson-shift):
consider the two eigenvalues λ_1, λ_2 of $\mathbf{A}(n-1:n, n-1:n)$ and choose the shift as the eigenvalue that is closer to $\mathbf{A}(n,n)$.
Problem: QR-algorithm does not converge with Wilkinson shift:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad \sigma(\mathbf{A}) = \{1, \frac{1}{2}(-1+\sqrt{3}\mathbf{i}), \frac{1}{2}(-1-\sqrt{3}\mathbf{i})\}$$

here, the (Wilkinson) shift is 0 and all eigenvalues have absolute value 1. Indeed, $\mathbf{A}_\ell = \mathbf{A}$ for all ℓ.
solution: If the QR-iteration does not converge, then make a "random shift". In general, this leads to a separation (in absolute value) of the eigenvalues and thus convergence: If $\lambda_3 \neq \lambda_1 \neq \lambda_2 \neq \lambda_3$, but $|\lambda_1| = |\lambda_2| = |\lambda_3|$, then $|\lambda_2 - \lambda| \neq |\lambda_1 - \lambda| \neq |\lambda_2 - \lambda| \neq |\lambda_3 - \lambda|$.

7.9.2 real matrices

Suppose $\mathbf{A}$ is real and one is not interested in complex shifts (e.g., because one wishes to stay with real arithmetic). In this case, eigenvalues appear in complex conjugate pairs λ, $\overline{\lambda}$. One can therefore makes two QR-steps with shifts λ and $\overline{\lambda}$. It is possible to combine these two steps purely in real arithmetic.

8 Conjugate Gradient method (CG)

Goal: iterative solution of $\mathbf{A}\mathbf{x}^* = \mathbf{b},\ \mathbf{A} \in \mathbb{R}^{N\times N}$ symmetric positive definite
"rules": employ solely the matrix-vector multiplication $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$

Remark 8.1 *In many applications* $\mathbf{A}$ *can be very large but sparse, i.e.,* $\mathbf{A}$ *has only few non-zero entries per row. Then, the matrix-vector multiplication is feasible but a factorization of* $\mathbf{A}$ *may be infeasible (Cholesky factors of* $\mathbf{A}$ *typically need much more memory than* $\mathbf{A}$*).*

We will employ two scalar products:

- $(\mathbf{x}, \mathbf{y})_2 := \mathbf{x}^T\mathbf{y} = \sum_i \mathbf{x}_i\mathbf{y}_i$ ("euklidean scalar product")
- $(\mathbf{x}, \mathbf{y})_\mathbf{A} := \mathbf{x}^T\mathbf{A}\mathbf{y}$ ("energy scalar product")

Exercise 8.2 $(\cdot,\cdot)_\mathbf{A}$ *is a scalar product and* $\|\mathbf{x}\|_\mathbf{A} := \sqrt{(\mathbf{x},\mathbf{x})_\mathbf{A}}$ *is a norm on* $\mathbb{R}^N$.

Notation:

- $\mathbf{x}_0 \in \mathbb{R}^N$ arbitrary (=initial value)
- $\mathbf{x}^*$ solution of $\mathbf{A}\mathbf{x}^* = \mathbf{b}$
- $\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$ = initial residual
- $\mathbf{e}_0 := \mathbf{x}^* - \mathbf{x}_0$ = initial error
- define, for each $\ell \in \mathbb{N}$ the *Krylov space*

$$\mathcal{K}_\ell := \mathcal{K}_\ell(\mathbf{A},\ \mathbf{r}_0) = \operatorname{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^{\ell-1}\mathbf{r}_0\}$$

We have the *residual equation*

$$\mathbf{A}\mathbf{e}_0 = \mathbf{r}_0 \tag{8.1}$$

Question: Can one approximate $\mathbf{e}_0$ well from the spaces $\mathcal{K}_\ell$ (for "small" ℓ)? Consider the *best approximation*

$$\text{find } \tilde{\mathbf{e}}_\ell \in \mathcal{K}_\ell,\ \text{s.t. } \|\mathbf{e}_0 - \tilde{\mathbf{e}}_\ell\|_\mathbf{A} \le \|\mathbf{e}_0 - \mathbf{x}\|_\mathbf{A} \qquad \forall \mathbf{x} \in \mathcal{K}_\ell \tag{8.2}$$

Correspondingly, one obtains an approximation $\mathbf{x}_\ell := \mathbf{x}_0 + \tilde{\mathbf{e}}_\ell$ of the original problem. Since $\mathbf{e}_0 = \mathbf{x}^* - \mathbf{x}_0$, we may characterize $\mathbf{x}_\ell$ also as:

$$\text{find } \mathbf{x}_\ell \in \mathbf{x}_0 + \mathcal{K}_\ell \text{ s.t. } \|\mathbf{x}^* - \mathbf{x}_\ell\|_\mathbf{A} \le \|\mathbf{x}^* - \mathbf{x}\|_\mathbf{A} \qquad \forall \mathbf{x} \in \mathbf{x}_0 + \mathcal{K}_\ell \tag{8.3}$$

The solution $\mathbf{x}_\ell$ of (8.3) can also be characterized as follows:

Lemma 8.3 *The following are equivalent for* $\mathbf{x}_\ell \in \mathbf{x}_0 + \mathcal{K}_\ell$*:*

(i) $\mathbf{x}_\ell$ *solves (8.3)*

(ii) $(\mathbf{x}^* - \mathbf{x}_\ell, \mathbf{v})_\mathbf{A} = 0 \quad \forall \mathbf{v} \in \mathcal{K}_\ell$

(iii) $(\mathbf{r}_\ell, \mathbf{v})_2 = 0 \quad \forall \mathbf{v} \in \mathcal{K}_\ell$, *where* $\mathbf{r}_\ell := \mathbf{b} - \mathbf{A}\mathbf{x}_\ell$

Proof:
(ii) $\Leftrightarrow$ (iii): $(\mathbf{x}^* - \mathbf{x}_\ell, \mathbf{v})_\mathbf{A} = 0 \ \forall \mathbf{v} \in \mathcal{K}_\ell \Leftrightarrow (\mathbf{A}(\mathbf{x}^* - \mathbf{x}_\ell), \mathbf{v})_2 = 0 \ \forall \mathbf{v} \in \mathcal{K}_\ell \Leftrightarrow (\mathbf{b} - \mathbf{A}\mathbf{x}_\ell, \mathbf{v})_2 = 0 \ \forall \mathbf{v} \in \mathcal{K}_\ell$
(i) $\Longrightarrow$ (ii): Define for arbitrary $\mathbf{v} \in \mathcal{K}_\ell$ the function $\pi : \mathbb{R} \to \mathbb{R}$ by $\pi(t) := \|\mathbf{x}^* - \mathbf{x}_\ell + t\mathbf{v}\|_\mathbf{A}^2 = \|\mathbf{x}^* - \mathbf{x}_\ell\|_\mathbf{A}^2 + 2t(\mathbf{x}^* - \mathbf{x}_\ell, \mathbf{v})_\mathbf{A} + t^2|\mathbf{v}|_\mathbf{A}^2$
By assumption, π has a minimum at $t = 0 \Rightarrow 0 = \pi'(0) = (\mathbf{x}^* - \mathbf{x}_\ell, \mathbf{v})_\mathbf{A}$.
Since $\mathbf{v} \in \mathcal{K}_l$ is arbitrary, the claim follows.
(ii) $\Longrightarrow$ (i): Let $\mathbf{x}_\ell \in \mathbf{x}_0 + \mathcal{K}_\ell$ be such that $(\mathbf{x}^* - \mathbf{x}_\ell, \mathbf{v})_\mathbf{A} = 0 \ \forall \mathbf{v} \in \mathcal{K}_\ell$
Hence, the function π defined above has, for each fixed $\mathbf{v}$, its minimum at $t = 0 \Longrightarrow \pi(0) \leq \pi(t) \ \forall t \in \mathbb{R}$. Since $\mathbf{v}$ is arbitrary, (i) follows. □

For small ℓ, the $\ell \times \ell$ linear system of equations corresponding to (8.2) (or, alternatively, (8.3)) could be set up and solved (exercise!). However, the CG-algorithm proceeds in a much more economical way that determines $\mathbf{x}_\ell$ as a cheap update of $\mathbf{x}_{\ell-1}$.
$\mathbf{x}_\ell - \mathbf{x}_0 \in \mathcal{K}_\ell$ implies

$$\mathbf{r}_\ell = \mathbf{b} - \mathbf{A}\mathbf{x}_\ell = \mathbf{b} - \mathbf{A}\mathbf{x}_0 - \mathbf{A}(\mathbf{x}_\ell - \mathbf{x}_0) = \underbrace{\mathbf{r}_0}_{\in \mathcal{K}_0 \subset \mathcal{K}_{\ell+1}} \underbrace{- \mathbf{A}(\underbrace{\mathbf{x}_\ell - \mathbf{x}_0}_{\in \mathcal{K}_\ell})}_{\in \mathcal{K}_{\ell+1}} \in \mathcal{K}_{\ell+1},$$

i.e., $\mathbf{r}_\ell \in \mathcal{K}_{\ell+1}$. Since $\mathbf{r}_\ell$ is orthogonal to $\mathcal{K}_\ell$ (cf. Lemma 8.3,(iii)), we obtain inductively that[1]

$$\mathcal{K}_{\ell+1} = \operatorname{span}\{\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_\ell\}.$$

We now focus on the algorithmic construction of the approximations $\mathbf{x}_\ell$. To that end, it is convenient to determine vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots$, such that $\{\mathbf{d}_0, \ldots, \mathbf{d}_\ell\}$ is an orthogonal basis (w.r.t. the $(\cdot,\cdot)_\mathbf{A}$-scalar product) of $\mathcal{K}_{\ell+1}$. This is achieved with Gram-Schmidt orthogonalization: In view of $\mathcal{K}_\ell = \operatorname{span}\{\mathbf{r}_0, \ldots, \mathbf{r}_{\ell-1}\} = \operatorname{span}\{\mathbf{d}_0, \ldots, \mathbf{d}_{\ell-1}\}$ and $\mathcal{K}_{\ell+1} = \operatorname{span}\{\mathbf{r}_0, \ldots, \mathbf{r}_\ell\}$, we have that $\mathbf{d}_\ell$ has the form

$$\mathbf{d}_\ell = \mathbf{r}_\ell - \sum_{i=0}^{\ell-1} \beta_i \mathbf{d}_i$$

for suitable β_i. The orthogonality conditions

$$(\mathbf{d}_\ell, \mathbf{d}_i)_\mathbf{A} = 0 \text{ for } 0 \leq i \leq \ell - 1$$

produce

$$\beta_i = \frac{(\mathbf{r}_\ell, \mathbf{d}_i)_\mathbf{A}}{\|\mathbf{d}_i\|_\mathbf{A}^2}, \qquad i = 0, \ldots, \ell - 1.$$

For $i \leq \ell - 2$ we have

$$\beta_i = \frac{(\mathbf{r}_\ell, \mathbf{d}_i)_\mathbf{A}}{\|\mathbf{d}_i\|_\mathbf{A}^2} = \frac{(\mathbf{r}_\ell, \overbrace{\mathbf{A}\mathbf{d}_i}^{\in \mathbf{A}\mathcal{K}_{i+1} \subset \mathcal{K}_{i+2} \subset \mathcal{K}_\ell = \operatorname{span}\{\mathbf{r}_0, \ldots, \mathbf{r}_{\ell-1}\}})_2}{\|\mathbf{d}_i\|_\mathbf{A}^2} \overset{\text{Lemma 8.3,(iii)}}{=} 0.$$

Therefore,

$$\mathbf{d}_\ell = \mathbf{r}_\ell - \beta_{\ell-1}\mathbf{d}_{\ell-1}, \quad \beta_{\ell-1} = \frac{(\mathbf{r}_\ell, \mathbf{d}_{\ell-1})_\mathbf{A}}{\|\mathbf{d}_{\ell-1}\|_\mathbf{A}^2}. \tag{8.4}$$

[1] in fact, Lemma 8.3, (iii) shows that $\{\mathbf{r}_0, \ldots, \mathbf{r}_\ell\}$ is an orthogonal basis of $\mathcal{K}_{\ell+1}$ (unless one of the $\mathbf{r}_i$ is zero).

