Numerical Methods

Numerical Methods

Piotr Tatjewski

Warszawa 2014
Oficyna Wydawnicza Politechniki Warszawskiej

Author:
Piotr Tatjewski, Institute of Control and Computation Engineering,
Faculty of Electronics and Information Technology, Warsaw University of Technology

Title of the Polish edition:
Metody Numeryczne (Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2013)

Reviewer of the Polish edition:
Professor Jerzy Klamka, D.Sc. Ph.D.

Editor:
Teresa Woźniak

ISBN 978-83-7814-263-8

Internet bookshop of the Warsaw University of Technology Publishing House (OWPW):
http://www.wydawnictwopw.pl; e-mail: oficyna@wpw.pw.edu.pl
phone 48 22 234-75-03; fax 48 22 234-70-60

Oficyna Wydawnicza Politechniki Warszawskiej
(Warsaw University of Technology Publishing House)
ul. Polna 50, 00-644 Warszawa, Poland
Printed and bound by OWPW, phone 48 22 234-55-93, 1st edition, order no 247/2014

Contents

Preface

The presented text is an English version of the original textbook written in Polish (”Metody numeryczne”), published in 2013 by OWPW (Oficyna Wydawnicza Politechniki Warszawskiej – Publishing Office of the Warsaw University of Technology).

The textbook was written based on the author’s long experience, as a teacher of several courses on numerical methods, at the Faculty of Electronics and Information Technology, Warsaw University of Technology. Recently, it was the course ”Numerical Methods” taught in Polish for ICT students and in English for Electrical and Computer Engineering students. The textbook sums up the experience of a long series of previously prepared Lecture Notes, successively improved from year to year, and available for students in an electronic form at the web page of the course.

The author would like to express his gratitude to all Students actively taking part in the courses, for all comments concerning the way of presentation, for indications of editorial errors. All these comments helped the author to prepare subsequent, better versions and finally resulted in the presented textbook.

Piotr Tatjewski
Warszawa,
September 2014

Chapter 1

Preliminaries

1.1. Computer representation of numbers, representation errors

The floating point representation $x_{t,r}$ of a real number x is defined in the following way:

$$x_{t,r} = m_t \cdot P^{c_r},$$

where: m_t – mantissa, t – number of positions in the mantissa,
c_r – exponent, r – number of positions in the exponent,
P – base, the case $P = 2$ will further only be considered.

To get a well-defined floating-point representation, the range of numbers represented by the mantissa itself should be defined – the mantissa should be *normalized*. The most natural representation of a binary number with a normalized mantissa is obtained assuming that

$$0.5 \leq |m_t| < 1, \tag{1.1}$$

because then the mantissa describes the number represented by first t negative powers of 2, i.e., first t digits of a binary number after the (decimal) point, e.g., $m_7 = 1011011$ corresponds to 0.1011011, $m_4 = 1011$ corresponds to 0.1011, etc. For instance, for $t = 4$, $r = 2$ and two bits for the signs, consider the floating-point binary representation:

$$\mathbf{0}10|\mathbf{0}1011,$$

where the vertical bar separates fields for the exponent and the mantissa, whereas the sign bits (in bold; 0 describes the positive sign, $(-1)^0 = 1$) are the first ones in places dedicated to both the exponent and the mantissa. In the decimal representation, the given number is equal to:

$$x = 2^{(-1)^0(1\cdot 2^1+0\cdot 2^0)} \cdot (-1)^0(1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4}) = 2.75$$

and subsequent powers of 2 smoothly change passing from the mantissa to the exponent (in ascending order). However, in the IEEE 754 standard for floating-point representation another *convention of normalization* has been assumed:

$$1 \leq |m_t| < 2, \tag{1.2}$$

i.e., the point (separating the fractional part of a number) is located after the first position of the mantissa treated as an individual number, e.g., $m_7 = 1011011$ corresponds to 1.011011, $m_4 = 1011$ corresponds to 1.011 (remember that 1 is always at the first position of any normalized mantissa). The previously presented number (2.75) in this convention is represented as follows (the mantissa multiplied by 2, the exponent less by one):

$$x = 2^{(-1)^0(0\cdot 2^1+1\cdot 2^0)} \cdot (-1)^0(1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}) = 2.75.$$

Moreover, the exponent in the IEEE 754 standard is represented as a binary number without the sign bit, but shifted by an appropriate value to represent both negative and positive exponents.

According to the *IEEE 754 standard*, the format of a 32-bits number has:

- 24 bits for the mantissa, with the normalization: $1 \leq |m_t| < 2$,
- 8 bits for the exponent, coded with the shift by $2^7 - 1 = 127$, i.e.,

sign bit (1 bit)	exponent (shifted) (8 bits)	24 bits normalized mantissa (first bit omitteb – it is dalways 1) (23 bits)

Therefore, the representation of a number according to this standard, is as follows:

$$x_{IEEE754,32} = (-1)^s \cdot m_{24} \cdot 2^{c_8 - 127},$$

where s – the sign bit, c_8 – the exponent shifted by 127, i.e., the exponent covers numbers from -127 (8 zeroes) to $+128$ (8 ones). The zero exponent (corresponding to decimal numbers from the interval [1,2)) is represented by ”01111111”. The representation of the number 2,75, discussed before, has in this standard the following binary form (where fields for the sign, exponent and mantissa are separated by vertical bars, to make the reading easier):

0|10000000|01100000000000000000000000000000

Notice that a binary (base 2) number with the 24-bits mantissa corresponds to a decimal number with 7 significant positions.

For any given values of t and r, the set of computer floating-point numbers $M \subset \mathbb{R}$ is *finite*, moreover:

- the larger t the more numbers is contained in the same interval (the set M is more ”dense”),
- the larger r the wider interval of numbers is covered by the set M.

Assuming the representation of the exponent is exact, i.e. $c_r = c$ (the set M covers the whole range of numbers of interest), and denoting floating-point representation of a number x by rd(x), we have

$$\mathrm{rd}\,(x) = m_t \cdot 2^c.$$

The floating-point representation is most exact, if

$$|\mathrm{rd}\,(x) - x| \leq \min_{g \in M} |g - x|\,.$$

This requirement is fulfilled by a standard *rounding*, which for the mantissa normalized according to (1.1) can be written in the form

$$m_t = \sum_{i=1}^{t} e_{-i} \cdot 2^{-i} + e_{-(t+1)} \cdot 2^{-t}, \tag{1.3}$$

where $e_{-1} = 1$, $e_{-i} = 0$ or 1 for $i = 2, 3, \ldots, t+1$. Hence, the *roundoff error* can be estimated as follows:

$$|m - m_t| \leq 2^{-(t+1)},$$

where m denotes the exact mantissa ($x = m \cdot 2^c$). If there is zero at the first truncated position, then we have only a truncation and the mantissa m_t is smaller than true one m, with an error not exceeding $2^{-(t+1)}$. However, if there is one at the first truncated position – which means that the truncated part of the true mantissa is a number from the range $[2^{-(t+1)}, 2^{-t})$, then the rounding (i.e., adding 2^{-t} to the remaining bits) takes place and the mantissa m_t is larger than the true one, again with an error not exceeding $2^{-(t+1)}$.

In fact, we are interested in *relative errors of floating-point representation of real numbers*. For the relative error we have

$$\frac{\mathrm{rd}\,(x) - x}{x} = \frac{(m_t - m)2^c}{m2^c} = \frac{m_t - m}{m},$$

thus *the roundoff relative error* satisfies

$$\left|\frac{\mathrm{rd}\,(x) - x}{x}\right| = \frac{|m_t - m|}{|m|} \leq \frac{2^{-(t+1)}}{2^{-1}} = 2^{-t},$$

because the mantissa normalized according to (1.1) satisfies $|m| \geq 2^{-1}$.

The relation

$$\left|\frac{\mathrm{rd}\,(x) - x}{x}\right| = \frac{|m_t - m|}{|m|} \leq 2^{-t} \tag{1.4}$$

is universally true, it does not depend on the assumed pattern of mantissa normalization – as different normalizations differ only in multiplication by an appropriate power of number 2, comp. (1.1) and (1.2), which is reduced when considering the relative error. Therefore, *the maximal possible relative error of the floating-point representation depends only on the number of bits of the mantissa*, it is called *the machine precision* and traditionally denoted by eps.

For the t-bit mantissa rounded according to (1.3) (for any normalization pattern), the machine precision $eps = 2^{-t}$.

The relation (1.4) can be, for any number x, written in the equivalent form

$$\mathrm{rd(x)} - x = x \cdot \varepsilon, \quad \text{where } |\varepsilon| \le 2^{-t} = eps,$$

thus

$$\mathrm{rd}\,(x) = x\,(1+\varepsilon)\,, \qquad |\varepsilon| \le eps, \tag{1.5}$$

where ε represents the relative rounding error of the number x. Equation (1.5) is *fundamental for the numerical analysis*.

It can be easily concluded that using the double-length mantissa results in

$$eps_{mdl} = (eps)^2,$$

where the subscript "mdl" denotes "mantissa double-length". According to the IEEE Standard 754 a *double precision* number is coded using 64 bits, as follows:

sign bit (1 bit)	exponent shifted (11 bits)	normalized 53-bits mantissa (first bit always 1 – omitted) (52 bits)

A binary number with 53-bits mantissa corresponds to a decimal number with 16 significant positions.

Base 2 t-digit floating-point representation resulting from the *truncation* (*chopping*) has, using normalization (1.1), the mantissa in the form (compare with (1.3)):

$$m_t = \sum_{i=1}^{t} e_{-i} \cdot 2^{-i},$$

where now the number representation error is called the *chopping (or truncation) error*. Its upper bound can be derived analogously as for the truncation error, resulting with twice larger $eps = 2^{-(t+1)}$, instead of $eps = 2^{-t}$,

$$\mathrm{rd}\,(x) = x\,(1+\varepsilon)\,, \qquad |\varepsilon| \le 2^{-t+1} = eps.$$

1.2. Floating-point arithmetic

Addition $(+)$, subtraction $(-)$, multiplication $(\cdot)$ and division $(/)$ are *elementary arithmetic operations*. The result of an exact elementary arithmetic operation on floating-point arguments does not belong, in general, to the set of computer floating-point numbers (due to a possible too long mantissa). However, denoting by "fl" the result of a floating point calculation, results of floating-point elementary arithmetic operations can be written in the form

$$fl\,(x \pm y) = (x \pm y) \cdot (1 + \varepsilon)\,,$$

$$fl\,(x \cdot y) = (x \cdot y)\,(1 + \varepsilon)\,,$$

$$fl\,(x/y) = (x/y)\,(1 + \varepsilon)\,,$$

where $|\varepsilon| \leq eps$. This means that these errors are not larger than errors resulting from standard rounding of exact results of these operations only. It is due to the IEEE 754 standard, which requires constructions of computers which yield results of elementary arithmetic operations satisfying this accuracy conditions.

Moreover, basic elementary functions are usually also implemented in a way assuring the same accuracy requirement (e.g., $fl(\sqrt{x}) = \sqrt{x}(1 + \varepsilon),\ |\varepsilon| \leq eps$, for the square root function).

Remark. The machine epsilon eps can also be defined as a minimal positive machine floating-point number g satisfying the relation $fl(1 + g) > 1$, i.e.,

$$eps \overset{\text{df}}{=} \min\{g \in M : fl(1 + g) > 1,\ g > 0\}. \tag{1.6}$$

Therefore, eps defined in the above way is also called *the unit round* [1].

In programs calculated by computers there are series of elementary arithmetic operations and calculations of elementary functions, also the input data are generally with machine representation errors. All these errors propagate then during later operations, which may lead to quite a significant *cumulation of errors*. Therefore, the overall relative error of the result of a mathematical problem calculated numerically by an appropriate computer program is generally much larger than the rounding of the result due to finite machine accuracy of number representation. Such errors can be analyzed in two ways:

– using probabilistic methods assuming that individual errors are independent, uncorrelated random variables and calculating, e.g., a *mean error*,
– using worst-case approach, which leads to estimation of an *upper bound of the possible error*.

Example 1.1. We shall evaluate upper-bound estimate of the summation error of three numbers:

$$y = a + b + c,$$

using the algorithm (which defines a unique order of elementary arithmetic operations):

$$y = (a + b) + c.$$

We have in the floating-point arithmetic:

$$y = [(a(1+\varepsilon_a) + b(1+\varepsilon_b))(1+\varepsilon_1) + c(1+\varepsilon_c)](1+\varepsilon_2),$$

where $\varepsilon_a, \varepsilon_b, \varepsilon_c$ are representation errors of the numbers a, b and c, whereas ε_1 and ε_2 represent errors of subsequent additions; all these errors are not greater than eps (in absolute values).

We shall now perform the following algebraic manipulations:

$$\begin{aligned} y &= [(a + a\varepsilon_a + b + b\varepsilon_b)(1+\varepsilon_1) + c + c\varepsilon_c)](1+\varepsilon_2) \\ &= [a + a\varepsilon_a + b + b\varepsilon_b + a\varepsilon_1 + a\varepsilon_a\varepsilon_1 + b\varepsilon_1 + b\varepsilon_b\varepsilon_1 + c + c\varepsilon_c)](1+\varepsilon_2) \\ &= a + a\varepsilon_a + b + b\varepsilon_b + a\varepsilon_1 + a\varepsilon_a\varepsilon_1 + b\varepsilon_1 + b\varepsilon_b\varepsilon_1 + c + c\varepsilon_c + \\ &\quad + a\varepsilon_2 + a\varepsilon_a\varepsilon_2 + b\varepsilon_2 + b\varepsilon_b\varepsilon_2 + a\varepsilon_1\varepsilon_2 + a\varepsilon_a\varepsilon_1\varepsilon_2 + \\ &\qquad + b\varepsilon_1\varepsilon_2 + b\varepsilon_b\varepsilon_1\varepsilon_2 + c\varepsilon_2 + c\varepsilon_c\varepsilon_2. \end{aligned}$$

Because the absolute value of each epsilon is not greater than eps, then products of two epsilons are not greater than eps^2, and of three epsilons not greater than eps^3. Because $eps^2 \ll eps$ and $eps^3 \ll eps$, then we can omit the components containing these products in our error analysis, writing the symbol ”$\underset{1}{=}$” instead of ”=”, to denote that we leave in the sum only the components containing errors entering linearly. Proceeding in this way we obtain

$$\begin{aligned} y &\underset{1}{=} a + a\varepsilon_a + b + b\varepsilon_b + a\varepsilon_1 + +b\varepsilon_1 + c + c\varepsilon_c + a\varepsilon_2 + b\varepsilon_2 + c\varepsilon_2 \\ &= a + b + c + a(\varepsilon_a + \varepsilon_1 + \varepsilon_2) + b(\varepsilon_b + \varepsilon_1 + \varepsilon_2) + c(\varepsilon_c + \varepsilon_2) \\ &= (a+b+c)\left[1 + \frac{a(\varepsilon_a + \varepsilon_1 + \varepsilon_2)}{a+b+c} + \frac{b(\varepsilon_b + \varepsilon_1 + \varepsilon_2)}{a+b+c} + \frac{c(\varepsilon_c + \varepsilon_2)}{a+b+c}\right] \\ &= (a+b+c)[1+\delta], \end{aligned}$$

where δ represents a relative error of the result. Estimating maximal absolute value of this error we get:

$$\begin{aligned} |\delta| &= \left|\frac{a(\varepsilon_a + \varepsilon_1 + \varepsilon_2)}{a+b+c} + \frac{b(\varepsilon_b + \varepsilon_1 + \varepsilon_2)}{a+b+c} + \frac{c(\varepsilon_c + \varepsilon_2)}{a+b+c}\right| \\ &\le \left|\frac{a(\varepsilon_a + \varepsilon_1 + \varepsilon_2)}{a+b+c}\right| + \left|\frac{b(\varepsilon_b + \varepsilon_1 + \varepsilon_2)}{a+b+c}\right| + \left|\frac{c(\varepsilon_c + \varepsilon_2)}{a+b+c}\right| \\ &\le \frac{|a|3eps}{|a+b+c|} + \frac{|b|3eps}{|a+b+c|} + \frac{|c|2eps}{|a+b+c|} \\ &= \frac{(|a|+|b|)3eps + |c|2eps}{|a+b+c|} \\ &\le \frac{(|a|+|b|+|c|)}{|a+b+c|}3eps. \end{aligned}$$

The above estimate indicates, that the relative error of the addition of three numbers may be very large if the sum $a + b + c$ is small, and absolute values of individual components of the sum are large (this is possible due to different signs of the arguments). This phenomenon may occur when adding any number of arguments (starting from two), it is called *the reduction of significant digits.* □

1.3. Condition number

Any calculation problem can be theoretically described as a mapping

$$\mathbf{w} = \phi\left(\mathbf{d}\right),$$

where $\mathbf{d} = [d_1\ d_2\ \cdots\ d_n]^{\mathrm{T}} \in \mathbb{R}^n$ is a data vector, $\mathbf{w} = [w_1\, w_2\ \cdots\ w_m]^{\mathrm{T}} \in \mathbb{R}^m$ is a vector of results. In practice, the floating-point representation of the data numbers is used for calculations, i.e., $rd\left(d_i\right) = d_i\left(1 + \varepsilon_i\right)$, where $|\varepsilon_i| \leq eps$, $i = 1, ..., n$.

Definition. A problem is *ill-conditioned* if small (relative) perturbations in the data values can result in large (relative) changes in the value of the result.

A number characterizing quantitatively an increase of the relative error of the result versus the relative error of the data is called *the condition number.* To be strictly correct, this definition defines *the relative condition number* (as relative errors are being considered).

Example 1.2. Consider the following problem: to calculate the scalar product of two vectors $\mathbf{a}$ and $\mathbf{b}$, $\phi(\mathbf{a}, \mathbf{b}) = \sum\limits_{i=1}^{n} a_i b_i$, where

a) $\mathbf{a} = [1\ \ 2\ \ 3]$, $\ \mathbf{b} = [2\ \ 6\ \ -5]$,

b) $\mathbf{a} = [1.02\ \ 2.04\ \ 2.96]$, $\ \mathbf{b} = [2\ \ 6\ \ -5]$.

In the case a) the result is $w = \phi(\mathbf{a}, \mathbf{b}) = -1$. But in the case b) $w = -0.52$, i.e., the data perturbations corresponding to 2% relative errors result in the 48% relative error of the result.

It can be easily checked that replacing the vector $\mathbf{b}$ by $[2\ \ 6\ \ 5]$ leads to different results: a) $w = \phi(\mathbf{a}, \mathbf{b}) = 29$, b) $w = 29.08$, i.e., the relative error in the result is similar to the relative errors in the data. □

Conclusion: *the conditioning of a given calculation problem depends on actual values of the data (may be different for different data sets).*

Denoting an absolute error of the solution by $\Delta\mathbf{w} = [\Delta w_1\ \Delta w_2 \cdots \Delta w_m]^{\mathrm{T}}$, the corresponding relative error can be written as

$$\frac{\|\Delta\mathbf{w}\|}{\|\mathbf{w}\|} = \frac{\|\phi\left(\mathbf{d}+\Delta\mathbf{d}\right) - \phi\left(\mathbf{d}\right)\|}{\|\phi\left(\mathbf{d}\right)\|},$$

where $\Delta\mathbf{d} = [\Delta d_1\ \Delta d_2 \cdots \Delta d_n]^{\mathrm{T}}$ is the vector of the data absolute errors.