Next, we derive recursions for the $\mathbf{x}_\ell$ and $\mathbf{r}_\ell$: Since $\mathbf{x}_\ell - \mathbf{x}_{\ell-1} = (\mathbf{x}_\ell - \mathbf{x}_0) - (\mathbf{x}_0 - \mathbf{x}_{\ell-1}) \in \mathcal{K}_\ell$ and the orthogonality of Lemma 8.3,(ii) implies that $\mathbf{x}_\ell - \mathbf{x}_{\ell-1} = (\mathbf{x}_\ell - \mathbf{x}^*) - (\mathbf{x}^* - \mathbf{x}_{\ell-1})$ is $(\cdot,\cdot)_\mathbf{A}$-orthogonal to $\mathcal{K}_{\ell-1}$ we conclude

$$\mathbf{x}_\ell - \mathbf{x}_{\ell-1} = \alpha_\ell \mathbf{d}_{\ell-1}$$

for some $\alpha_\ell \in \mathbb{R}$. To derive an equation for the unknown α_ℓ we note that applying $\mathbf{A}$ to this equation yields

$$\alpha_\ell \mathbf{A}\mathbf{d}_{\ell-1} = \mathbf{A}(\mathbf{x}_\ell - \mathbf{x}_{\ell-1}) = \mathbf{A}\mathbf{x}_\ell - \mathbf{b} - (\mathbf{A}\mathbf{x}_{\ell-1} - \mathbf{b}) = -\mathbf{r}_\ell + \mathbf{r}_{\ell-1}$$

so that

$$\alpha_\ell (\mathbf{A}\mathbf{d}_{\ell-1}, \mathbf{r}_{\ell-1})_2 = (-\mathbf{r}_\ell + \mathbf{r}_{\ell-1}, \mathbf{r}_{\ell-1})_2 \overset{\text{Lemma 8.3, (iii)}}{=} \|\mathbf{r}_{\ell-1}\|_2^2. \tag{8.5}$$

We have thus obtained:

(α) $\mathbf{d}_\ell = \mathbf{r}_\ell - \beta_{\ell-1}\mathbf{d}_{\ell-1}$, $\quad\beta_{\ell-1}$ given by (8.4)

(β) $\mathbf{r}_\ell = \mathbf{r}_{\ell-1} - \alpha_\ell \mathbf{A}\mathbf{d}_{\ell-1}$, $\quad\alpha_\ell$ given by (8.5).

(γ) $\mathbf{x}_\ell = \mathbf{x}_{\ell-1} + \alpha_\ell \mathbf{d}_{\ell-1}$

Remark 8.4 *Computationally, is it better to compute α_ℓ, β_ℓ as follows:*

$$\alpha_\ell = \frac{\|\mathbf{r}_{\ell-1}\|_2^2}{(\mathbf{d}_{\ell-1}, \mathbf{r}_{\ell-1})_\mathbf{A}} = \frac{\|\mathbf{r}_{\ell-1}\|_2^2}{(\mathbf{d}_{\ell-1}, \mathbf{d}_{\ell-1} + \beta_{\ell-2}\mathbf{d}_{\ell-2})_\mathbf{A}} = \frac{\|\mathbf{r}_{\ell-1}\|_2^2}{\|\mathbf{d}_{\ell-1}\|_\mathbf{A}^2}$$

$$\beta_{\ell-1} = \frac{(\mathbf{r}_\ell, \mathbf{d}_{\ell-1})_\mathbf{A}}{\|\mathbf{d}_{\ell-1}\|_\mathbf{A}^2} = \frac{(\mathbf{r}_\ell, \mathbf{A}\mathbf{d}_{\ell-1})_2}{\|\mathbf{d}_{\ell-1}\|_\mathbf{A}^2} = -\frac{(\mathbf{r}_\ell, \frac{\mathbf{r}_\ell - \mathbf{r}_{\ell-1}}{\alpha_\ell})_2}{\|\mathbf{d}_{\ell-1}\|_\mathbf{A}^2} = \frac{-\|\mathbf{r}_\ell\|_2^2}{\alpha_\ell \|\mathbf{d}_{\ell-1}\|_\mathbf{A}^2} = -\frac{\|\mathbf{r}_\ell\|_2^2}{\|\mathbf{r}_{\ell-1}\|_2^2}$$

We have thus derived the following algorithm:

Algorithm 8.5 (CG) *% input: SPD matrix* $\mathbf{A}$, $\mathbf{b} \in \mathbb{R}^N$, *initial vector* $\mathbf{x}_0$
% output: (approx.) solution $\mathbf{x}_n \approx \mathbf{A}^{-1}\mathbf{b}$

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\mathbf{d}_0 := \mathbf{r}_0$
for $\ell = 1, \ldots,$ *until stopping criterion is satisfied* **do** {
  $\alpha_\ell := \frac{\|\mathbf{r}_{\ell-1}\|_2^2}{\|\mathbf{d}_{\ell-1}\|_A^2}$
  $\mathbf{r}_\ell := \mathbf{r}_{\ell-1} - \alpha_\ell \mathbf{A}\mathbf{d}_{\ell-1}$
  $\mathbf{x}_\ell := \mathbf{x}_{\ell-1} + \alpha_\ell \mathbf{d}_{\ell-1}$
  $\beta_{\ell-1} := -\frac{\|\mathbf{r}_\ell\|_2^2}{\|\mathbf{r}_{\ell-1}\|_2^2}$
  $\mathbf{d}_\ell := \mathbf{r}_\ell - \beta_{\ell-1}\mathbf{d}_{\ell-1}$
}

slide 22 - CG method

Remark 8.6
- *CG ist very economical w.r.t. memory requirements: merely 4 vectors of length N have to be kept in memory concurrently ($\mathbf{x}_\ell$, $\mathbf{r}_\ell$, $\mathbf{d}_\ell$, $\mathbf{A}\mathbf{d}_\ell$).*
- *In exact arithmetic, CG terminates with the exact solution after at most N steps. Technically, one may view CG therefore as a direct solver. Round-off problems, however, stop the method from realizing the exact solution after N steps.*

8.1 convergence behavior of CG

- $\mathbf{A} \in \mathbb{R}^{N\times N}$ SPD $\Rightarrow \exists$ ONB $\{\xi_1, \ldots, \xi_N\}$ of $\mathbb{R}^N$ consisting of eigenvectors of $\mathbf{A}$ with corresponding eigenvalues $\lambda_i,\ i = 1, \ldots, N$
- $\mathbf{x} = \sum_{i=1}^N \mathbf{x}_i\xi_i \Rightarrow \|\mathbf{x}\|_{\mathbf{A}}^2 = (\mathbf{x}, \mathbf{A}\mathbf{x})_2 = \sum_{i,j}(\mathbf{x}_i\xi_i, \lambda_j\mathbf{x}_j\xi_j) = \sum_{i,j}\mathbf{x}_i^2\lambda_j\delta_{ij} = \sum_{i=1}^N \mathbf{x}_i^2\lambda_i$
- $p \in \mathcal{P}_m \wedge \mathbf{x} = \sum_i \mathbf{x}_i\xi_i \Rightarrow p(\mathbf{A})\mathbf{x} = \sum_i p(\lambda_i)\xi_i\mathbf{x}_i$:
  Write $p(\mathbf{A}) = \sum_{j=0}^m p_j\mathbf{A}^j \Longrightarrow p(\mathbf{A})\mathbf{x} = \sum_j p_j\mathbf{A}^j \sum_i \mathbf{x}_i\xi_i = \sum_{i,j} p_j\mathbf{x}_i\lambda_i^j\xi_i = \sum_i \mathbf{x}_i\xi_i p(\lambda_i)$
- $p \in \mathcal{P}_m$ and $\mathbf{x} = \sum \mathbf{x}_i\xi_i \Longrightarrow \|p(\mathbf{A})\mathbf{x}\|_{\mathbf{A}}^2 = \sum \mathbf{x}_i^2|p(\lambda_i)|^2\lambda_i$:

$$\begin{aligned}\|p(\mathbf{A})\mathbf{x}\|_{\mathbf{A}}^2 &= (p(\mathbf{A})\mathbf{x}, \mathbf{A}p(\mathbf{A})\mathbf{x})_2 = \sum_{i,j}(\mathbf{x}_i\xi_i p(\lambda_i), \mathbf{x}_j\xi_j\lambda_j p(\lambda_j))_2 \\ &= \sum_{i,j}\mathbf{x}_i\mathbf{x}_j p(\lambda_i)p(\lambda_j)\lambda_j \underbrace{(\xi_i, \xi_j)_2}_{\delta_{i,j}} = \sum_i |\mathbf{x}_i|^2\lambda_i|p(\lambda_i)|^2\end{aligned}$$

In view of $\mathcal{K}_\ell = \operatorname{span}\{\mathbf{r}_0, \ldots, \mathbf{A}^{\ell-1}\mathbf{r}_0\} = \{q(\mathbf{A})\mathbf{r}_0 \,|\, q \in \mathcal{P}_{\ell-1}\}$ and $\mathbf{r}_0 = \mathbf{A}\mathbf{e}_0$:

$$\begin{aligned}\|\mathbf{x}^* - \mathbf{x}_\ell\|_{\mathbf{A}} &= \min_{\mathbf{x}\in\mathbf{x}_0+\mathcal{K}_\ell}\|\mathbf{x}^* - \mathbf{x}\|_{\mathbf{A}} = \min_{\mathbf{z}\in\mathcal{K}_\ell}\|\mathbf{e}_0 - \mathbf{z}\|_{\mathbf{A}} = \min_{\mathbf{z}\in\mathcal{K}_\ell=\operatorname{span}\{\mathbf{r}_0,\ldots,\mathbf{A}^{\ell-1}\mathbf{r}_0\}}\|\mathbf{e}_0 - \mathbf{z}\|_{\mathbf{A}} \\ &= \min_{q\in\mathcal{P}_{\ell-1}}\|\mathbf{e}_0 - q(\mathbf{A})\mathbf{r}_0\|_{\mathbf{A}} = \min_{q\in\mathcal{P}_\ell:\, q(0)=0}\|\mathbf{e}_0 - q(\mathbf{A})\mathbf{e}_0\|_{\mathbf{A}} = \min_{q\in\mathcal{P}_\ell:\, q(0)=1}\|q(\mathbf{A})\mathbf{e}_0\|_{\mathbf{A}}.\end{aligned}$$

Therefore:

Theorem 8.7 *The iterates $\mathbf{x}_\ell$ of the CG method satisfy*

$$\|\mathbf{x}^* - \mathbf{x}_\ell\|_{\mathbf{A}} = \min_{q\in\mathcal{P}_\ell:\, q(0)=1}\|q(\mathbf{A})\mathbf{e}_0\|_{\mathbf{A}}$$

We estimate further with $\mathbf{e}_0 = \sum \mathbf{x}_i\xi_i$:

$$\|q(\mathbf{A})\mathbf{e}_0\|_{\mathbf{A}}^2 \le \sum_i \mathbf{x}_i^2\lambda_i q^2(\lambda_i) \le \max_{\lambda\in\sigma(\mathbf{A})} q^2(\lambda)\sum_i \mathbf{x}_i^2\lambda_i = \max_{\lambda\in\sigma(\mathbf{A})} q^2(\lambda)\|\mathbf{e}_0\|_{\mathbf{A}}^2$$

Hence:

$$\|\mathbf{x}^* - \mathbf{x}_\ell\|_{\mathbf{A}} \le \min_{q\in\mathcal{P}_\ell:\, q(0)=1}\max_{\lambda\in\sigma(\mathbf{A})}|q(\lambda)|\ \|\mathbf{e}_0\|_{\mathbf{A}}$$

Theorem 8.8 *Let $\mathbf{A} \in \mathbb{R}^{N\times N}$ be SPD , $0 < \lambda_{min}(\mathbf{A}) \le \lambda_{max}(\mathbf{A})$, $\kappa := \operatorname{cond}_2(\mathbf{A}) = \frac{\lambda_{max}(\mathbf{A})}{\lambda_{min}(\mathbf{A})}$. Then: The iterates of the CG method satisfy*

$$\|\mathbf{x}^* - \mathbf{x}_\ell\|_{\mathbf{A}} \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^\ell\|\mathbf{e}_0\|_{\mathbf{A}}$$

Proof: We have

$$\|\mathbf{x}^* - \mathbf{x}_\ell\|_{\mathbf{A}} \leq \min_{q=\mathcal{P}_\ell,\, q(0)=1} \underbrace{\max_{\lambda\in\sigma(\mathbf{A})} |q(\lambda)|}_{\leq \max_{\lambda\in[\lambda_{min},\,\lambda_{max}]} |q(\lambda)|} \|\mathbf{e}_0\|_{\mathbf{A}}$$

We select a specific q:

$$q(x) := \frac{T_\ell(\frac{a+b-2x}{b-a})}{T_\ell(\frac{b+a}{b-a})} \qquad a = \lambda_{min},\ b = \lambda_{max}$$

$T_\ell(x)$ = Chebyshev polynomial = $\frac{1}{2}\left[(x+\sqrt{x^2-1})^\ell + (x-\sqrt{x^2-1})^\ell\right]$ and uses

$$\max_{x\in[a,b]} \left| T_\ell\left(\frac{a+b-2x}{b-a}\right)\right| = \max_{x\in[-1,1]} |T_\ell(x)| = 1.$$

□

Remark 8.9 *Theorem 8.8 shows that the condition number of* $\mathbf{A}$ *is very important for the convergence behavior of the CG method. For matrices* $\mathbf{A}$ *with large condition number, one will therefore apply the CG not to* $\mathbf{A}$ *directly but to* $\mathbf{B}^{-1}\mathbf{A}$ *where the SPD matrix* $\mathbf{B}$ *is SPD. For more, see literature on the so-called "preconditioned CG" (PCG).*