Denoting by $\frac{\|\Delta\mathbf{d}\|}{\|\mathbf{d}\|}$ the relative error of the data vector, the *condition number* $\mathrm{cond}_\phi(\mathbf{d})$ of the problem $\mathbf{w} = \phi\,(\mathbf{d})$ can be defined as follows:

$$\mathrm{cond}_\phi(\mathbf{d}) = \lim_{\delta\to 0}\ \sup_{\|\Delta\mathbf{d}\|\le\delta} \frac{\frac{\|\phi(\mathbf{d}+\Delta\mathbf{d})-\phi(\mathbf{d})\|}{\|\phi(\mathbf{d})\|}}{\frac{\|\Delta\mathbf{d}\|}{\|\mathbf{d}\|}}, \tag{1.7}$$

i.e., (1.7) defines a maximal possible increase of relative errors of the result, versus relative errors in the data, provided all possible very small (converging to zero) data perturbations are taken into account.

Definition (1.7) is not particularly convenient, but it can be transformed to a more constructive form if the problem $\phi(\mathbf{d})$ can be expressed *in the form of a known continuous and differentiable mapping*.

Assume that $\phi(\mathbf{d}) = f(\mathbf{d})$, $f : \mathbb{R}^n \to \mathbb{R}$, where the function f is differentiable and assume that the considered vector norm is Euclidean. Then we can write, for sufficiently small data variations $\triangle\mathbf{d}$:

$$\|f(\mathbf{d}+\Delta\mathbf{d}) - f(\mathbf{d})\| \approx \left\|f^{'}(\mathbf{d})\cdot\triangle\mathbf{d}\right\|,$$

where $f^{'}(\mathbf{d})$ is a row vector of partial derivatives, hence for the Euclidean norm

$$\max_{\triangle\mathbf{d}} \left\|f^{'}(\mathbf{d})\cdot\triangle\mathbf{d}\right\| = \left\|f^{'}(\mathbf{d})\right\| \|\triangle\mathbf{d}\|,$$

as the maximum is attained for co-linear vectors. Then, directly from (1.7)

$$\mathrm{cond}_f(\mathbf{d}) = \lim_{\delta\to 0}\ \sup_{\|\Delta\mathbf{d}\|\le\delta} \frac{\frac{\|f(\mathbf{d}+\Delta\mathbf{d})-f(\mathbf{d})\|}{\|f(\mathbf{d})\|}}{\frac{\|\Delta\mathbf{d}\|}{\|\mathbf{d}\|}} = \frac{\left\|f^{'}(\mathbf{d})\right\| \|\mathbf{d}\|}{\|f(\mathbf{d})\|}. \tag{1.8}$$

The above relation clearly indicates that the value of the condition number depends on actual values of the data (as stated in the Conclusion given above), i.e., for every problem $\phi(\cdot)$, $\mathrm{cond}_\phi(\mathbf{d})$ is a mapping $\mathbb{R}^n \supseteq D \ni \mathbf{d} \to \mathrm{cond}_\phi(\mathbf{d}) \in \mathbb{R}_+$, where D is a set of the data, for which the problem $\phi(\cdot)$ can be sensible formulated.

Let us emphasize that for an actual *finite* change in the data, the condition number (1.8) describes only roughly a corresponding maximal possible relative change in the result – the smaller the data perturbation vector and with the direction closer to the function gradient, the better the estimation.

Example 1.3. We shall calculate the condition number (1.8) for the problem of the scalar product of two vectors: $\phi(\mathbf{d}) = f(\mathbf{a},\mathbf{b}) = \sum_{i=1}^{n} a_i b_i$, using the Euclidean vector norm. Let us define the data vector in the form

$$\mathbf{d} = [a_1\ a_2 \cdots a_n\ b_1\ b_2 \cdots b_n]^{\mathrm{T}}.$$

Calculating the condition number according to (1.8) we have

$$\mathrm{cond}_f(\mathbf{d}) = \frac{\|[b_1\, b_2 \cdots b_n\, a_1\, a_2 \cdots a_n]\|_2\, \|\mathbf{d}\|_2}{|\sum\limits_{i=1}^{n} a_i b_i|} = \frac{\sum_{i=1}^{n}(a_i^2 + b_i^2)}{|\sum\limits_{i=1}^{n} a_i b_i|}. \tag{1.9}$$

For the data from Example 1.2 we have:

1) $\mathbf{a} = [1\ \ 2\ \ 3],\ \ \mathbf{b} = [2\ \ 6\ \ -5]$, hence $\mathrm{cond}_f(\mathbf{d}) = 79$,

2) $\mathbf{a} = [1\ \ 2\ \ 3],\ \ \mathbf{b} = [2\ \ 6\ \ +5]$, hence $\mathrm{cond}_f(\mathbf{d}) = \frac{79}{29} \approx 2.7$. □

Example 1.4. Let us consider the problem of an approximation of the derivative of the function $g(x)$, based on a backward finite difference, i.e.,

$$f(\mathbf{d}) = \frac{g(x) - g(x-h)}{h},$$

where $\mathbf{d} = [g(x)\ g(x-h)]^{\mathrm{T}}$ represents the data vector corrupted by relative errors $\varepsilon_1, \varepsilon_2$, i.e.,

$$\begin{aligned} fl(g(x)) &= g(x)(1+\varepsilon_1), \quad \varepsilon_1 \le Eps, \\ fl(g(x-h)) &= g(x-h)(1+\varepsilon_2), \quad \varepsilon_2 \le Eps, \end{aligned}$$

where $Eps \ge eps$, because $\varepsilon_1, \varepsilon_2$ represent relative errors of numerically calculated values $g(x)$ and $g(x-h)$. It will be also assumed that the step-size $h > 0$ is a power of 2, i.e., without the representation error (data error). Thus we have

$$\begin{aligned} \mathrm{cond}_f(\mathbf{d}) &= \frac{\|[1/h, -1/h]\|_2\, \|[g(x), g(x-h)]\|_2}{|g(x) - g(x-h)|/h} \\ &= \frac{\sqrt{2}\sqrt{g(x)^2 + g(x-h)^2}}{|g(x) - g(x-h)|} \\ &\approx \frac{2|g(x)|}{|g(x) - g(x-h)|}. \end{aligned} \tag{1.10}$$

Therefore, it is easily seen that the step h cannot be taken too small when numerically estimating the derivative, because a smaller step results in a decrease of the difference $|g(x) - g(x-h)|$, thus conditioning of the problem deteriorates. On the other hand, the smaller the step the better, at least theoretically, approximation of the derivative – a counteracting effect. We shall show in Section 8.1 that, for the considered problem, there is an optimal step-size, minimizing the overall error (the sum of the method error and of the numerical one). □

Example 1.5. Condition numbers of the well known functions

$$x_i : \mathbb{R}^3 \ni (a, b, c) \to \mathbb{C}, \ \ i = 1, 2,$$

calculating the roots of a quadratic polynomial $w(x) = ax^2 + bx + c$, $a \neq 0$, will be considered:

$$x_1(a,b,c) = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \tag{1.11}$$

$$x_2(a,b,c) = \frac{-b - \sqrt{b^2 - 4ac}}{2a}. \tag{1.12}$$

Polynomial coefficients constitute the data vector, $\mathbf{d} = [a\ b\ c]^{\mathrm{T}}$. The derivatives:

$$x_1'(a,b,c) = \left[\frac{2ac - b^2 + b\sqrt{b^2 - 4ac}}{2a^2\sqrt{b^2 - 4ac}} \quad \frac{b - \sqrt{b^2 - 4ac}}{2a\sqrt{b^2 - 4ac}} \quad \frac{-1}{\sqrt{b^2 - 4ac}} \right],$$

$$x_2'(a,b,c) = \left[\frac{-2ac + b^2 + b\sqrt{b^2 - 4ac}}{2a^2\sqrt{b^2 - 4ac}} \quad \frac{-b - \sqrt{b^2 - 4ac}}{2a\sqrt{b^2 - 4ac}} \quad \frac{1}{\sqrt{b^2 - 4ac}} \right].$$

Applying now (1.8) we get

$$\mathrm{cond}_{x_1}(d) = \frac{2a \left\| x_1'(d) \right\| \sqrt{a^2 + b^2 + c^2}}{|-b + \sqrt{b^2 - 4ac}|},$$

$$\mathrm{cond}_{x_2}(d) = \frac{2a \left\| x_2'(d) \right\| \sqrt{a^2 + b^2 + c^2}}{|-b - \sqrt{b^2 - 4ac}|}.$$

Analyzing the above formulae, we can observe two characteristic features:

Firstly, absolute values of all partial derivatives tend to the infinity when the expression $b^2 - 4ac$ tends to zero – therefore, the condition numbers behave in a similar way. Recall that if $b^2 - 4ac = 0$, then both roots are real and equal – the case separating situations of two different real roots and a pair of complex conjugate roots.

Secondly, the condition number of the function defining the root with a smaller absolute value significantly increases when $b^2 \gg 4ac$, leading to a relative error significantly larger than the one for the second root. We can avoid that using only one from the two considered formulae – that which defines, for actual values of a, b and c, the root with larger absolute value (that corresponding to a larger absolute value of the numerator). Then, the second root is calculated using one of the Viéte's formulae, preferably that for the sum of the roots ($x_1 + x_2 = -b/a$).