8.2 GMRES (CSE)

goal: iterative methods for non-symmetric matrices $\mathbf{A} \in \mathbb{R}^{N\times N}$.
technique: for the Krylov space $\mathcal{K}_\ell := \text{span}\{\mathbf{r}_0, \ldots, \mathbf{A}^{\ell-1}\mathbf{r}_0\}$ GMRES seek $\mathbf{x}_\ell \in \mathbf{x}_0 + \mathcal{K}_\ell$ such that

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}_\ell\|_2 \leq \|\mathbf{b} - \mathbf{A}x\|_2 \qquad \forall \mathbf{x} \in \mathbf{x}_0 + \mathcal{K}_\ell \tag{8.6}$$

The minimization property (8.6) implies an orthogonality condition:

Exercise 8.10 *Show that the residual $\mathbf{r}_\ell := \mathbf{b} - \mathbf{A}\mathbf{x}_\ell$ satisfies*

$$(\mathbf{b} - \mathbf{A}x_\ell, \mathbf{v})_2 = (\mathbf{r}_\ell, \mathbf{v})_2 = 0 \qquad \forall \mathbf{v} \in \mathbf{A}\mathcal{K}_\ell. \tag{8.7}$$

Hint: *Proceed as in the proof of Lemma 8.3 or in the derivation of the normal equations in Least Squares. (Note: GMRES can effectively be understood as a Least Squares method!)*

Remark 8.11 *The form (8.7) of GMRES suggests generalizations of GMRES: given a second space $\mathcal{L}_\ell$ one could consider: Find $\mathbf{x}_\ell \in \mathbf{x}_0 + \mathcal{K}_\ell$ such that*

$$(\mathbf{b} - \mathbf{A}x_\ell, \mathbf{v})_2 = (\mathbf{r}_\ell, \mathbf{v})_2 = 0 \qquad \forall \mathbf{v} \in \mathcal{L}_\ell. \tag{8.8}$$

Different choices of $\mathcal{L}_\ell$ lead to different method. The choice $\mathcal{L}_\ell = \mathcal{K}_\ell$ leads (for SPD matrices) to the CG-method (cf. Lemma 8.3), the choice $\mathcal{L}_\ell = \mathbf{A}\mathcal{K}_\ell$ to the classical GMRES.

One can show (this is not complicated, see literature) that for invertible matrices $\mathbf{A}$ GMRES finds (in exact arithmetic) the exact solution in N steps. As with the CG method, the importance lies in the fact that in practice good approximations are obtained $\ell \ll N$.

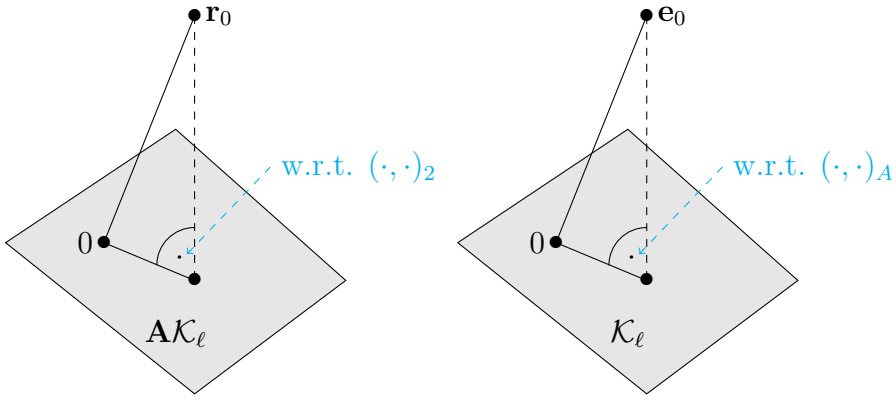


Figure 8.1: The orthogonality conditions (8.7) and Lemma 8.3, (ii).

Computing the $\mathbf{x}_\ell$

As in the CG method, one computes the approximations $\mathbf{x}_\ell$ successively until one is found that is sufficiently accurate. It is, of course, essential that the $\mathbf{x}_\ell$ be computed efficiently from the orthogonality conditions (8.7). The general procedure is:

- Construct $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_\ell]$ an $N \times \ell$ matrix the columns of which form a basis for the space $\mathcal{K}_\ell$. It will be computationally convenient to choose the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_\ell$ orthogonal.
- Construct $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_\ell]$ an $n \times \ell$ matrix the columns of which form a basis for the space $\mathcal{L}_\ell = \mathbf{A}\mathcal{K}_\ell$.
- Write the approximate solution as

  $$\mathbf{x}_\ell = \mathbf{x}_0 + \mathbf{V}\mathbf{y},$$

  where $\mathbf{y} \in \mathbb{R}^\ell$ is the vector of weights to be determined.
- Enforcing the orthogonality conditions (8.7) the system of equations

  $$\mathbf{W}^T\mathbf{A}\mathbf{V}\mathbf{y} = \mathbf{W}^T\mathbf{r}_0, \tag{8.9}$$

  from which the approximate solution $\mathbf{x}_\ell$ can be written as

  $$\mathbf{x}_\ell = \mathbf{x}_0 + \mathbf{V}(\mathbf{W}^T\mathbf{A}\mathbf{V})^{-1}\mathbf{W}^T\mathbf{r}_0. \tag{8.10}$$

  We note that the matrix $\mathbf{W}^T\mathbf{A}\mathbf{V}$ is only of the order $\ell \times \ell$; therefore its inversion is affordable (for $\ell \ll N$). In exact arithmetic the choice of the basis of $\mathcal{K}_\ell$ (i.e., the choice of $\mathbf{V}$) is immaterial and the vectors $\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^{\ell-1}\mathbf{r}_0\}$ could be used. However, then the corresponding matrix $\mathbf{V}$ is rather ill-conditioned (recall: the vectors $\mathbf{A}^j\mathbf{r}_0$ are scaled versions of the vectors of the power method, and they converge to the dominant eigenvector!) so that one expects numerical difficulties when solving (8.9). In practice, therefore, some orthogonalization as discussed next is advised.

8.2.1 realization of the GMRES method

GMRES computes the vectors $\mathbf{v}_1, \ldots,$ successively such that the $\{\mathbf{v}_1, \ldots, \mathbf{v}_\ell\}$ is a basis of $\mathcal{K}_\ell$. Then the linear system described by (8.10) is solved in an efficient way.
We note (exercise!) that $\mathcal{K}_{\ell+1} = \mathbf{A}\mathcal{K}_\ell \supset \mathcal{K}_\ell$ for all ℓ. In the following, we will make the assumption that the inclusion is strict: $\mathcal{K}_\ell \subsetneqq \mathcal{K}_{\ell+1}$ for all ℓ of interest. That is, $\dim \mathcal{K}_\ell = \ell + 1$. One can show (see literature) that the case $\mathcal{K}_\ell = \mathcal{K}_{\ell+1}$ is a fortuitous case as then $\mathbf{x}_\ell = \mathbf{x}^*$ ("lucky breakdown").
The first step of the GMRES algorithm is to generate a vectors $\mathbf{v}_1, \ldots,$. Since we want the vectors $\mathbf{v}_j$, $j = 1, \ldots, \ell$, to be orthogonal, we will construct them using a variant of the Gram-Schmidt orthogonalization procedure given in Alg. 8.12 (in practice, a variant, the so-called "modified Gram-Schmidt" procedure, is used that is numerically more stable—see lines 5–8 of Alg. 8.14).

Algorithm 8.12 (Arnoldi, standard Gram-Schmidt variant)
% input $\mathbf{r}_0$*;*
%output: ONB of $\mathcal{K}_\ell = \mathrm{span}\{\mathbf{r}_0, \ldots, \mathbf{A}^{\ell-1}\mathbf{r}_0\}$

1: $\mathbf{v}_1 = \mathbf{r}_0/\|\mathbf{r}_0\|_2$
2: ***for*** $j = 1, 2, \ldots, \ell$ ***do***
3: ***for*** $i = 1, 2, \ldots, j$ ***do***

4: $\quad h_{ij} = (\mathbf{A}\mathbf{v}_j, \mathbf{v}_i)_2$
5: ***end for***
6: $\mathbf{w}_j = \mathbf{A}\mathbf{v}_j - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i$
7: $h_{j+1,j} = \|\mathbf{w}_j\|_2$
8: $\mathbf{v}_{j+1} = \mathbf{w}_j / h_{j+1,j}$
9: ***end for***

The algorithm generates the $(\ell+1) \times \ell$ Hessenberg matrix

$$\bar{\mathbf{H}}_\ell = \begin{pmatrix} (\mathbf{A}\mathbf{v}_1, \mathbf{v}_1) & (\mathbf{A}\mathbf{v}_2, \mathbf{v}_1) & (\mathbf{A}\mathbf{v}_3, \mathbf{v}_1) & \dots & (\mathbf{A}\mathbf{v}_\ell, \mathbf{v}_1) \\ (\mathbf{A}\mathbf{v}_1, \mathbf{v}_2) & (\mathbf{A}\mathbf{v}_2, \mathbf{v}_2) & (\mathbf{A}\mathbf{v}_3, \mathbf{v}_2) & & \\ & (\mathbf{A}\mathbf{v}_2, \mathbf{v}_3) & (\mathbf{A}\mathbf{v}_3, \mathbf{v}_3) & & \\ & & (\mathbf{A}\mathbf{v}_3, \mathbf{v}_4) & \ddots & \\ & & & \ddots & (\mathbf{A}\mathbf{v}_\ell, \mathbf{v}_\ell) \\ & & & & (\mathbf{A}\mathbf{v}_\ell, \mathbf{v}_{\ell+1}) \end{pmatrix} \in \mathbb{R}^{(\ell+1)\times\ell}$$

together with the orthonormal vectors $\mathbf{v}_i = \frac{\mathbf{w}_{i-1}}{\|\mathbf{w}_{i-1}\|_2}$ that are produced by the Gram-Schmidt orthogonalization procedure:

$$\begin{aligned} \mathbf{w}_1 &= \mathbf{A}\mathbf{v}_1 - (\mathbf{A}\mathbf{v}_1, \mathbf{v}_1)\mathbf{v}_1 \\ \mathbf{w}_2 &= \mathbf{A}\mathbf{v}_2 - (\mathbf{A}\mathbf{v}_2, \mathbf{v}_1)\mathbf{v}_1 - (\mathbf{A}\mathbf{v}_2, \mathbf{v}_2)\mathbf{v}_2 \\ &\vdots \end{aligned}$$

as well as (note $\mathbf{v}_{\ell+1} = \mathbf{w}_\ell / \|\mathbf{w}_\ell\|_2$ implies $(\mathbf{w}_\ell, \mathbf{v}_{\ell+1})_2 = \|\mathbf{w}_\ell\|_2$)

$$\|\mathbf{w}_\ell\|_2 = (\mathbf{w}_\ell, \mathbf{v}_{\ell+1})_2 = (\mathbf{A}\mathbf{v}_\ell, \mathbf{v}_{\ell+1})_2 = h_{\ell+1,\ell}.$$

Exercise 8.13 *Assuming that Alg. 8.12 doesn't terminate prematurely, the vectors $\mathbf{v}_j$, $j = 1, \dots, \ell$, form an orthonormal basis of the Krylov space $\mathcal{K}_\ell$.*

We set $\mathbf{V}_\ell := (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_\ell) \in \mathbb{R}^{N\times\ell}$. Since the vectors $\mathbf{v}_j$, $j = 1, \dots, \ell$, are orthonormal and since $\mathbf{A}\mathbf{v}_j \in \operatorname{span}\{\mathbf{v}_1, \dots, \mathbf{v}_{j+1}\}$ and thus $\mathbf{A}\mathbf{v}_j = \sum_{i=1}^{j+1} (\mathbf{A}\mathbf{v}_j, \mathbf{v}_i)\mathbf{v}_i$ we get

$$\begin{aligned} \mathbf{A}\mathbf{V}_\ell &= [\mathbf{A}\mathbf{v}_1 \quad \dots \quad \mathbf{A}\mathbf{v}_\ell] \\ &= \left[(\mathbf{A}\mathbf{v}_1, \mathbf{v}_1)\mathbf{v}_1 + (\mathbf{A}\mathbf{v}_1, \mathbf{v}_2)\mathbf{v}_2 \quad \dots \quad \sum_{i=1}^{\ell+1} (\mathbf{A}\mathbf{v}_\ell, \mathbf{v}_i)\mathbf{v}_i \right] \\ &= \mathbf{V}_{\ell+1}\bar{\mathbf{H}}_\ell. \end{aligned} \tag{8.11}$$

Additionally,

$$\mathbf{V}_\ell^\top \mathbf{A}\mathbf{V}_\ell = \mathbf{V}_\ell^\top \mathbf{V}_{\ell+1}\bar{\mathbf{H}}_\ell = [\, I \,|\, 0 \,]\bar{\mathbf{H}}_\ell = \mathbf{H}_\ell \tag{8.12}$$

where $\mathbf{H}_\ell$ is the square matrix obtained by removing the last row of $\bar{\mathbf{H}}_\ell$.
We abbreviate

$$\beta := \|\mathbf{r}_0\|_2$$

and note that $\beta\mathbf{v}_1 = \mathbf{r}_0$. Additionally, we observe $\beta\mathbf{V}_{\ell+1}\mathbf{e}_1 = \beta\mathbf{v}_1 = \mathbf{r}_0$, where $\mathbf{e}_1 = (1,0,0,\ldots,0)^\top \in \mathbb{R}^{\ell+1}$.
GMRES minimizes the residuum (cf. (8.6)). Hence, seeking $\mathbf{x}_\ell$ in the form $\mathbf{x}_\ell = \mathbf{x}_0 + \mathbf{V}_\ell\mathbf{y}$ we can write

$$\begin{aligned}\mathbf{b} - \mathbf{A}\mathbf{x}_\ell &= \mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \mathbf{V}_\ell\mathbf{y}) = \mathbf{r}_0 - \mathbf{A}\mathbf{V}_\ell\mathbf{y} = \beta\mathbf{v}_1 - \mathbf{V}_{\ell+1}\bar{\mathbf{H}}_\ell\mathbf{y} \\ &= \mathbf{V}_{\ell+1}(\beta\mathbf{e}_1 - \bar{\mathbf{H}}_\ell\mathbf{y});\end{aligned}$$

exploiting the fact that the columns of $\mathbf{V}_{\ell+1}$ are orthonormal, we can determine the vector $\mathbf{y}$ by (8.6), i.e., $\mathbf{y}$ is the minimizer of

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}_\ell\|_2 = \min_{\mathbf{x}\in\mathbf{x}_0+\mathcal{K}_\ell} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2 = \min_{\mathbf{y}\in\mathbb{R}^\ell} \|\beta\mathbf{e}_1 - \bar{\mathbf{H}}_\ell\mathbf{y}\|_2. \tag{8.13}$$

One way to solve for $\mathbf{y}$ is to set up and solve the *normal equations* using the Cholesky factorization with cost $O(\ell^3)$. However, since $\bar{\mathbf{H}}$ has Hessenberg form, its QR-factorization can be computed with $O(\ell^2)$ using, e.g., Givens rotations. The pseudo-code for the GMRES-algorithm can now be given as Algorithm 8.14.