In certain applications only the root with smaller absolute value is needed. Then, instead of the procedure described above, it is more convenient to use alternative formulae for the roots of the quadratic polynomial, namely

$$x_1(a,b,c) = \frac{2c}{-b - \sqrt{b^2 - 4ac}}, \tag{1.13}$$

$$x_2(a,b,c) = \frac{2c}{-b + \sqrt{b^2 - 4ac}}, \tag{1.14}$$

which can be derived substituting the equalities (1.11) and (1.12) to the Viéte's formula for the product of the roots ($x_1 \cdot x_2 = c/a$). Looking for the root with a smaller absolute value, we use then only this from the equalities (1.13) and (1.14) which has the denominator with a larger absolute value (the function defining this root has smaller condition number). □

If the data errors result only from the machine number representation, i.e., $\mathrm{rd}(d_i) = d_i(1+\varepsilon_i)$, $\Delta d_i = d_i\varepsilon_i$, $|\varepsilon_i| \le eps$, $i = 1,\dots,n$, then for each of the most useful vector norms, i.e., $\|\cdot\|_1$, $\|\cdot\|_2$ and $\|\cdot\|_\infty$ (see Chapter 2), we have $\frac{\|\Delta\mathbf{d}\|}{\|\mathbf{d}\|} \le eps$. For instance, for the Euclidean norm

$$\frac{\|\Delta\mathbf{d}\|}{\|\mathbf{d}\|} = \frac{\sqrt{(d_1\varepsilon_1)^2 + (d_2\varepsilon_2)^2 + \cdots + (d_n\varepsilon_n)^2}}{\|\mathbf{d}\|} \le \frac{\|\mathbf{d}\| eps}{\|\mathbf{d}\|} = eps. \tag{1.15}$$

The following (approximate) inequality follows directly from (1.7):

$$\frac{\|\Delta\mathbf{w}\|}{\|\mathbf{w}\|} \lessapprox \mathrm{cond}_\phi(\mathbf{d}) \cdot \frac{\|\Delta\mathbf{d}\|}{\|\mathbf{d}\|}, \tag{1.16}$$

and the smaller the data errors the better the approximation. If the problem cannot be defined in a known function form, then it is usually difficult to evaluate the condition number directly from the definition (1.7). Then, we can take as the condition number the smallest number for which the relation

$$\frac{\|\Delta\mathbf{w}\|}{\|\mathbf{w}\|} \le \mathrm{cond}_\phi(\mathbf{d}) \cdot \frac{\|\Delta\mathbf{d}\|}{\|\mathbf{d}\|} \tag{1.17}$$

is true, for all possible and small data perturbations $\Delta\mathbf{d}$ (comp. (1.16)). If the data errors are only *the machine number representation errors*, then taking into account (1.15), the relation (1.17) can be written in the form

$$\frac{\|\Delta\mathbf{w}\|}{\|\mathbf{w}\|} \le \mathrm{cond}_\phi(\mathbf{d}) \cdot eps. \tag{1.18}$$

The presented way of reasoning will be illustrated by a problem of the scalar product calculation, for which the condition number (1.8) was evaluated in Example 1.3.

Example 1.6. Let the problem be $\phi(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{n} a_i b_i$. The data are perturbed, i.e., $\mathrm{rd}(a_i) = a_i(1+\alpha_i)$, $\mathrm{rd}(b_i) = b_i(1+\beta_i)$, where α_i and β_i are relative errors of the data representation, $|\alpha_i| \le eps$ and $|\beta_i| \le eps$. We shall evaluate an upper bound of the condition number, doing subsequent estimations (and omitting components containing products of relative errors, which is denoted by the symbol "$\underset{1}{=}$" introduced earlier in Example 1.1):

$$\begin{aligned}
\frac{\|\Delta w\|}{\|w\|} = \frac{|\Delta w|}{|w|} &= \frac{\left|\sum\limits_{i=1}^{n} a_i\left(1+\alpha_i\right)\cdot b_i\left(1+\beta_i\right) - \sum\limits_{i=1}^{n} a_i b_i\right|}{\left|\sum\limits_{i=1}^{n} a_i b_i\right|} \\
&= \frac{\left|\sum\limits_{i=1}^{n}\left(a_i b_i \alpha_i + a_i b_i \beta_i + a_i b_i \alpha_i \beta_i\right)\right|}{\left|\sum\limits_{i=1}^{n} a_i b_i\right|} \\
&\underset{1}{=} \frac{\left|\sum\limits_{i=1}^{n}\left(a_i b_i \alpha_i + a_i b_i \beta_i\right)\right|}{\left|\sum\limits_{i=1}^{n} a_i b_i\right|} \\
&\le \frac{\sum\limits_{i=1}^{n}\left|a_i b_i\left(\alpha_i+\beta_i\right)\right|}{\left|\sum\limits_{i=1}^{n} a_i b_i\right|} = \frac{\sum\limits_{i=1}^{n}\left|a_i b_i\right|\cdot\left|\left(\alpha_i+\beta_i\right)\right|}{\left|\sum\limits_{i=1}^{n} a_i b_i\right|} \\
&\le \frac{\sum\limits_{i=1}^{n}\left|a_i b_i\right|\cdot 2eps}{\left|\sum\limits_{i=1}^{n} a_i b_i\right|} = \operatorname{cond}_f^s\left(\mathbf{a},\mathbf{b}\right)\cdot eps, \qquad (1.19)
\end{aligned}$$

where

$$\operatorname{cond}_f^s\left(\mathbf{d}\right) = \operatorname{cond}_f^s\left(\mathbf{a},\mathbf{b}\right) = \frac{2\sum\limits_{i=1}^{n}\left|a_i b_i\right|}{\left|\sum\limits_{i=1}^{n} a_i b_i\right|}, \qquad (1.20)$$

and where the superscript "s" was added to make the difference with (1.9). The value *eps* in (1.20) is the upper bound of the relative data error, defined as

$$\max\{|\alpha_1|, ..., |\alpha_n|, |\beta_1|, ..., |\beta_n|\},$$

for the data vector defined as $\mathbf{d} = [\mathbf{a}^{\mathrm{T}}\ \mathbf{b}^{\mathrm{T}}]^{\mathrm{T}}$.

□

It should be emphasized that the condition number describes a maximal, relative error of the result caused by *errors in the data only*, i.e., when all mathematical operations leading from the (perturbed) data to the result are performed without errors – the precise result for the perturbed data.

1.4. The algorithm and its numerical realizations

Consider three *basic and different* definitions:

- **A mathematical problem** (for calculation) $\mathbf{w} = \phi(\mathbf{d})$,
- **An algorithm** $A(\mathbf{d})$ chosen to calculate **w**, i.e., a recipe how to calculate the value $\mathbf{w} = \phi(\mathbf{d})$ – a definition of a uniquely ordered sequence of elementary arithmetical operations leading from the data to the result,
- **A numerical realization** $fl\,(A\,(\mathbf{d}))$ **of the algorithm** $A\,(\mathbf{d})$:
  a) all numbers (constants) present in the definition of the algorithm $A\,(\mathbf{d})$ are replaced by numerical representations,
  b) calculation of all elementary arithmetical operations (including standard functions) in a sequence defined by the algorithm $A\,(\mathbf{d})$, in a floating-point arithmetic "fl", i.e., with numerical errors in all operations.

Example 1.7. Consider a mathematical problem: $\phi(a.b) = a^2 - b^2$, asssuming $\mathrm{rd}(a) = a$, $\mathrm{rd}(b) = b$, i.e., the data are without representation errors, as we are interested in the analysis of errors resulting from arithmetical operations.

Two algorithms will be considered, $A1\,(a,b)$ and $A2\,(a,b)$:

$A1\,(a,b)$: $a \cdot a - b \cdot b$,
$A2\,(a,b)$: $(a+b)\cdot(a-b)$.

Analysis of the numerical realization of the algorithm $A1(a,b)$:

$$\begin{aligned}
fl(A1(a,b) &= fl(a\cdot a - b\cdot b)\\
&= fl[fl(a\cdot a) - fl(b\cdot b)]\\
&= [a^2(1+\varepsilon_1) - b^2(1+\varepsilon_2)](1+\varepsilon_3)\\
&= [a^2 - b^2 + a^2\varepsilon_1 - b^2\varepsilon_2](1+\varepsilon_3)\\
&= a^2 - b^2 + a^2\varepsilon_1 - b^2\varepsilon_2 + (a^2-b^2)\varepsilon_3 + a^2\varepsilon_1\varepsilon_3 - b^2\varepsilon_2\varepsilon_3\\
&\underset{1}{=} a^2 - b^2 + a^2\varepsilon_1 - b^2\varepsilon_2 + (a^2-b^2)\varepsilon_3\\
&= (a^2-b^2)(1 + \frac{a^2\varepsilon_1 - b^2\varepsilon_2}{a^2-b^2} + \varepsilon_3)\\
&= (a^2-b^2)(1+\delta_1),
\end{aligned}$$

where
$$\delta_1 = \frac{a^2\varepsilon_1 - b^2\varepsilon_2}{a^2-b^2} + \varepsilon_3$$

is the relative error of the result,

$$|\delta_1| = \left|\frac{a^2\varepsilon_1 - b^2\varepsilon_2}{a^2-b^2} + \varepsilon_3\right| \le \frac{a^2+b^2}{|a^2-b^2|}eps + eps.$$

Analysis of the numerical realization of the algorithm $A2(a, b)$:

$$\begin{aligned} fl(A2(a,b) &= fl[(a+b)\cdot(a-b)] \\ &= fl[fl(a+b)\cdot fl(a-b)] \\ &= [(a+b)(1+\varepsilon_1)\cdot(a-b)(1+\varepsilon_2)](1+\varepsilon_3) \\ &= (a^2-b^2)(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3) \\ &\underset{1}{=} (a^2-b^2)(1+\varepsilon_1+\varepsilon_2+\varepsilon_3) \\ &= (a^2-b^2)(1+\delta_2), \end{aligned}$$

where

$$\delta_2 = \varepsilon_1+\varepsilon_2+\varepsilon_3$$

is the relative error of the result,

$$|\delta_2| = |\varepsilon_1+\varepsilon_2+\varepsilon_3| \le 3eps.$$

It is evident that the algorithm $A1$ is numerically inferior : for the cases when $a^2 \approx b^2$ the relative error δ_1 can become even several orders of magnitude larger than the data relative error, whereas the relative error δ_2 of the algorithm $A2$ is insensitive to the values of the data a and b. □

1.5. Numerical stability of algorithms

Consider again a mathematical problem ϕ, $\mathbf{w} = \phi(\mathbf{d})$, and the relative data error estimated by eps, $\frac{\|\Delta\mathbf{d}\|}{\|\mathbf{d}\|} \le eps$. According to the definition (1.16) of the condition number, we have then

$$\frac{\|\,\phi\,(\mathbf{d}+\Delta\mathbf{d})-\phi\,(\mathbf{d})\,\|}{\|\phi\,(\mathbf{d})\|} \le \mathrm{cond}_\phi(\mathbf{d})\cdot eps,$$

where $\mathbf{w} = \phi\,(\mathbf{d})$ is the *precise* mathematical result corresponding to accurate data values $\mathbf{d}$ and $\phi\,(\mathbf{d}+\Delta\mathbf{d})$ is the *precise* mathematical result corresponding to perturbed data values $\mathbf{d}+\Delta\mathbf{d}$ – in the considered case numerical errors in a numerical realization $fl(A(\mathbf{d}))$ of the algorithm $A(\mathbf{d})$ solving the problem $\phi\,(\mathbf{d})$ were not taken into account. Now, taking into account these errors leads to the definition of numerical stability of algorithms.

Definition. An algorithm $A(\mathbf{d})$ for calculation of a mathematical problem $\phi\,(\mathbf{d})$ is called *numerically stable*, if there exist a positive constant K_s such that for every data values $\mathbf{d}$, from a given data set of interest D, and for every sufficiently small eps (i.e., for every sufficiently strong arithmetic) the following inequality holds:

$$\frac{\|fl\left(A\left(\mathbf{d}\right)\right)-\phi\left(\mathbf{d}\right)\|}{\|\phi\left(\mathbf{d}\right)\|}\leq K_s\cdot\mathrm{cond}_{\phi}\left(\mathbf{d}\right)\cdot eps, \tag{1.21}$$

where the constant K_s is called *the stability index.*

Therefore, the algorithm is numerically stable if it guarantees the result with a bounded relative error, for any sensible data values, which is at most K_s times larger than a maximal relative error caused by any data perturbation only.

It follows from (1.21) that the total relative error of the result, calculated using a numerically stable algorithm, depends on:

– the condition number corresponding to the actual data values ($\mathrm{cond}_{\phi}(\mathbf{d})$),
– the applied floating-point arithmetic (the machine precision eps),
– the algorithm stability index (K_s), i.e., numerical quality of the algorithm.

It follows directly from the definition (1.21) that for a numerically stable algorithm the following holds:

$$\lim_{eps\to 0}\frac{\|\ fl\left(A\left(\mathbf{d}\right)\right)-\phi\left(\mathbf{d}\right)\ \|}{\|\phi\left(\mathbf{d}\right)\|}=0. \tag{1.22}$$

This relation can be found in the literature as a *definition of numerical stability.*

Comment[*]. The quantity

$$P\left(\mathbf{d},\phi\right)=\mathrm{cond}_{\phi}(\mathbf{d})\cdot eps\cdot\|\mathbf{w}\|+eps\cdot\|\mathbf{w}\| \tag{1.23}$$

is called *an inevitable error* (or *an optimal error level*), because *it is not dependent on the algorithm* used to calculate the task $\phi(\mathbf{d})$. The inevitable error consists of two terms: the first is an estimate of an absolute solution error caused by data perturbations only and the second is an estimate of the (exact) solution floating-point representation error. The inevitable error depends *on the task condition number* ($\mathrm{cond}(\mathbf{d})$) and *on the floating-point arithmetic used* (eps). It follows directly from (1.23) that the following important equality holds

$$\lim_{eps\to 0}P\left(\mathbf{d},\phi\right)=0.$$

The numerical stability of the algorithm $\mathbf{A}(\mathbf{d})$ can be now defined by the following inequality:

$$\|fl\left(A\left(\mathbf{d}\right)\right)-\phi\left(\mathbf{d}\right)\|\leq K_s\cdot P\left(\mathbf{d},\phi\right),$$

[*]Optional.

which is, of course, equivalent to the previous one. Therefore, a numerically stable algorithm calculates the result with the error at most K_s times larger than the inevitable error. □

The numerical stability of algorithms can be often proved using *the method of equivalent perturbations*. This method relies on proving (if possible) that the result of a numerical numerical calculation $fl(A(\mathbf{d}))$ is equivalent to a slightly perturbed exact solution of the task with slightly perturbed data, i.e.,

$$fl\left(A\left(\mathbf{d}\right)\right) = \phi\left(\mathbf{d} + \mathbf{\Delta d}\right) \cdot \left(1 + \eta\right), \tag{1.24}$$

where

$$\begin{aligned} \left|\Delta d_i\right| &\le k_i eps \cdot \left|d_i\right|, \\ \left|\eta_j\right| &\le k_j eps, \end{aligned}$$

where k_i and k_j are constants. Algorithms satisfying this equality are called *numerically correct. Every numerically correct algorithm is numerically stable.*

The effectiveness of an algorithm is defined as a number of elementary arithmetic operations needed to pass from the data to the result, in particular the numbers of *additions (and subtractions)* A and *multiplications (and divisions)* M are usually given. This is a more precise information than the total number of needed floating point operations only, which can also be met as a measure of the algorithm effectiveness.

Problems

1. Draw the sets of points satisfying: $||x||_1 \le 1, \ ||x||_2 \le 1, \ ||x||_\infty \le 1$, for $x \in \mathbb{R}^2$.
2. Calculate the norms: first, infinity and Frobenius, for the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & 3 \\ 2 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix}.$$

3. Propose the algorithm (i.e. the order of elementary operations) for the task

$$\phi\left(\mathbf{x}\right) = x_1 + x_2 + x_3, \qquad \text{where } x_1 > x_2 > x_3 > 0,$$

in a way leading to the best (smallest) influence of numerical errors.

4. Calculate the condition number of the function $f(a, b) = a^2 - b^2$.

5. Evaluate an error bound (i.e., a maximal absolute value of the relative error) of the result of numerical calculation (in floating-point arithmetic) of the algorithms
$$A1(a,b) = (a - a \cdot b) \cdot (a + 2 \cdot b),$$
$$A2(a,b) = a \cdot (1 - b) \cdot (a + 2 \cdot b),$$
assuming a lack of data representation errors: $\mathrm{rd}(a) = a$, $\mathrm{rd}(b) = b$.

6. Evaluate an error bound (i.e., a maximal absolute value of the relative error) of the result of numerical calculation (in floating-point arithmetic) of the algorithm
$$A(a,b) = (a + b) \cdot (a - b),$$
assuming: $\mathrm{rd}(a) = a(1 + \varepsilon_a)$, $\mathrm{rd}(b) = b(1 + \varepsilon_b)$. Compare the result with that obtained in Example 1.7, try to interpret.

7. Evaluate an error bound of the result (i.e., a maximal absolute value of the relative error) of the numerical calculation of the derivative of a function $f(x)$, approximated using the progressive divided difference:
$$\frac{f(x+h) - f(x)}{h},$$
assuming that the step-size h is a power of 2. Remark: the errors stemming from a numerical calculation of the values $f(x)$ i $f(x+h)$ cannot be neglected. These (relative) errors should be assumed in the form :
$fl(f(x)) = f(x)(1 + \varepsilon_1)$,
$fl(f(x+h)) = f(x+h)(1 + \varepsilon_2)$, $|\varepsilon_1| \le Eps, |\varepsilon_2| \le Eps$, ($Eps \ge eps$).

8. Compare numerical properties (i.e., error bounds of relative errors) and the effectiveness of the following two algorithms calculating values of the polynomial $w(\mathbf{x})$,
$$w(\mathbf{x}) = x^3 + a_1 \cdot x^2 + a_2 \cdot x + a_3,$$
A1: $x \cdot x \cdot x + a_1 \cdot x \cdot x + a_2 \cdot x + a_3$,
A2: $x \cdot [x \cdot (x + a_1) + a_2] + a_3$ (Horner's scheme).

9.[*] Evaluate upper bounds for $|E_i|$, for the expression
$$fl\left(\sum_{i=1}^{3} a_i b_i\right) = \sum_{i=1}^{3} a_i b_i (1 + E_i),$$
where $\mathrm{rd}(a_\mathrm{i}) = a_\mathrm{i}$, $\mathrm{rd}(b_\mathrm{i}) = b_\mathrm{i}$, $i = 1, 2, ..., n$.

[*]Optional.

Chapter 2

Linear equations, matrix factorizations

2.1. Norms of vectors and matrices

Recall that a function $\|\cdot\|\colon \mathbb{V} \to \mathbb{R}_+$ (where $\mathbb{V}$ is a linear space over the set $\mathbb{K}$ of real numbers $\mathbb{R}$ or complex numbers $\mathbb{C}$)) is called a norm if it satisfies the following axioms:

1. $\|\mathbf{x}\| \geq 0, \quad \|\mathbf{x}\| = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}, \quad \forall\, \mathbf{x} \in \mathbb{V},$
2. $\|\alpha\mathbf{x}\| = |\alpha|\, \|\mathbf{x}\|\,, \quad \forall\, \alpha \in \mathbb{K}, \ \forall\, \mathbf{x} \in \mathbb{V},$
3. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|, \quad \forall\, \mathbf{x}, \mathbf{y} \in \mathbb{V}.$

Vector Norms

Consider first the normed linear space $\mathbb{V} = \mathbb{R}^n$, i.e. the space of real-valued vectors. The Hölder norms of vectors are defined in the following way

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}, \qquad p = 1, 2, 3, ...$$

Most important are the following three Hölder norms:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i| \ \ - \text{ first norm,} \tag{2.1}$$

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} \ \ - \text{ Euclidean norm,} \tag{2.2}$$

$$\|\mathbf{x}\|_\infty = \max_{1\leq i\leq n} |x_i| \ \ - \text{maximum, or infinity norm.} \tag{2.3}$$

These norms are equivalent, i.e.

$$\forall \mathbf{x} \in \mathbb{R}^n \quad \exists\, \alpha, \beta \in \mathbb{R} \quad \alpha\, \|\mathbf{x}\|_a \leq \|\mathbf{x}\|_b \leq \beta\, \|\mathbf{x}\|_a\,.$$

For instance:

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_1 \leq n\, \|\mathbf{x}\|_\infty\,,$$

$$\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2 \le \sqrt{n}\,\|\mathbf{x}\|_\infty\,,$$
$$\frac{1}{\sqrt{n}}\,\|\mathbf{x}\|_1 \le \|\mathbf{x}\|_2 \le \|\mathbf{x}\|_1\,.$$