Algorithm 8.14 (GMRES (basic form)) *% input:* $\mathbf{x}_0$*, number of steps* ℓ

1: *Compute* $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$*,* $\beta = \|\mathbf{r}_0\|_2$*, and* $\mathbf{v}_1 = \mathbf{r}_0/\beta$
2: *Define the* $(\ell+1)\times\ell$ *matrix* $\bar{\mathbf{H}}_\ell$ *and set its elements* h_{ij} *to zero*
3: ***for*** $j = 1, 2, \ldots, \ell$ ***do***
4: $\quad\mathbf{w}_j = \mathbf{A}\mathbf{v}_j$
5: $\quad$***for*** $i = 1, \ldots, j$ ***do***
6: $\quad\quad h_{ij} = (\mathbf{w}_j, \mathbf{v}_i)_2$
7: $\quad\quad\mathbf{w}_j = \mathbf{w}_j - h_{ij}\mathbf{v}_i$
8: $\quad$***end for***
9: $\quad h_{j+1,j} = \|\mathbf{w}_j\|_2$.
10: $\quad$**If** $h_{j+1,j} = 0$ **goto** *12* *% lucky break—exact solution found*
11: $\quad\mathbf{v}_{j+1} = \mathbf{w}_j/h_{j+1,j}$
12: ***end for***
13: *Compute* $\mathbf{y}_\ell$ *as the minimizer of* $\|\beta\mathbf{e}_1 - \bar{\mathbf{H}}([1:\ell+1],[1:\ell])\mathbf{y}\|_2^2$ *(e.g., QR-factorization)*
14: $\mathbf{x}_\ell = \mathbf{x}_0 + \mathbf{V}_\ell\mathbf{y}_\ell$

A few comments concerning Alg. 8.14 are:

Remark 8.15

- The derivation of Alg. 8.14 assumed that matrix $\bar{\mathbf{H}}$ has full rank since we assumed that $\dim\mathcal{K}_\ell = \ell + 1$. Alg. 8.14 takes this into account by stopping if $h_{j+1,j} = 0$, which happens if $\mathcal{K}_{j+1} = \mathcal{K}_j$. However, a more careful analysis of the algorithm reveals that if $\bar{\mathbf{H}}$ does not have full rank, i.e., if $\mathcal{K}_j = \mathcal{K}_{j+1}$, then GMRES has actually found the exact solution $\mathbf{x}^*$. This situation is therefore called a "lucky breakdown".

- Solving the minimization problem in line 13 is done by QR-factorization of the Hessenberg matrix $\bar{\mathbf{H}}$, e.g., with Givens rotations.

- The algorithm is implemented differently in practice. The parameter ℓ is not determined *a priori*. Instead, a maximum number ℓ_{max} is given (typically dictated by the computational resources). The vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_\ell$ are computed successively together with the

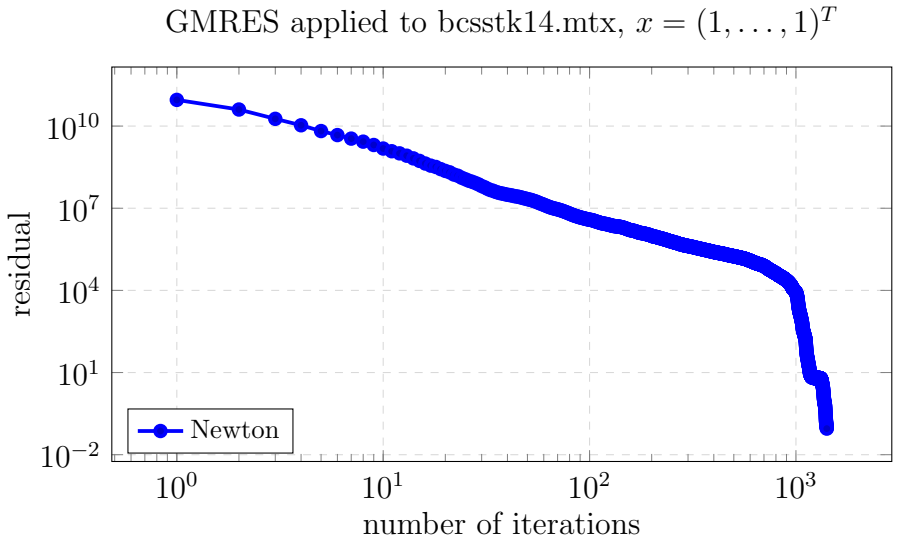


Figure 8.2: Convergence history of GMRES (A is SPD).

matrices $\bar{\mathbf{H}}_\ell$; that is, if the vectors $\mathbf{v}_j$, $1 \leq j \leq \ell - 1$ and the matrix $\bar{\mathbf{H}}_{\ell-1}$ have already been computed, one merely needs to compute $\mathbf{v}_\ell$ and the matrix $\bar{\mathbf{H}}_\ell$ is obtained from $\bar{\mathbf{H}}_{\ell-1}$ by adding one column and the entry $h_{\ell+1,\ell}$. An appropriate termination condition (typically, the size of the residual $\|\mathbf{b} - \mathbf{A}\mathbf{x}_\ell\|_2$) is employed to stop the iteration. If the maximum number of iterations has been reached without triggering the termination condition, then a *restart* is done, i.e., GMRES is started afresh with the last approximation $\mathbf{x}_{\ell_{max}}$ as the initial guess. This is called *restarted GMRES*(ℓ_{max}) in the literature.

slide 22a - GMRES

Remark 8.16 *Faute de mieux*, the residual $\|\mathbf{b}-\mathbf{A}\mathbf{x}_\ell\|_2$ is typically used as a stopping criterion in GMRES. It should be noted that for matrices A with large $\kappa_2(\mathbf{A})$, the error may be large in spite of the residual being small:

$$\frac{\|\mathbf{x} - \mathbf{x}_\ell\|_2}{\|\mathbf{x}\|_2} \leq \kappa_2(\mathbf{A}) \frac{\|\mathbf{b} - \mathbf{A}\mathbf{x}_\ell\|_2}{\|\mathbf{b}\|_2}.$$

Example 8.17 MATLAB has a robust version of restarted GMRES that can be used for experimentation. Applying this version of GMRES to the SPD matrix $\mathbf{A} \in \mathbb{R}^{1806\times 1806}$ `bcsstk14.mtx` of `MatrixMarket` with exact solution $x = (1, 1, \cdots, 1)^\top$ results in the convergence history plotted in Fig. 8.2. We note that the residual decays as the number if iterations increases. If the number of iterations reaches the problem size, the exact solution should be found. As in this example, this doesn't happen in practice due to round-off problems, but the residual is quite small. It should be noted that, generally speaking, GMRES is employed in connection with a suitable preconditioner. We expect this to greatly improve the convergence behavior.

9 Numerical Methods for ODEs

goal: solve, for given $y_0 \in \mathbb{R}$ and function f the initial value problem

$$y'(t) = f(t, y(t)), \qquad y(0) = y_0 \tag{9.1}$$

We will be interested in the solution y in the interval $[0, T]$. The numerical methods will seek approximations $y_i \approx y(t_i)$ in the points

$$0 = t_0 < t_1 < \cdots < t_N = T.$$

We denote by $h_i := t_{i+1} - t_i$ the *step lengths* and by $h := \max_i h_i$ the maximal step length.

9.1 Euler's method

The simplest method is the explicit Euler method. Starting from the known value $y_0 = y(t_0)$ we seek an approximation y_1. By Taylor approximation we observe

$$y(t_1) = y(t_0) + h_0 y'(t_0) + O(h_0^2) \overset{(9.1)}{=} y_0 + h_0 f(t_0, y_0) + O(h^2).$$

Hence, we are led to the approximation

$$y_1 := y_0 + h_0 f(t_0, y_0).$$

Since we assume that y_1 is a good approximation to $y(t_1)$, we may repeat the Taylor argument to obtain $y_2 := y_1 + h_1 f(t_1, y_1)$. This leads to the *explicit Euler method*: define the approximations y_i to the exact values $y(t_i)$ successively by

$$y_{i+1} := y_i + h_i f(t_i, y_i), \qquad i = 0, \ldots, N-1. \tag{9.2}$$

It is not obvious that the final approximation $y(T) - y_N = y(t_N) - y_N$ really is a good one as errors over many steps may accumulate. Indeed, while the Taylor approximation is valid in the first step (y_0 is exact), already in the second step we replace $y(t_1)$ with the approximation y_1 in the Taylor approximation and we have to expect that this additional error is potentially amplified by the recursion (9.2). Nevertheless, under reasonable assumptions the explicit Euler method converges. For the proof, we need the notion of the *consistency error* τ: Let $t \mapsto y(t)$ be the exact solution. The consistency error $\tau_{eE}(t, h)$ of the explicit Euler method at t is defined as

$$\tau_{eE}(t, h) := y(t + h) - [y(t) + h f(t, y(t))] \tag{9.3}$$

We recognize that τ_{eE} measures the error of *one* step of the method when started with the exact value $y(t)$. We note that Taylor's formula gives

$$|\tau_{eE}(t, h)| \leq \frac{1}{2} h^2 \|y''\|_{\infty, [0, T+h]} \tag{9.4}$$

Theorem 9.1 (convergence of explicit Euler) *Let $f \in C^1(\mathbb{R}^2)$ with bounded derivatives, i.e., there is $L > 0$ such that $|\nabla f(t, x)| \leq L$ for all $(t, x) \in \mathbb{R}^2$. Then there exists $C > 0$ such that for the approximation y_i obtained by (9.2)*

$$\max_{i=0,\ldots,N-1} |y(t_i) - y_i| \leq C e^{LT} h.$$

Proof: We define the errors

$$e_i := y(t_i) - y_i$$

and note that $e_0 = 0$. For simplicity of notation, we assume a uniform step length $h_i = h$ (exercise: check that the proof also works for variable step size). We seek a recursion for the errors. To that end, we write

$$\begin{aligned} y_{i+1} &= y_i + hf(t_i, y_i), \\ y(t_{i+1}) &= y(t_i) + hf(t_i, y(t_i)) + \tau(t_i, h). \end{aligned}$$

We obtain

$$e_{i+1} = e_i + h \underbrace{[f(t_i, y(t_i)) - f(t_i, y_i)]}_{(y(t_i)-y_i)\partial_y f(t_i,\xi)} + \tau(t_i, h),$$

where used the intermediate value theorem for some ξ between y_i and $y(t_i)$. We obtain

$$|e_{i+1}| \le |e_i| + h \underbrace{|y(t_i) - y_i|}_{=|e_i|} \underbrace{|\partial_y f(t_i, \xi)|}_{\le L} + |\tau(t_i, h)| \le (1 + hL)|e_i| + |\tau(t_i, h)| \le e^{hL}|e_i| + |\tau(t_i, h)|$$

We now repeatedly use this estimate:

$$\begin{aligned} |e_i| &\le e^{hL}|e_{i-1}| + |\tau(t_{i-1}, h)| \le e^{hL}\left[e^{hL}|e_{i-2}| + |\tau(\tau_{i-2}, h)|\right] + |\tau(t_{i-1}, h)| \\ &= e^{2hL}|e_{i-2}| + e^{hL}|\tau(\tau_{i-2}, h)| + |\tau(t_{i-1}, h)| \\ &\le \cdots \le e^{ihL}|e_0| + \sum_{j=0}^{i-1} e^{jhL}|\tau(t_{i-j-1}, h)| \end{aligned}$$

We note that $e_0 = 0$. For the sum, we use that $jh = t_j \le T$ and (9.4) to infer

$$|e_i| \le \sum_{j=0}^{i-1} e^{jhL}|\tau(t_{i-j-1}, h)| \le \frac{1}{2}e^{TL}\sum_{j=0}^{i-1} h^2\|y''\|_{\infty,[0,T+h]} = \frac{1}{2}e^{TL}\|y''\|_{\infty,[0,T+h]} ih^2 \le \frac{1}{2}e^{TL}\|y''\|_{\infty,[0,T+h]} Th,$$

which is the desired first order convergence. □

slide 23 - Euler method

The explicit Euler method was motivated by Taylor expansion around t_i to obtain the value y_{i+1} at t_{i+1}. Alternatively, one could perform Taylor expansion around t_{i+1}. That is,

$$y(t_i) = y(t_{i+1}) + (t_i - t_{i+1}) \underbrace{y'(t_{i+1})}_{=f(t_{i+1}, y(t_{i+1}))} + O(h_i^2),$$

so that, by replacing $y(t_i)$ with y_i and $y(t_{i+1})$ with y_{i+1} and dropping the $O(h_i^2)$, we get the *implicit Euler method*

$$y_{i+1} = y_i + h_i f(t_{i+1}, y_{i+1}). \tag{9.5}$$

The method is *implicit* since y_{i+1} is obtained from y_i by solving a (in general) nonlinear equation.

Exercise 9.2 *Formulate the Newton method to compute y_{i+1} given y_i.*

Analogous to the consistency error for the explicit Euler method (9.4), we have for the consistency error for the implicit Euler method the equation

$$\tau_{iE}(t,h) = y(t+h) - h\left[y(t) + hf(t+h, y(t+h))\right], \tag{9.6}$$

where $t \mapsto y(t)$ is again the exact solution of $y'(t) = f(t, y(t))$. Taylor expansion again gives $\tau_{iE}(t,h) = O(h^2)$ for exact solutions $y \in C^2$. One can show that the implicit Euler method satisfies

$$\max_i |y(t_i) - y_i| \leq Ch.$$

Both explicit and implicit Euler method are *first order* methods.

9.2 Runge-Kutta methods

The explicit and implicit Euler methods are one-step methods[1] of order 1. A generalization of these two one-step methods are methods of the form

$$y_{i+1} = y_i + h_i \Phi(t_i, h_i, y_i, y_{i+1}) \tag{9.7}$$

for some given *increment function* Φ ($\Phi(t_i, h_i, y_i, y_{i+1}) = f(t_i, y_i)$ is the explicit Euler, $\Phi(t_i, h_i, y_i, y_{i+1}) = f(t_i + h_i, y_{i+1})$ is the implicit Euler method.) We are interested in deriving increment functions Φ such that the method is of order p, i.e., that (given sufficient smoothness of f) one has

$$\max_i |y(t_i) - y_i| \leq Ch^p.$$

9.2.1 Explicit Runge-Kutta methods

There are many different ways to introduce one-step methods of order higher than 1. Here, we motivate the structure of so-called *Runge-Kutta*-methods by extrapolation techniques, which we encountered already in Section 1.4. The extrapolation technique relies on comparing two different approximations: a) one step of the explicit Euler with step length h and b) two steps of the explicit Euler method with step length $h/2$, viz

$$\begin{aligned} y_1^{(1)} &= y_0 + hf(t_0, y_0), \\ y_1^{(2)} &= y_{1/2} + \frac{h}{2} f(t_{1/2}, y_{1/2}), \qquad y_{1/2} = y_0 + \frac{h}{2} f(t_0, y_0), \quad t_{1/2} = t + \frac{h}{2} \end{aligned}$$

From the above developments, each of these approximations has error $O(h^2)$, i.e.,

$$\begin{aligned} y(t_1) - y_1^{(1)} &= \tau^{(1)}(t_0, h) = O(h^2), \\ y(t_1) - y_1^{(2)} &= \tau^{(2)}(t_0, h) = O(h^2). \end{aligned}$$

[1] that is, the value y_{i+1} is determined by y_i and not, for example, by y_i and y_{i-1}

We define the actual step of the method as a linear combination of $y_1^{(1)}$ and $y_1^{(2)}$ in such a way that the resulting consistency error is $y(t_1) - y_1 = O(h^3)$. To that end, we carefully employ Taylor's theorem:

$$\begin{aligned}
y(t_1) &= \underbrace{y(t_0)}_{y=y_0} + hy'(t_0) + \frac{1}{2}h^2 y''(t_0) + O(h^3),\\
y_1^{(1)} &= y_0 + hy'(t_0)\\
y_1^{(2)} &= y_{1/2} + \frac{h}{2}f(t_{1/2}, y_{1/2}) = y_0 + \frac{h}{2}f(t_0, y_0) + \frac{h}{2}f(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}f(t_0, y_0))\\
&= y_0 + \frac{h}{2}f(t_0, y_0) + \frac{h}{2}\left[f(t_0, y_0) + \partial_t f(t_0, y_0)\frac{h}{2} + \partial_y f(t_0, y_0)\frac{h}{2}f(t_0, y_0) + O(h^2)\right]\\
&= y_0 + hf(t_0, y_0) + \frac{h^2}{4}\left[\partial_t f(t_0, y_0) + \partial_y f(t_0, y_0)f(t_0, y_0)\right] + O(h^3)
\end{aligned}$$

Next, we use that $t \mapsto y(t)$ is a solution of the differential equation, i.e., $y'(t) = f(t, y(t))$. Hence, by differentation with respect to t we get with the chain rule

$$y''(t) = \partial_t f(t, y(t)) + \partial_y f(t, y(t))y'(t) = \partial_t f(t, y(t)) + \partial_y f(t, y(t))f(t, y(t)).$$

In particular, for $t = t_0$ and $y(t_0) = y_0$, we obtain

$$y_1^{(2)} = y_0 + hy'(t_0) + \frac{h^2}{4}y''(t_0) + O(h^3)$$

Therefore, for parameters α, β we can compute

$$\begin{aligned}
&y(t_1) - [\alpha y_1^{(1)} + \beta y_1^{(2)}]\\
&\quad = y_0 + hy'(t_0) + \frac{h^2}{2}y''(t_0) + O(h^3) - \alpha[y_0 + hy'(t_0, y_0)] - \beta[y_0 + hy'(t_0) + \frac{h^2}{4}y''(t_0) + O(h^3)]\\
&\quad = y_0(1 - \alpha - \beta) + y'(t_0)h[1 - \alpha - \beta] + y''(t_0)h^2\left[\frac{1}{2} - \beta\frac{1}{4}\right] + O(h^3)
\end{aligned}$$

The conditions on α and β are therefore

$$\begin{aligned}
1 - \alpha - \beta &= 0\\
2 - \beta &= 0
\end{aligned}$$

with solution $\beta = 2$ and $\alpha = -1$. The method is therefore $y_1 = 2y_1^{(2)} - y_1^{(1)}$ or, more explicitly,

$$k_1 := f(t_0, y_0), \tag{9.8a}$$

$$k_2 := f(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1), \tag{9.8b}$$

$$y_1 := 2\left(y_0 + \frac{h}{2}k_1 + \frac{h}{2}k_2\right) - y_0 + hk_1) = y_0 + hk_2 \tag{9.8c}$$

This method is of order 2, i.e., $\max_i |y(t_i) - y_i| \leq Ch^2$ (for sufficiently smooth exact solution y). In principle, even higher order methods can be constructed by this extrapolation idea. However, a more general class of methods emerges from the structure in (9.8), the *explicit Runge-Kutta methods*:

Definition 9.3 (explicit Runge-Kutta method) *For a given number of* stages $s \in \mathbb{N}$*, parameters* $c_i \in [0,1]$*,* $b_i \in \mathbb{R}$ *and* $a_{ij} \in \mathbb{R}$ *define*

$$\begin{aligned} k_1 &= f(t_0, y_0), \\ k_2 &= f(t_0 + c_2 h, y_0 + h a_{21} k_1), \\ &\vdots \\ k_s &= f(t_0 + c_s h, y_0 + h \sum_{j=1}^{s-1} a_{sj} k_j) \end{aligned}$$

and the update $y_1 = y_0 + h \sum_{i=1}^{s} b_i k_i$*. The method is compactly described by the* Butcher tableau*:*

$$\begin{array}{c|cccc} 0 & 0 & & & \\ c_2 & a_{21} & 0 & & \\ c_3 & a_{31} & a_{32} & 0 & \\ \vdots & \vdots & & & \\ c_s & a_{s1} & \cdots & a_{s,s-1} & 0 \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

Exercise 9.4 *Write down the Butcher tableau for the explicit Euler method and the method of order* 2 *derived above.*

Example 9.5 (RK4) *A popular explicit Runge-Kutta method is RK4 with 4 stages and order 4 given by* $y_1 = y_0 + h\Phi(t, y_1, h)$*, where*

$$\begin{aligned} \Phi(t,y,h) &:= \frac{1}{6}\left[k_1 + 2k_2 + 2k_3 + k_4\right], \\ k_1 &:= f(t,y), \\ k_2 &:= f\left(t + \frac{h}{2}, y + \frac{1}{2}hk_1\right), \\ k_3 &:= f\left(t + \frac{h}{2}, y + \frac{1}{2}hk_2\right), \\ k_4 &:= f\left(t + h, y + hk_3\right). \end{aligned}$$

The corresponding Butcher tableau is

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array}$$

Exercise 9.6 *Program RK4 and apply it to the right-hand side* $f_1(t,y) = y$ *and* $y_0 = 1$*. Plot the error at* $T = 1$ *versus* h *for* $h = 2^{-n}$*,* $n = 1, \ldots, 10$*.*

Exercise 9.7 *The solution of* $y'(t) = f(t)$*,* $y(t_0) = 0$ *is given by* $y(t) = \int_{t_0}^{t} f(\tau)\, d\tau$*. Hence, for right-hand sides of the form* $f(t,y) = f(t)$*, a Runge-Kutta method results in a quadrature formula. Which quadrature formula is obtained for RK4?*

9.2.2 implicit Runge-Kutta methods

The form of the explicit Runge-Kutta methods in Def. 9.8 suggests a generalization, the so-called *implicit Runge-Kutta methods*:

Definition 9.8 (implicit Runge-Kutta method) *For a given number of* stages $s \in \mathbb{N}$*, parameters* $c_i \in [0,1]$*,* $b_i \in \mathbb{R}$ *and* $a_{ij} \in \mathbb{R}$ *define the stages* k_i*,* $i = 1, \ldots, s$ *as the solution of the following (nonlinear) system of equations:*

$$k_i = f(t_0 + c_i h, y_0 + h \sum_{j=1}^{s} a_{ij} k_j), \qquad i = 1, \ldots, s.$$

One step of the implicit Runge-Kutta is then given by $y_1 = y_0 + h \sum_{i=1}^{s} b_i k_i$*. The method is compactly described by the* Butcher tableau*:*

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{s,s-1} & a_s \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

Exercise 9.9 *Show that the implicit Euler method is a* 1*-stage implicit Runge-Kutta method by writing down the corresponding Butcher tableau.*

Example 9.10 (θ-scheme) *For* $\theta \in [0,1]$ *the scheme with Butcher tableau*

$$\begin{array}{c|c} \theta & \theta \\ \hline & 1 \end{array}$$

is called the θ*-scheme. It is given by*

$$k_1 = f(t_0 + \theta h, y_0 + \theta h k_1), \qquad y_1 = y_0 + h k_1$$

The auxiliary variable k_1 *can be eliminated using* $y_0 + \theta h k_1 = \theta(y_0 + h k_1) + (1-\theta) y_0 = \theta y_1 + (1-\theta) y_1$ *so that it is*

$$y_1 = y_0 + h f(t_0 + \theta h, \theta y_1 + (1-\theta) y_0)$$

We recognize the explicit Euler method for $\theta = 0$ *and the implicit Euler method for* $\theta = 1$*. For* $\theta = 1/2$*, the method is called "midpoint rule" (the simplest Gauss rule). We mention that the* θ*-scheme is of order* 1 *for* $\theta \neq 1/2$ *and it is of order* 2 *for* $\theta = 1/2$*.*

9.2.3 Why implicit methods?

Explicit Runge-Kutta methods are usually preferred over implicit methods as they do not require solving a (nonlinear) equation in each step. These nonlinear equations are typically solved by Newton's method (or some variant), and the user has to provide the derivative $\partial_y f$. Nevertheless, implicit Runge-Kutta methods (or variants) are the method of choice for certain classes of problems such as *stiff ODEs*. A typical situation where implicit methods shine are problems that describe problems with vastly differing time-scales. In these situations, explicit methods would require very small step sizes for reasonable results whereas implicit methods achieve good accuracy with much larger step sizes.