Matrix norms

Matrices can be treated as linear operators. The set of all matrices $\mathbf{A}$ of dimension $m \times n$,

$$\mathbb{R}^n \ni \mathbf{x} \to \mathbf{A}\mathbf{x} = \mathbf{y} \in \mathbb{R}^m,$$

constitutes a linear space, denoted by $\mathbb{L}\left(\mathbb{R}^n, \mathbb{R}^m\right)$, i.e., $\mathbb{V} = \mathbb{L}\left(\mathbb{R}^n, \mathbb{R}^m\right)$. The norm axioms written for elements of this space are:

1. $\|\mathbf{A}\| \ge 0, \quad \|\mathbf{A}\| = 0 \Leftrightarrow \mathbf{A} = \mathbf{0},$
2. $\|\alpha\mathbf{A}\| \le |\alpha|\,\|\mathbf{A}\|\,, \quad \forall\alpha \in \mathbb{K}, \quad \forall\mathbf{A} \in \mathbb{L}\left(\mathbb{R}^n, \mathbb{R}^m\right),$
3. $\|\mathbf{A} + \mathbf{B}\| \le \|\mathbf{A}\| + \|\mathbf{B}\|\,, \quad \forall\mathbf{A}, \mathbf{B} \in \mathbb{L}\left(\mathbb{R}^n, \mathbb{R}^m\right).$

A matrix norm is *induced* by a vector norm (then also called *an operator matrix norm*, see e.g., [1]), if

$$\|\mathbf{A}\| = \sup_{\mathbf{x}\neq 0} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} \quad \text{or, equivalently:} \quad \|\mathbf{A}\| = \sup_{\{\mathbf{x}:\|\mathbf{x}\|=1\}} \|\mathbf{A}\mathbf{x}\|\,. \tag{2.4}$$

The following inequality is valid for induced norms:

$$\|\mathbf{A}\mathbf{B}\| \le \|\mathbf{A}\|\,\|\mathbf{B}\|\,, \quad \forall\mathbf{A}, \mathbf{B} \in \mathbb{L}\left(\mathbb{R}^n, \mathbb{R}^m\right), \tag{2.5}$$

because

$$\begin{aligned}\|\mathbf{A}\mathbf{B}\| &= \sup_{\mathbf{x}\neq 0} \frac{\|\mathbf{A}\mathbf{B}\mathbf{x}\|}{\|\mathbf{x}\|} = \sup_{\mathbf{x}\neq 0} \frac{\|\mathbf{A}\mathbf{B}\mathbf{x}\|}{\|\mathbf{B}\mathbf{x}\|}\cdot\frac{\|\mathbf{B}\mathbf{x}\|}{\|\mathbf{x}\|}\\ &\le \sup_{\mathbf{y}\neq 0} \frac{\|\mathbf{A}\mathbf{y}\|}{\|\mathbf{y}\|}\cdot\sup_{\mathbf{x}\neq 0} \frac{\|\mathbf{B}\mathbf{x}\|}{\|\mathbf{x}\|} = \|\mathbf{A}\|\,\|\mathbf{B}\|\,.\end{aligned}$$

Any induced matrix norm is *compatible* with the associated vector norm, i.e.,

$$\|\mathbf{A}\mathbf{x}\| \le \|\mathbf{A}\|\,\|\mathbf{x}\| \quad \forall\mathbf{x} \in \mathbb{R} \quad \forall\mathbf{A} \in \mathbb{L}\left(\mathbb{R}^n, \mathbb{R}^m\right). \tag{2.6}$$

Most important induced (operator) matrix norms:

$$\|\mathbf{A}\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}| \quad - \quad \text{1-norm}, \tag{2.7}$$

$$\|\mathbf{A}\|_2 = \max_{\lambda\in\mathrm{sp}(\mathbf{A}^\mathrm{T}\mathbf{A})} \sqrt{\lambda} \quad - \quad \text{spectral norm (2-norm)}, \tag{2.8}$$

$$\|\mathbf{A}\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}| \quad - \quad \text{maksimum (infinity) norm}, \tag{2.9}$$

where $\mathrm{sp}(\mathbf{A}^{\mathrm{T}}\mathbf{A})$ denotes the spectrum of the matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, i.e., the set of all its eigenvalues.

The formulae for the 1-norm and maximum norm can easily be derived from the definition of induced norms. For instance, for the maximum norm:

$$\begin{aligned} \|\mathbf{Ax}\|_\infty &= \max_{1\le i\le m}\left|\sum_{j=1}^{n} a_{ij}x_j\right| \le \max_{1\le i\le m}\sum_{j=1}^{n}|a_{ij}x_j| \\ &= \max_{1\le i\le m}\sum_{j=1}^{n}|a_{ij}||x_j| \le \max_{1\le i\le m}\sum_{j=1}^{n}\left(|a_{ij}|\max_{1\le j\le n}|x_j|\right) \\ &= \left(\max_{1\le i\le m}\sum_{j=1}^{n}|a_{ij}|\right)\max_{1\le j\le n}|x_j| = \|\mathbf{A}\|_\infty\,\|\mathbf{x}\|_\infty\,. \end{aligned}$$

The *Frobenius norm,* called also the*Euclidean norm* is defines as follows:

$$\|\mathbf{A}\|_F \overset{\mathrm{df}}{=} \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}. \tag{2.10}$$

The Frobenius norm in not induced by any vector norm, (because $\|\mathbf{I}\|_F = \sqrt{n}$ and for any induced norm $\|\mathbf{I}\| = 1$). However, the Frobenius norm is compatible with the Euclidean vector norm, as it can be shown that

$$\|\mathbf{A}\|_2 \le \|\mathbf{A}\|_F \le \sqrt{\min(m,n)}\cdot\|\mathbf{A}\|_2\,,$$

thus

$$\|\mathbf{Ax}\|_2 \le \|\mathbf{A}\|_2\,\|\mathbf{x}\|_2 \le \|\mathbf{A}\|_F\,\|\mathbf{x}\|_2\,.$$

A spectral radius of a square matrix $\mathbf{A}$ is defined in the following way:

$$\mathrm{sr}(\mathbf{A}) \overset{\mathrm{df}}{=} \max_{\lambda\in\mathrm{sp}(\mathbf{A})}|\lambda|\,. \tag{2.11}$$

For every (induced) matrix norm the following inequality holds:

$$\mathrm{sr}(\mathbf{A}) \le \|\mathbf{A}\|\,, \tag{2.12}$$

because

$$\|\mathbf{A}\|\,\|\mathbf{v}\| \ge \|\mathbf{Av}\| = \|\lambda\mathbf{v}\| = |\lambda|\,\|\mathbf{v}\|\,,$$

which directly implies that

$$\forall\lambda\in\mathrm{sp}(\mathbf{A}) \quad |\lambda| \le \|\mathbf{A}\|\,.$$

2.2. Conditioning of a matrix, of a system of linear equations

Consider a system of linear equations of the form $\mathbf{Ax} = \mathbf{b}$, where:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}, \quad a_{ij}, b_i \in \mathbb{R}$$

and the matrix $\mathbf{A}$ is assumed to be nonsingular.

a) A perturbation in $\mathbf{b}$ will be first considered, i.e., $\mathbf{b} + \delta\mathbf{b} \Rightarrow \mathbf{x} + \delta\mathbf{x}$. We have

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b}, \\ \mathbf{A}(\mathbf{x} + \delta\mathbf{x}) &= (\mathbf{b} + \delta\mathbf{b}), \\ \mathbf{Ax} + \mathbf{A} \cdot \delta\mathbf{x} &= \mathbf{b} + \delta\mathbf{b}, \\ \delta\mathbf{x} &= \mathbf{A}^{-1} \cdot \delta\mathbf{b}. \end{aligned}$$

Therefore, for any operator matrix norm

$$\|\delta\mathbf{x}\| \leq \left\|\mathbf{A}^{-1}\right\| \|\delta\mathbf{b}\|.$$

On the other hand

$$\|\mathbf{b}\| = \|\mathbf{Ax}\| \leq \|\mathbf{A}\| \|\mathbf{x}\| \quad \Rightarrow \quad \|\mathbf{x}\| \geq \frac{\|\mathbf{b}\|}{\|\mathbf{A}\|}.$$

Dividing the former inequality by the last one we get

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\| \cdot \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}, \tag{2.13}$$

thus

$$\operatorname{cond}(\mathbf{A}) = \left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\|. \tag{2.14}$$

The condition number cond($\mathbf{A}$) given by the above formula is called the *condition number of the matrix* $\mathbf{A}$ – but, precisely, it is the condition number of the problem of solving a system of linear equations with a right hand side vector $\mathbf{b}$ perturbed.

Observe that for all induced Hölder norms

$$\operatorname{cond}_p(\mathbf{A}) = \left\|\mathbf{A}^{-1}\right\|_p \|\mathbf{A}\|_p \geq 1, \quad p = 1, 2, \ldots\infty,$$

because

$$1 = \|I\|_p = \left\|\mathbf{A}^{-1}\mathbf{A}\right\|_p \leq \left\|\mathbf{A}^{-1}\right\|_p \|\mathbf{A}\|_p.$$

The matrix condition number can easily become very large, e.g., it can be easily checked that for Hilbert matrices $\mathbf{H}_n = (h_{ij})$ defined by

$$h_{ij} = \frac{1}{i+j-1}, \quad i,j = 1,2,...,n,$$

we have: $\mathrm{cond}_2(\mathbf{H}_6) = 1.5 \cdot 10^7$, $\mathrm{cond}_2(\mathbf{H}_{10}) = 1.6 \cdot 10^{13}$.

b) Consider now a perturbation $\delta\mathbf{A}$ in the matrix $\mathbf{A}$, i.e., $\mathbf{A} + \delta\mathbf{A} \Rightarrow \mathbf{x} + \delta\mathbf{x}$ ($\det(\mathbf{A} + \delta\mathbf{A}) \neq 0$ is assumed, i.e., the perturbed matrix is still nonsingular),

$$(\mathbf{A} + \delta\mathbf{A})(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b}.$$

We have

$$\begin{gathered}\mathbf{A}\cdot\mathbf{x} + \mathbf{A}\cdot\delta\mathbf{x} + \delta\mathbf{A}\cdot\mathbf{x} + \delta\mathbf{A}\cdot\delta\mathbf{x} = \mathbf{b}, \\ \mathbf{A}\mathbf{x} = \mathbf{b} \quad\Rightarrow\quad \mathbf{A}\cdot\delta\mathbf{x} + \delta\mathbf{A}\cdot\mathbf{x} + \delta\mathbf{A}\cdot\delta\mathbf{x} = 0, \\ \delta\mathbf{x} = \mathbf{A}^{-1}\left(\delta\mathbf{A}\cdot\mathbf{x} + \delta\mathbf{A}\cdot\delta\mathbf{x}\right).\end{gathered}$$

Hence, we can write, for any operator matrix norm

$$\|\delta\mathbf{x}\| \le \left\|\mathbf{A}^{-1}\right\|\left(\|\delta\mathbf{A}\|\,\|\mathbf{x}\| + \|\delta\mathbf{A}\|\,\|\delta\mathbf{x}\|\right),$$

$$\left(1 - \left\|\mathbf{A}^{-1}\right\|\|\delta\mathbf{A}\|\right)\|\delta\mathbf{x}\| \le \left\|\mathbf{A}^{-1}\right\|\|\delta\mathbf{A}\|\,\|\mathbf{x}\|.$$

Thus, assuming that $\delta\mathbf{A}$ is small enough to assure$\left\|\mathbf{A}^{-1}\right\|\|\delta\mathbf{A}\| < 1$, we obtain

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \frac{\left\|\mathbf{A}^{-1}\right\|\|\mathbf{A}\|\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}}{1 - \left\|\mathbf{A}^{-1}\right\|\|\mathbf{A}\|\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}},$$

or, equivalently

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \frac{\mathrm{cond}(\mathbf{A})\cdot\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}}{1 - \mathrm{cond}(\mathbf{A})\cdot\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}}. \tag{2.15}$$

Notice, that for small perturbations of the matrix $\mathbf{A}$ (e.g., of the level of number representation errors) and for moderate values of the condition number $\mathrm{cond}(\mathbf{A})$ we have $\mathrm{cond}(\mathbf{A})\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} \ll 1$. Therefore, the inequality (2.15) can be then well approximated by $\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \mathrm{cond}(\mathbf{A})\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}$.

2.3. Gaussian elimination, LU factorization

Methods of a numerical solution of systems of linear equations can be divided into two groups:

1. *Finite methods* – the solution is obtained after a finite number of elementary arithmetical operations precisely defined by the method itself and the dimension of the problem (e.g., Gaussian elinination, LL$^{\mathrm{T}}$ factorization).
2. *Iterative methods* – starting from an initial point (an assumed initial estimate of the solution) the method improves this point in subsequent iterations. The number of iterations is unknown, it depends of an assumed solution accuracy.

2.3.1. Upper-triangular systems of linear equations

Consider a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, with *an upper triangular matrix* $\mathbf{A}$:

$$\begin{array}{ccccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1,n-1}x_{n-1} & + & a_{1n}x_n & = & b_1, \\
 & & a_{22}x_2 & + & \cdots & + & a_{2,n-1}x_{n-1} & + & a_{2n}x_n & = & b_2, \\
 & & & & \ddots & & \vdots & & \vdots & & \vdots \\
 & & & & & & a_{n-1,n-1}x_{n-1} & + & a_{n-1,n}x_n & = & b_{n-1}, \\
 & & & & & & & & a_{nn}x_n & = & b_n.
\end{array} \tag{2.16}$$

The algorithm for finding a solution of this system of equations is straightforward: we solve first the last equation for x_n , then the last but one for x_{n-1} (using the obtained value x_n), etc., this results in the *back-substitution* algorithm:

$$x_n = \frac{b_n}{a_{nn}},$$

$$x_{n-1} = \frac{(b_{n-1} - a_{n-1,n}x_n)}{a_{n-1,n-1}},$$

$$x_k = \frac{\left(b_k - \sum_{j=k+1}^{n} a_{kj}x_j\right)}{a_{kk}}, \qquad k = n-2, n-3, ..., 1. \tag{2.17}$$

If the obtained numerical solution is denoted by $\widetilde{x}$ (numerical – obtained using floating-point arithmetic), then the estimate of an equivalent perturbation $\delta\mathbf{A}$ (equivalent to numerical errors) can easily be obtained,

$$(\mathbf{A} + \delta\mathbf{A})\,\widetilde{\mathbf{x}} = \mathbf{b},$$

where

$$|\delta a_{kj}| \le (n - j + 2) \cdot eps \cdot |a_{kj}|, \quad k = 1, 2, ..., n-1, \quad j = k+1, ..., n,$$
$$|\delta a_{kk}| \le 2 \cdot eps \cdot |a_{kk}|, \quad k = 1, 2, ..., n.$$

The above estimates imply the following useful matrix norm estimate:

$$\|\delta\mathbf{A}\|_\infty \le \left(\frac{1}{2}n^2 + \frac{n}{2} + 1\right) \cdot eps \cdot a,$$

where a is the maximal (in the absolute value) element of $\mathbf{A}$.

The described back-substitution algorithm requires A$=\frac{1}{2}n^2-\frac{1}{2}n$ additions and M$=\frac{1}{2}n^2+\frac{1}{2}n$ multiplications. That is, the numbers of these operations are of order $\frac{1}{2}n^2$, which is usually briefly denoted as A$=O(\frac{1}{2}n^2)$, M$=O(\frac{1}{2}n^2)$.

2.3.2. Gaussian elimination

The Gaussian elimination algorithm consists of two phases:

1. *The Gaussian elimination phase*– the system of linear equations $\mathbf{Ax}=\mathbf{b}$ is converted to an equivalent system with an upper-triangular matrix.
2. *The back-substitution phase* – the upper-triangular system of linear equations is solved.

Gaussian elimination phase

The initial system of linear equations (the superscript ”(k)” denotes a system of equations before the k-th step of the algorithm):

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

Step 1 – a transformation zeroing all elements of the first column except the element in row 1 (i.e., elimination of the unknown x_1 from all equations except the first one). Assume $a_{11}^{(1)}\neq 0$ and define the row multipliers by

$$l_{i1}\overset{\text{df}}{=}\frac{a_{i1}^{(1)}}{a_{11}^{(1)}},\qquad i=2,3,...,n.$$

The first row $\mathbf{w}_1$ is multiplied by l_{i1} and subtracted from the i-th row $\mathbf{w}_i$, subsequently for $i=2,3,...,n$:

$$\mathbf{w}_i=\mathbf{w}_i-l_{i1}\mathbf{w}_1 \quad\Longleftrightarrow\quad \begin{array}{ll}
a_{ij}^{(2)} & =a_{ij}^{(1)}-l_{i1}a_{1j}^{(1)},\ j=1,2,...,n, \\
b_i^{(2)} & =b_i^{(1)}-l_{i1}b_1^{(1)}, \qquad\qquad i=2,3,...,n.
\end{array}$$

We obtain:

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

$$\text{i.e.,}\quad \mathbf{A}^{(2)}\mathbf{x}=\mathbf{b}^{(2)}.$$

Step 2 – a conversion of all elements in the second column to zero except those in the first and second rows, proceeding as in Step 1. The row multipliers are now:

$$l_{i2} \overset{\mathrm{df}}{=} \frac{a_{i2}^{(2)}}{a_{22}^{(2)}}, \qquad i = 3, 4, ..., n.$$

The second row $\mathbf{w}_2$ is, subsequently, multiplied by l_{i2} and subtracted from the i-th row $\mathbf{w}_i$, $i = 3, 4, ..., n$,

$$\mathbf{w}_i = \mathbf{w}_i - l_{i2}\mathbf{w}_2 \quad \Longleftrightarrow \quad \begin{aligned} a_{ij}^{(3)} &= a_{ij}^{(2)} - l_{i2}a_{2j}^{(2)}, \;\; j = 2, 3, ..., n, \\ b_i^{(3)} &= b_i^{(2)} - l_{i2}b_2^{(2)}, \qquad\qquad i = 3, 4, ..., n, \end{aligned}$$

leading to the system of equations

$$\mathbf{A}^{(3)}\mathbf{x} = \mathbf{b}^{(3)}.$$

In general, after $k-1$ **steps** the following system of equations is obtained:

$$\begin{array}{cccccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 + & \cdots & + & a_{1k}^{(1)}x_k & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)}, \\ & & a_{22}^{(2)}x_2 + & \cdots & + & a_{2k}^{(2)}x_k & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)}, \\ & & & \ddots & & \vdots & & & & \vdots & & \vdots \\ & & & & & a_{kk}^{(k)}x_k & + & \cdots & + & a_{kn}^{(k)}x_n & = & b_k^{(k)}, \\ & & & & & \vdots & & & & \vdots & & \vdots \\ & & & & & a_{nk}^{(k)}x_k & + & \cdots & + & a_{nn}^{(k)}x_n & = & b_n^{(k)}. \end{array}$$

The k-**th step** consists in elimination of the unknown x_k from equations located below the k-th one (i.e., number $k+1$, $k+2$, ..., n) – it is done subtracting from each of these equations the k-th one multiplied by

$$l_{ik} \overset{\mathrm{df}}{=} \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \qquad i = k+1, k+2, ..., n.$$

Therefore, the transformations performed in the k-th step are given by the formulae:

$$\mathbf{w}_i = \mathbf{w}_i - l_{ik}\mathbf{w}_k \quad \Longleftrightarrow \quad \begin{aligned} a_{ij}^{(k+1)} &= a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)}, \;\; j = k, k+1, ..., n, \\ b_i^{(k+1)} &= b_i^{(k)} - l_{ik}b_k^{(k)}, \qquad\quad i = k+1, k+2, ..., n. \end{aligned}$$

Finally, after $n-1$ steps an equivalent system of equations

$$\mathbf{A}^{(n)}\mathbf{x} = \mathbf{b}^{(n)}$$

is obtained, where $\mathbf{A}^{(n)}$ is an upper-triangular matrix.