Example 9.11 Consider the solution of initial value problem

$$\mathbf{y}' = \mathbf{A}\mathbf{y}, \qquad \mathbf{y}(0) = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \qquad \mathbf{A} = \begin{pmatrix} -21 & 19 & -20 \\ 19 & -21 & 20 \\ 40 & -40 & -40 \end{pmatrix}.$$

The eigenvalues of $\mathbf{A}$ are given by $\lambda_1 = -2$, $\lambda_2 = -40(1+\mathbf{i})$, $\lambda_3 = -40(1-\mathbf{i})$. The solution is:

$$\begin{aligned} \mathbf{y}_1(t) &= \frac{1}{2}e^{-2t} + \frac{1}{2}e^{-40t}\left(\cos 40t + \sin 40t\right), \\ \mathbf{y}_2(t) &= \frac{1}{2}e^{-2t} - \frac{1}{2}e^{-40t}\left(\cos 40t + \sin 40t\right), \\ \mathbf{y}_3(t) &= -e^{-40t}\left(\cos 40t - \sin 40t\right). \end{aligned}$$

All 3 solution components vary rather rapidly in the regime $0 \le t \le 0.1$ so that a step length restriction $h \ll 1$ seems plausible. For $t > 0.1$, however, the components $\mathbf{y}_1$ and $\mathbf{y}_2$ vary rather slowly (the rapidly oscillatory contribution has been damped out due to the factor e^{-40t}) and $\mathbf{y}_3$ is close to zero. From an approximation point of view, therefore, one would hope that larger time steps are possible. However, Fig. 9.1 shows that, for example, for $h = 0.05$, the explicit Euler method yields completely unacceptable results. Indeed, one can show that the explicit Euler method can only be expected to yield acceptable results if the step length h satisfies the *stability condition*

$$|1 + hz| \le 1 \qquad z \in \{\lambda_1, \lambda_2, \lambda_3\}$$

i.e., it has to satisfy $h \le \frac{1}{40} = 0.025$. In contrast, the implicit Euler method, which is also visible in Fig. 9.1 performs much better since it does not have to satisfy such a stability condition.

slide 24 - Implicit vs. explicit methods

9.2.4 the concept of A-stability (CSE)

The above examples have shown that for certain examples of ODEs explicit methods "fail" in the sense that convergence only sets in for very small step sizes. In contrast, (certain) implicit methods perform well for much larger step sizes. Mathematically, the notion of *A-stability* captures the difference in behavior.

the stability function R

Consider the scalar model equation

$$y' = \lambda y, \qquad y(0) = y_0 \tag{9.9}$$

where $\lambda \in \mathbb{C}$. The exact solution is $y(t) = e^{\lambda t} y_0$. One step of length h of an RK-method has the form

$$y_1 = R(\lambda h) y_0, \tag{9.10}$$

where $R(z)$ is a polynomial for an explicit method and a rational function for an implicit method:

Exercise 9.12 *Show:*

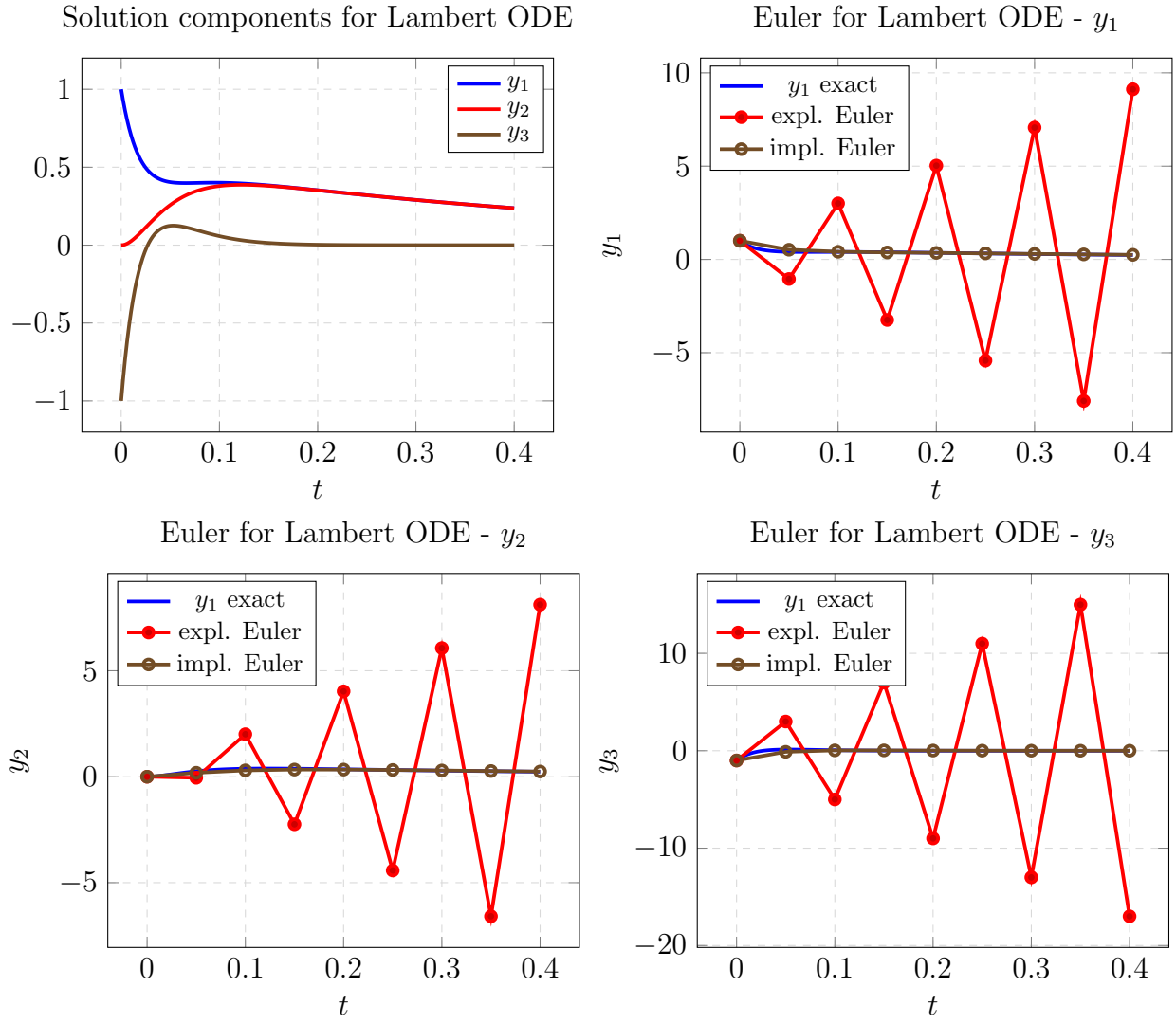


Figure 9.1: Comparison of explicit and implicit Euler method for the stiff problem of Example 9.11: exact solution (top left) and numerical approximation.

1. *explicit Euler method:* $R(z) = 1 + z$
2. *implicit Euler method:* $R(z) = 1/(1 - z)$
3. *θ-scheme with $\theta = 1/2$:* $R(z) = \frac{1+z/2}{1-z/2}$
4. *RK4:* $R(z) = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!}$

Without proof, we mention that for any RK-method that leads to a convergent method the stability function R has the form $R(z) = 1 + z + O(z^2)$ as $z \to 0$.

Definition 9.13 *An RK-method is said to be* A-stable, *if*

$$|R(z)| \leq 1 \qquad \forall z \text{ with } \operatorname{Re} z \leq 0.$$

Exercise 9.14 *If R is a polynomial, then the corresponding RK-method cannot be A-stable. Since the function R associated with an explicit RK-method is a polyomial, explicit RK-methods cannot be A-stable.*
In particular, the explicit Euler method is not *A-stable whereas the implicit Euler method is. The θ-scheme with $\theta = 1/2$ is A-stable. See also Fig. 9.2.*

Consider the case $\operatorname{Re}\lambda \leq 0$. Then the exact solution $y(t) = e^{\lambda t}y_0$ stays bounded for $t \to \infty$. (For $\operatorname{Re}\lambda < 0$ the solution even decays to 0.) From (9.10) we see that the discrete solutions y_i are given by

$$y_i = (R(\lambda h))^i y_0, \qquad i = 0, 1, \ldots,$$

Hence, for the discrete approximations to be bounded (as $i \to \infty$), we have to require $|R(\lambda h)| \leq 1$. Since $\operatorname{Re}\lambda \leq 0$ and $h > 0$, we see that this is ensured for A-stable methods irrespective of $h > 0$.
Put differently: A-stability of an RK-method ensures that the property that the solution $y(t) = e^{\lambda t}y_0$ is bounded (for $\operatorname{Re}\lambda \leq 0$) is reproduced by the numerical method for *any* $h > 0$.

Example 9.15 *A-stability ensures boundedness of the discrete solution for* $\operatorname{Re}\lambda \leq 0$ *and any h. For* $\operatorname{Re}\lambda < 0$ *and* sufficiently small *h the condition $|R(\lambda h)| \leq 1$ is ensured. We illustrate this for the explicit Euler method: For the explicit Euler method, one has $R(\lambda h) = 1 + \lambda h$. Hence, for $\lambda < 0$ one has*

$$|R(\lambda h)| \leq 1 \quad \Longleftrightarrow \quad |1 + \lambda h| \leq 1 \quad \Longleftrightarrow \quad h \leq \frac{2}{|\lambda|}.$$

If $\lambda \ll -1$, then this condition on h is very restrictive.

slide 25 - Stability regions

stability of RK-methods

To get insight into the performance of an RK-method, we consider the *model*

$$\mathbf{y}' = \mathbf{A}\mathbf{y}, \qquad \mathbf{y}(t_0) = \mathbf{y}_0 \tag{9.11}$$

where $\mathbf{A} \in \mathbb{C}^{n\times n}$ is a (constant) matrix. Such a model may be viewed as a linearization of a more complex ODE and one hopes that studying the RK-method applied to the linearization captures the key properties. We assume additionally that $\mathbf{A}$ can be diagonalized:

$$\mathbf{A} = \mathbf{T}^{-1}\mathbf{D}\mathbf{T}$$

so that after the change of variables $\widehat{\mathbf{y}} = \mathbf{T}\mathbf{y}$ the ODE (9.11) is equivalent to

$$\widehat{\mathbf{y}}' = \mathbf{D}\widehat{\mathbf{y}}, \qquad \widehat{\mathbf{y}}(t_0) = \widehat{\mathbf{y}}_0 = \mathbf{T}\mathbf{y}_0. \tag{9.12}$$

It can be checked that for RK-methods, one step of the RK-method could be computed in two different ways: either one applies the RK-method directly to (9.11) or one applies it to the transformed equation (9.12) and transforms back. This is depicted in Fig. 9.3. The RK-method applied to (9.12) is simpler to understand since it reduces to the application of the RK-method to *scalar* problems of the form (9.9) where $\lambda \in \mathbb{C}$ is a diagonal entry of $\mathbf{D}$, i.e., an eigenvalue

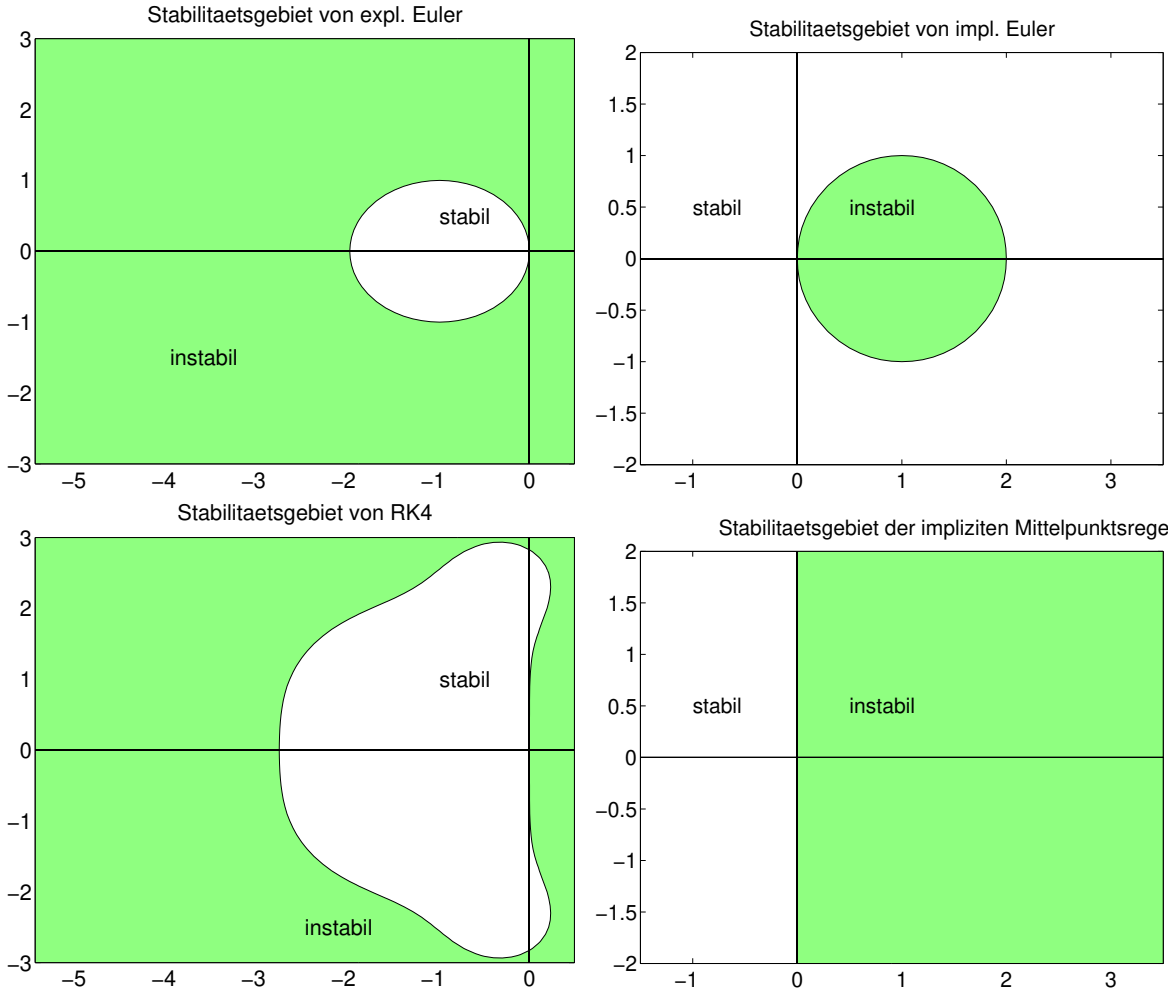


Figure 9.2: stability regions $\{z \in \mathbb{C} \,|\, |R(z)| \leq 1\}$ for explicit Euler, implicit Euler, RK4, and implicit midpoint rule.

of $\mathbf{A}$. One step of length h of an RK-method applied to (9.9) has the form $\widehat{\mathbf{y}}_k^{i+1} = R(\lambda_k h)\widehat{\mathbf{y}}_k^i$ where $R(z)$ is the *stability function* and we use the subscript k to indicate the component of the vector $\widehat{\mathbf{y}}$ while the superscripts $i+1$ and i indicate the association with time steps t_{i+1} and t_i. Hence,

$$\widehat{\mathbf{y}}_k^{i+1} = R(\lambda_k h)\widehat{\mathbf{y}}_k^i = \cdots = (R(\lambda_k h))^{i+1}\widehat{\mathbf{y}}_k^0, \qquad k = 1, \ldots, n,$$

If $\operatorname{Re}\lambda_k \ll -1$ then one is well-advised to ensure $|R(\lambda_k h)| \leq 1$ to reproduce this boundedness of the exact solution component. For A-stable methods, this is ensured no matter what h is. The following considerations argue why this is a sensible condition. For simplicity of notation, we assume that the eigenvalues λ_k are real (so as to be able to formulate conditions on λ_k instead of on $\operatorname{Re}\lambda_k$):

- One may expect a good approximation for those components $\widehat{\mathbf{y}}_k$ for which $|\lambda_k h|$ is small. For these components, one has $R(\lambda_k h) \approx 1$ (note that $R(z) = 1 + O(z)$ in the examples of Exercise 9.12). Suppose that for some $\lambda_k \ll -1$ and an $h > 0$ one has $|R(\lambda_k h)| > 1$. If the RK-method is applied to the diagonalized form (9.12), then the error in these component $\widehat{\mathbf{y}}_k$ is very large while the other components may be reasonably well approximated. One may be tempted to argue that this is acceptable since that solution component is practically zero (and thus known!) so that there is no need to approximate it numerically anyway. However, if the RK-method is applied to the original form (9.11), then the presence of a single eigenvalue λ_k with $|R(\lambda_k h)| > 1$ will ruin all components since the transformation $\mathbf{y} = \mathbf{T}^{-1}\widehat{\mathbf{y}}$ mixes all components of $\widehat{\mathbf{y}}$ so that one expects that all components of $\mathbf{y}^1$ have contributions of $\widehat{\mathbf{y}}_k^1$ (unless $\mathbf{T}^{-1}$ has special structure). In other words: **When applying the RK-method to (9.11), the time step $h > 0$ is dictated by the maximum of $\{-\lambda_j \,|\, j = 1, \ldots, n\}$.** However, is very unsatisfactory that solution components $\widehat{\mathbf{y}}_k$ with large $-\lambda_k$ dictate the step size although they hardly contribute to the exact solution.

- Related to the above point is a consideration of error propagation. In each step of the RK, some consistency error is made. Consider again the RK-method in the variable $\widehat{\mathbf{y}}$ and an initial error $\widehat{\mathbf{e}}^0$. For $\lambda_k \ll -1$ and $|R(\lambda_k h)| > 1$ the error in the kth component is actually damped by the exact evolution (by a factor $e^{\lambda_k h}$) whereas it is amplified by a factor $|R(\lambda_k h)|$ by the RK-method. Thus, initial errors are amplified by a factor $|R(\lambda_k h)|^i$ in the ith step. Fixing $t_i = ih$, we rewrite this amplification factor as

  $$|R(\lambda_k h)|^i = |R(\lambda_k h)|^{t_i/h} = \left(|R(\lambda_k h)|^{1/h}\right)^{t_i}.$$

  For fixed t_i and $|R(\lambda_k h)| > 1$, we have that $|R(\lambda_k h)|^{1/h}$ is very large. In conclusion, we have to expect that the method will dramatically amplify initial errors for small h. Again, this error amplification in one component will affect all components if one applies the RK-method to the original form (9.11).

What about $\lambda_k > 0$? In a nutshell: large (positive) λ_k also impose step size restrictions, i.e., they also require that $\lambda_k h$ be small. However, this step size restriction is acceptable since it is necessary to approximate the solution. To be more specific, we note that the error amplification discussed above arises for $|R(\lambda_k h)| > 1$ and this situation occurs also for $\lambda_k > 0$ (e.g., for the explicit Euler method is $R(\lambda_k h) = 1 + \lambda_k h$). However, the exact solution grows as well so that

$$\begin{array}{ccc} \mathbf{y}^i & \xrightarrow[\widehat{\mathbf{y}}^i = \mathbf{T}\mathbf{y}^i]{\text{change of variables}} & \widehat{\mathbf{y}}^i \\ \text{RK method}\downarrow & & \downarrow\text{RK method} \\ \mathbf{y}^{i+1} & \xleftarrow[\mathbf{y}_i = \mathbf{T}^{-1}\widehat{\mathbf{y}}^i]{\text{change of variables}} & \widehat{\mathbf{y}}^{i+1} \end{array}$$

Figure 9.3: RK-method applied to $\mathbf{y}' = \mathbf{A}\mathbf{y}$ and to $\widehat{\mathbf{y}} = \mathbf{D}\widehat{\mathbf{y}}$ after change of variables. The superscripts i and $i+1$ refer to the time steps t_i and t_{i+1}.

the amplification of the relative error is not dramatic. To fix ideas, consider the explicit Euler method. Then with initial error $\widehat{\mathbf{e}}_k$ the relative error at t_i is

$$\frac{|R(\lambda_k h)|^i |\widehat{\mathbf{e}}_k^0|}{|\widehat{\mathbf{y}}_k^0|} e^{\lambda_k t_i} = \frac{|\widehat{\mathbf{e}}_k^0|}{|\widehat{\mathbf{y}}_k^0|} \frac{(1+\lambda_k h)^{t_i/h}}{e^{\lambda_k t_i}} \leq \frac{|\widehat{\mathbf{e}}_k^0|}{|\widehat{\mathbf{y}}_k^0|} = \text{rel. error at } t_0,$$

where we used $(1+x) \leq e^x$ so that $(1+\lambda_k h)^{t_i \lambda_k/(\lambda_k h)} \leq e^{t_i \lambda_k}$.

9.3 Multistep methods (CSE)

goal: high order methods with less function evaluations than RK-methods
observation: one-step methods such as RK-methods do *not* make use of the "history" available
$\rightarrow$ reuse previous function evaluations for efficiency increase
setting: we employ uniform step size h (multistep methods with variable step size are rather complicated!)
notation: $t_i = t_0 + ih$, $f_i := f(t_i, y_i)$

9.3.1 Adams-Bashforth methods

Let $r \in \mathbb{N}_0$. Given $(t_{i-k}, y_{i-k}), \ldots, (t_i, y_i)$ we wish to find y_{i+1}. To motivate the method, we note that the exact solution satisfies

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} y'(t)\,dt = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))\,dt. \tag{9.13}$$

Noting that we have approximations to $f(t_{i-j}, y(t_{i-j})) \approx f_{i-j}$, $j = 0, \ldots, k$, in our hand, we approximate the integrand by its interpolating polynomial of degree k, i.e.,

$$f(t, y(t)) \approx \sum_{j=0}^{k} f_{i-j}\ell_{i-j}(t), \qquad \ell_{i-j}(t) := \prod_{\substack{s=0 \\ s\neq j}}^{k} \frac{t - t_{i-s}}{t_{j-s} - t_{i-s}}$$

In view of $t_i = t_0 + hi$ and the change of variables $t = t_i + h\tau$ we compute

$$\int_{t_i}^{t_{i+1}} \sum_{j=0}^{k} f_{i-j}\ell_{i-j}(t)\,dt = \sum_{j=0}^{k} f_{i-j} \int_{t_i}^{t_{i+1}} \prod_{\substack{s=0 \\ s\neq j}}^{k} \frac{t - t_{i-s}}{t_{j-s} - t_{i-s}} = \sum_{j=0}^{k} f_{i-j} h \underbrace{\int_{\tau=0}^{\tau=1} \prod_{\substack{s=0 \\ s\neq j}}^{k} \frac{\tau + s}{j - i}\,d\tau}_{\beta_j}$$

We note that β_j is independent of h and f and can be precomputed (see Example 9.16). We have thus arrived at the *Adams-Bashforth method*

$$y_{i+1} = y_i + h\sum_{j=0}^{k} \beta_j f_{i-j} \tag{9.14}$$

Example 9.16 Adams-Bashforth methods *(explicit):*

$$\begin{aligned}
k = 0 \quad & y_{i+1} = y_i + hf_i \qquad \textit{(explicit Euler)} \\
k = 1 \quad & y_{i+1} = y_i + h\frac{1}{2}\left(3f_i - f_{i-1}\right) \\
k = 2 \quad & y_{i+1} = y_i + h\frac{1}{12}\left(23f_i - 16f_{i-1} + 5f_{i-2}\right) \\
k = 3 \quad & y_{i+1} = y_i + h\frac{1}{24}\left(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}\right)
\end{aligned}$$

Theorem 9.17 *The $k+1$-step Adams-Bashforth method is a method of order $k+1$, i.e., if f is sufficiently smooth, then the consistency error (i.e., the error in one step) is $O(h^{k+2})$. Specifically, provided the initial errors $|y(t_i) - y_i|$, $i = 0, \ldots, k$, are $O(h^{k+1})$ then*

$$\max_i |y(t_i) - y_i| \leq Ch^{k+1},$$

where the constant C depends on f and the terminal time $T = t_N$.

We note that the method requires only one evaluation of f per step. It is therefore more economical than the RK-methods (if the number of function evaluations is taken as the cost measure).

slide 25a - An unstable 2-step method

9.3.2 Adams-Moulton methods

The Adams-Bashforth method above is an explicit method. The Adams method exist also in implicit variant. This method is derived in a way very similar to (9.13) by simply changing the domain of integration. The exact solution satisfies

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))\, dt. \tag{9.15}$$

Again, replacing the integrand with the polynomial interpolating in (t_{i+1-j}, y_{i+1-j}), $j = 0, \ldots, k$, we arrive at the method

$$y_{i+1} = y_i + \sum_{j=0}^{k} f_{i+1-j} \int_{t_i}^{t_{i+1}} \ell_{i+1-j}(t)\, dt, \qquad \ell_{i+1-j}(t) = \prod_{\substack{s=0 \\ s\neq j}}^{k} \frac{t - t_{i+1-s}}{t_{i+1-j} - t_{i+1-s}}\, dt$$

Again, the terms $\int_{t_i}^{t_{i+1}} \ell_{i+1-j}(t)\, dt$ are of the form $h\beta_j$ for some coefficients β_j depending only on k (Exercise!). The first few Adams-Moulton methods are given in the following example:

Example 9.18 Adams-Moulton method *(implicit):*

$$\begin{aligned}
k = 0 \quad & y_{i+1} = y_i + hf_{i+1} && \textit{(implicit Euler)} \\
k = 1 \quad & y_{i+1} = y_i + h\frac{1}{2}(f_{i+1} + f_i) && \textit{(trapezoidal rule)} \\
k = 2 \quad & y_{i+1} = y_i + h\frac{1}{12}(5f_{i+1} + 8f_i - f_{i-1}) \\
k = 3 \quad & y_{i+1} = y_i + h\frac{1}{24}(9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2})
\end{aligned}$$

Theorem 9.19 *The k-step Adams-Moulton method[2] is a method of order $k+1$, i.e., if f is sufficiently smooth, then the consistency error (i.e., the error in one step) is $O(h^{k+2})$. Specifically, provided the initial errors $|y(t_i) - y_i|$, $i = 0, \ldots, k$, are $O(h^{k+1})$ then*

$$\max_i |y(t_i) - y_i| \leq Ch^{k+1},$$

where the constant C depends on f and the terminal time $T = t_N$.

[2]for $k = 0$, the Adams-Moulton method is the implicit Euler method, i.e., also a 1-step method

The Adams-Moulton methods are *implicit* methods since the involve $f_{i+1} = f(t_{i+1}, y_{i+1})$ to compute the y_{i+1}.

Exercise 9.20 *The Adams-Moulton method have fixed point form for the unknown y_{i+1} and could be solved with the fixed point iteration (cf. (6.1)). Assume that f is smooth. Show, using Theorem 6.5, that the fixed point iteration converges for sufficiently small h.*

In practice of Adams-Moulton methods, the fixed point iteration discussed in Exercise 9.20 is refined:

Remark 9.21 (predictor-corrector method) *A reasonable starting guess for y_{i+1} is y_i. (By Taylor, one expects $y_{i+1} - y_i = O(h)$.) A better approximation could be obtained by an explicit Adams-Bashforth method. Then, one performs m steps of the fixed-point iteration (9.16). That is,*

$$
\begin{aligned}
&y_{i+1}^{(0)} := \textit{approximation obtained from an Adams-Bashforth method} \qquad (P)\\
&\textit{for } n = 0, \dots, m-1 \textit{ do}\{\\
&\quad f_{i+1}^{(n)} := f(t_{i+1}, y_{i+1}^{(n)}) \qquad (E)\\
&\quad y_{i+1}^{(n+1)} := y_i + h\beta_0 f_{i+1}^{(n)} + h\sum_{j=1}^{k} f(t_{i+1-j} y_{i+1-j}), \qquad (C)\\
&\}
\end{aligned}
$$

In this context, the first step is called "prediction" (denoted P) and the m steps of evaluating f are called "evaluate" (denoted E) and "correction" (denoted C). Correspondingly method is abbreviated $P(EC)^m$.

9.3.3 BDF methods

The most important class of linear multistep methods are the BDF methods ("backward differentiation methods"). The BDF methods are suitable for stiff problems.
The BDF-k methods is obtained by interpolating (t_{i+1-j}, y_{i+1-j}), $j = 0, \dots, k$ by a polynomial of degree k and *collocating* the differential equation in the point t_{i+1}:

$$
\text{interpolating polynomial:} \qquad \pi_k(t) := \sum_{j=0}^{k} y_{i+1-j}\ell_{i+1-j}(t), \qquad \ell_{i+1-j}(t) = \prod_{\substack{s=0\\ s\neq j}}^{k} \frac{t - t_{i+1-s}}{t_{i+1-j} - t_{i+1-s}},
$$

$$
\text{collocate in } t_{i+1}: \quad \pi_k'(t_{i+1}) \overset{!}{=} f(t_{i+1}, \pi_k(t_{i+1})) = f(t_{i+1}, y_{i+1}).
$$

Example 9.22 *We consider the simplest case, $k = 1$:*

$$
\begin{aligned}
&\textit{interpolating polynomial:} && \pi_k(t) := y_{i+1}\frac{t - t_i}{h} - y_i\frac{t - t_{i+1}}{h}\\
&\textit{collocate in } t_{i+1}: && \pi_k'(t_{i+1}) = \frac{y_{i+1} - y_i}{h} \overset{!}{=} f(t_{i+1}, \pi_k(t_{i+1})) = f(t_{i+1}, y_{i+1}).
\end{aligned}
$$

This is the implicit Euler method.

More generally, we have:

Example 9.23

$$
\begin{aligned}
&k=1 \qquad y_{i+1}-y_i = hf_{i+1} \qquad \textit{(implict Euler)}\\
&k=2 \qquad y_{i+1}-\frac{4}{3}y_i+\frac{1}{3}y_{i-1} = h\frac{2}{3}f_{i+1} \qquad \textit{BDF2}\\
&k=3 \qquad y_{i+1}-\frac{18}{11}y_i+\frac{9}{11}y_{i-1}-\frac{2}{11}y_{i-2} = h\frac{6}{11}f_{i+1} \qquad \textit{BDF3.}
\end{aligned}
$$

Theorem 9.24 *The BDFk-method is a method of order k, i.e., if f is sufficiently smooth then the consistency error (i.e., the error in one step of the method) is $O(h^{k+1})$. Specifically, provided the initial errors $|y(t_i)-y_i|$, $i=0,\dots,k-1$, are $O(h^k)$ then*

$$\max_i |y(t_i)-y_i| \le Ch^k$$

Remark 9.25 *Since the BDF methods are typically employed for stiff problems, the implicit equations are* not *solved by a simple fixed point iteration but by Newton's method or a Newton-like method.*

9.3.4 Remarks on multistep methods

The Adams methods and the BDF-formulas are special cases of so-called *linear multistep methods*, which are update formulas of the form

$$\sum_{j=0}^{k} \alpha_j y_{i+1-j} = h\sum_{j=0}^{k} \beta_j f_{i+1-j} \tag{9.16}$$

for coefficients α_j, β_j. These methods are explicit if $\beta_j = 0$, otherwise they are implicit.

Starting value:

A multistep method determines y_{i+1} from several previous values y_{i+1-j}, $j=1,\dots,k$. Since at the beginning only y_0 is given, one needs to compute the initial values $y_1,\dots,y_k$ by some other method. For example, a Runge-Kutta method is employed. These values need to be computed to accuracy $O(h^p)$, where p is the order of the multistep method ($p=k+1$ for Adams-Bashforth and Adams-Moulton, $p=k$ for BDFk).

A Notations

A.1 Function spaces

Functions spaces are special vector spaces (i.e. spaces that allow operations + and scalar multiplication together with some rules), where every object in the space is a function $f : \Omega \to \mathbb{R}$, where Ω is some set.
Hereby, the vector space operations read as: For any $x \in \Omega$ and any $\lambda \in \mathbb{R}$, there holds

$$(f+g)(x) = f(x) + g(x)$$
$$(\lambda f)(x) = \lambda f(x)$$

In this lecture notes, we use the following function spaces:

- **Polynomials:** A polynomial is a function

  $$x \mapsto \sum_{\ell=0}^{n} p_\ell x^\ell$$

  with $p_\ell \in \mathbb{R}, \ell = 0, \dots, n$ being the coefficients of the polynomial and n being the degree. The function space of polynomials of maximal degree n is denoted by $\mathcal{P}_n$.

- **Continuous functions:** Let $[a, b] \subset \mathbb{R}$ be an intervall. Then, by $C([a, b])$, we denote the set of all continuous functions on $[a, b]$.

- **Continuously differentiable functions:** Let $[a, b] \subset \mathbb{R}$ be an intervall and $p \in \mathbb{N}$. Then,

  $$C^p([a, b]) := \{f \in C([a, b]) \ : \ f^{(p)} \in C([a, b])\}$$

  denote the set of all p-times continuously differentiable functions on $[a, b]$.

Sometimes, the term **smooth functions** is loosely used, which means that the number $p \in \mathbb{N}$, such that $f \in C^p([a, b])$ holds, is as large as one needs.

A.2 Norms and inner products

Definition A.1 *Let V be a vector space. A mapping $\|\cdot\| : V \to \mathbb{R}$ is called a* norm*, if*

(i) (triangle inequality) $\|\mathbf{x} + \mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y} \in V$

(ii) (homogeneity) $\|\lambda\mathbf{x}\| = |\lambda|\|\mathbf{x}\|$ for all $\mathbf{x} \in V$, $\lambda \in \mathbb{R}$

(iii) (definiteness) $\|\mathbf{x}\| \ge 0$ for all $\mathbf{x} \in V$, and $\|\mathbf{x}\| = 0$ implies $\mathbf{x} = 0$.

Important norms on $V = \mathbb{R}^n$ are:

1. the *euklidian norm* $\|\mathbf{x}\|_2 := \sqrt{\sum_{i=1}^{n} |\mathbf{x}_i|^2}$

2. the ∞-*norm* $\|\mathbf{x}\|_\infty := \max_{i=1,\dots,n} |\mathbf{x}_i|$
3. the 1-*norm* $\|\mathbf{x}\|_1 := \sum_{i=1}^n |\mathbf{x}_i|$

Important norms on function spaces (e.g. $C([a,b])$) are:

1. the *maximum norm* $\|f\|_{\infty,[a,b]} := \max_{x\in[a,b]} |f(x)|$
2. the L^2-*norm* $\|f\|_{L^2(a,b)} := \sqrt{\int_a^b f(x)^2\,dx}$

Definition A.2 *An* inner-product *(also called scalar-product) is a scalar-valued function* $(\cdot,\cdot) : V \times V \to \mathbb{R}$*, which acts on tuples of vectors of a vector space* V*, and satisfies*

1. *(symmetry)* $(x,y) = (y,x)$ *for all* $x, y \in V$*.*
2. *(linearity)* $(\alpha x + \beta y, z) = \alpha(x,z) + \beta(y,z) \qquad \alpha,\beta \in \mathbb{R}, x, y, z \in V.$
3. *(definiteness)* $(x,x) \geq 0$ *for all* $x \in V$ *and* $(x,x) = 0 \iff x = 0.$

Example A.3

- *The Euclidean scalar product on* $\mathbb{R}^n$ *is defined as*

$$(x,y)_2 := x_1y_1 + \cdots + x_ny_n.$$

- *Let* $A \in \mathbb{R}^{n\times n}$ *be a symmetrix, positive definite matrix. Then,* A *induces the inner product*

$$(x,y)_A := (Ax,y)_2.$$

On inner product spaces (vector spaces V with an inner product $(\cdot,\cdot)$ defined on $V \times V$) we have several important geometric properties.

- Schwarz inequality: For all $x, y \in V$, we have

$$|(x,y)| \leq \|x\| \cdot \|y\|.$$

- Measuring of angles: For all $x, y \in V$, we can define the angle α between the vectors x, y by

$$\cos\alpha = \frac{(x,y)}{\|x\| \cdot \|y\|},$$

  which generalizes the formula in the Euclidean space by using the (general) inner product and norm.

- Orthogonality: Let $x, y \in V$. We call x, y *orthogonal*, if

$$(x,y) = 0.$$

  Note that using that in the above formula for the angle gives $\cos(\alpha) = 0$ or $\alpha = \frac{\pi}{2}$, which coincides with the geometric interpretation of orthogonality as vectors which span an angle of 90 degrees.

A.3 Further notations

- The Kronecker Delta δ_{ij} is a shorthand for

$$\delta_{ij} := \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

A.3.1 The $O(\cdot)$-notation

- Let $x \mapsto f(x)$ be a function and $x_0 \in \mathbb{R} \cup \pm\infty$. The notation (called Landau symbol)

$$O(f(x)) \qquad x \to x_0$$

  describes a (non-specified) function g with the property that

$$|g(x)| \leq C|f(x)| \quad \text{for all } x \text{ sufficiently close to } x_0.$$

- Oftentimes the $O(\cdot)$-notation is used to describe some asymptotics for $n \to \infty$ or $h \to 0$. Examples can be found throughout these lecture notes, such as cost of an algorithm in $O(n^2)$. E.g., the function $f(n) = 2n(n+1)$ is $O(n^2)$ as there holds

$$2n(n+1) \leq 4n^2 \qquad \text{for all } n > 0.$$

B Polynomial approximation

By the following theorem, called Weierstraß theorem, it is always possible to approximate a given continuous function by a polynomial up to a prescribed tolerance.

Theorem B.1 (Weierstraß) *Let $f \in C([a,b])$. Then, for every $\varepsilon > 0$, there exists a polynomial p such that*

$$\|f - p\|_{\infty,[a,b]} \leq \varepsilon.$$

Note that this theorem does not provide a way to construct a polynomial approximation. For smooth function, a possible approximation is given by the Taylor polynomial, recalled in the following theorem.

Theorem B.2 (Taylor) *Let $f \in C^{p+1}([a,b])$ and $x_0 \in (a,b)$. Define the p-th order Taylor polynomial of f about the point x_0*

$$T_p(x) := \sum_{j=0}^{p} \frac{1}{j!} f^{(j)}(x_0)(x - x_0)^j. \tag{B.1}$$

Then, there holds

$$f(x) = T_p(x) + \frac{1}{p!} \int_{x_0}^{x} (x-t)^p f^{(p+1)}(t)\, dt.$$

Thus, every function $f \in C^{p+1}([a,b])$ can be written as a sum of a polynomial and a remainder

$$R_p(x) := \frac{1}{p!} \int_{x_0}^{x} (x-t)^p f^{(p+1)}(t)\, dt.$$

In other words: The function f can be approximated by its Taylor polynomial with pointwise error $R_p(x)$. From the definition, one directly infers the error bound

$$|R_p(x)| \leq \frac{1}{p!} |x - x_0|^{p+1} \max_{\xi \in \langle x_0, x \rangle} |f^{(p+1)}(\xi)|. \tag{B.2}$$

Often, one simply writes

$$f(x) = T_p(x) + O(|x - x_0|^{p+1}).$$