2.3.3. LU matrix factorization

The Gaussian elimination leads, in fact, directly to *a triangular factorization* of a matrix $\mathbf{A}$ – to a decomposition of $\mathbf{A}$ into *an upper-triangular matrix* $\mathbf{U} = \mathbf{A}^{(n)}$ and *a lower-triangular matrix* $\mathbf{L}$. This will now be presented in detail.

At every step of the Gauss elimination a few *linear transformations* are performed, which is equivalent to a multiplication of the actual system of equations by a matrix (a linear operator). It can be easily checked that Step 1 is equivalent to the multiplication of the initial system $\mathbf{A} = \mathbf{b}$ by a nonsingular matrix $\mathbf{L}^{(1)}$,

$$\mathbf{A}^{(1)}\mathbf{x} = \mathbf{b}^{(1)} \quad \rightarrow \quad \mathbf{A}^{(2)}\mathbf{x} = \mathbf{b}^{(2)} \quad \equiv \quad \mathbf{L}^{(1)}\mathbf{A}^{(1)}\mathbf{x} = \mathbf{L}^{(1)}\mathbf{b}^{(1)}, \quad \text{where}$$

$$\mathbf{L}^{(1)} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -l_{21} & 1 & 0 & \cdots & 0 \\ -l_{31} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -l_{n1} & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

In general, Step k can be described as a multiplication by $\mathbf{L}^{(k)}$,

$$\mathbf{L}^{(k)}\mathbf{A}^{(k)} = \mathbf{L}^{(k)}\mathbf{b}^{(k)}, \tag{2.18}$$

where $\mathbf{L}^{(k)}$ differs from the unity matrix $\mathbf{I}$ only in k-th column:

$$\mathbf{L}^{(k)} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & & \vdots & \vdots & & 0 \\ & & \ddots & 0 & 0 & & \\ \vdots & & & 1 & 0 & & \vdots \\ & & & -l_{k+1,k} & 1 & & \\ & & & \vdots & & \ddots & 0 \\ 0 & 0 & \cdots & -l_{nk} & 0 & \cdots & 1 \end{bmatrix}. \tag{2.19}$$

It can also easily be checked that inverting $\mathbf{L}^{(k)}$ results only in the change of the signs of elements l_{ik}, i.e.,

$$(\mathbf{L}^{(k)})^{-1} = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \mathbf{0} \\ & & 1 & & \\ & & l_{k+1,k} & 1 & \\ & \mathbf{0} & \vdots & & \ddots \\ & & l_{nk} & & & 1 \end{bmatrix}. \tag{2.20}$$

As a result of the Gaussian elimination we finally obtain

$$\mathbf{A}^{(n)} = \mathbf{L}^{(n-1)}\mathbf{L}^{(n-2)}\cdots\mathbf{L}^{(1)}\mathbf{A}^{(1)}, \tag{2.21}$$

$$\mathbf{b}^{(n)} = \mathbf{L}^{(n-1)}\mathbf{L}^{(n-2)}\cdots\mathbf{L}^{(1)}\mathbf{b}^{(1)}. \tag{2.22}$$

Define

$$\mathbf{U} \overset{\mathrm{df}}{=} \mathbf{A}^{(n)} = [\mathbf{L}^{(n-1)}\mathbf{L}^{(n-2)}\cdots\mathbf{L}^{(1)}]\mathbf{A}^{(1)} = [\mathbf{L}^{(n-1)}\mathbf{L}^{(n-2)}\cdots\mathbf{L}^{(1)}]\mathbf{A},$$

than

$$[\mathbf{L}^{(n-1)}\mathbf{L}^{(n-2)}\cdots\mathbf{L}^{(1)}]^{-1}\cdot\mathbf{U} = \mathbf{A}.$$

Further, denoting

$$\mathbf{L} \overset{\mathrm{df}}{=} [\mathbf{L}^{(n-1)}\mathbf{L}^{(n-2)}\cdots\mathbf{L}^{(1)}]^{-1} = \left(\mathbf{L}^{(1)}\right)^{-1}\cdots\left(\mathbf{L}^{(n-1)}\right)^{-1}, \tag{2.23}$$

we get finally the *LU factorization*:

$$\mathbf{A} = \mathbf{L}\mathbf{U}.$$

It can easily be verified that

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ l_{31} & l_{32} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix}. \tag{2.24}$$

Notice that it follows directly from (2.22) that

$$\mathbf{L}\mathbf{b}^{(n)} = \mathbf{b}, \tag{2.25}$$

i.e., transformations of the vector $\mathbf{b}$ during the Gaussian elimination phase is equivalent to solution of the above lower-triangular system of equations. Number of computations for the LU factorization: A= $O(\frac{1}{3}n^3)$, M= $O(\frac{1}{3}n^3)$.

If only the LU factorization of a matrix $\mathbf{A}$ is available (we have only a original, initial value of the vector $\mathbf{b}$, we do not have transformed value $\mathbf{b}^{(n)}$), then the efficient way to solve the original system of equations $\mathbf{LUx} = \mathbf{b}$ is to solve two systems of equations with triangular matrices (a lower- and an upper-triangular):

$$\mathbf{L}\mathbf{y} = \mathbf{b}, \text{ and then} \tag{2.26}$$

$$\mathbf{U}\mathbf{x} = \mathbf{y}. \tag{2.27}$$

Thus, having obtained the matrices $\mathbf{L}$ and $\mathbf{U}$, the triangular systems (2.26) - (2.27) must be solved to obtain the solution $\mathbf{x}$. This requires M= $O(n^2)$ multiplications and A$= O(n^2)$ additions, i.e., significantly less that the numbers needed to obtain $\mathbf{L}$ and $\mathbf{U}$ (which are of order $\frac{1}{3}n^3$). Therefore, the LU factorization is especially convenient if the original system of equations is solved several times, for different values of the right-hand side vector $\mathbf{b}$ only.

Example 2.1. The following system of linear equations will be solved:

$$\begin{bmatrix} 3 & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 4 \end{bmatrix}.$$

Step 1 is as follows (the central element is given in bold):

$$\left[\begin{array}{ccc|c} \mathbf{3} & 1 & 6 & 2 \\ 2 & 1 & 3 & 7 \\ 1 & 1 & 1 & 4 \end{array}\right] \Rightarrow \begin{array}{l} l_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}} = \frac{2}{3}, \quad \mathbf{w}_2 = \mathbf{w}_2 - l_{21}\mathbf{w}_1 \\ l_{31} = \frac{a_{31}^{(1)}}{a_{11}^{(1)}} = \frac{1}{3}, \quad \mathbf{w}_3 = \mathbf{w}_3 - l_{31}\mathbf{w}_1 \end{array}$$

$$\Rightarrow \left[\begin{array}{ccc|c} 3 & 1 & 6 & 2 \\ 0 & \frac{1}{3} & -1 & \frac{17}{3} \\ 0 & \frac{2}{3} & -1 & \frac{10}{3} \end{array}\right].$$

Now a special way of presentation of the results to follow will be introduced, which, especially in computer implementations, saves the memory – the larger the number of equations n the more savings. It exploits the fact that the number of new elements of $\mathbf{L}$ emerging at every step is precisely equal to the number of elements transformed to zero. Therefore, at the beginning of Step 2, we can locate the just evaluated first elements of $\mathbf{L}$ in places of the elements of $\mathbf{A}$ just brought to zero:

$$\left[\begin{array}{c|cc|c} 3 & 1 & 6 & 2 \\ \hline \frac{2}{3} & \mathbf{\frac{1}{3}} & -1 & \frac{17}{3} \\ \frac{1}{3} & \frac{2}{3} & -1 & \frac{10}{3} \end{array}\right] \Rightarrow \quad l_{32} = \frac{a_{32}^{(2)}}{a_{22}^{(2)}} = 2, \quad \mathbf{w}_3 = \mathbf{w}_3 - l_{32}\mathbf{w}_2 \quad \Rightarrow$$

$$\left[\begin{array}{ccc|c} 3 & 1 & 6 & 2 \\ \frac{2}{3} & \frac{1}{3} & -1 & \frac{17}{3} \\ \frac{1}{3} & 2 & 1 & -8 \end{array}\right], \quad \text{thus: } L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{1}{3} & 2 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & 1 & 6 \\ 0 & \frac{1}{3} & -1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Thus, having the matrix $\mathbf{U} = \mathbf{A}^{(3)}$ and the transformed vector of the right-hand side $\mathbf{b}^{(3)} = [2\ \frac{17}{3}\ -8]^{\mathrm{T}}$, we can finally calculate the solution, solving the system of equations:

$$\begin{bmatrix} 3 & 1 & 6 \\ 0 & \frac{1}{3} & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ \frac{17}{3} \\ -8 \end{bmatrix},$$

which results in $\hat{\mathbf{x}} = [19 \ -7 \ -8]^{\mathrm{T}}$. □

2.3.4. Gaussian elimination with pivoting

It can easily happen while performing the presented Gaussian elimination algorithm, that $a_{kk}^{(k)} = 0$ for certain k and the algorithm cannot continue. This is easily avoided using the *Gaussian elimination with pivoting*, which can be partial or full.

Gaussian elimination with partial pivoting

The actual system of linear equations just before the k-th step:

$$\begin{array}{ccccccccccccc}
a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1k}^{(1)}x_k & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)}, \\
 & & & & \ddots & & \vdots & & \vdots & & \vdots & & \vdots \\
 & & & & & & a_{kk}^{(k)}x_k & + & \cdots & + & a_{kn}^{(k)}x_n & = & b_k^{(k)}, \\
 & & & & & & \vdots & & \vdots & & \vdots & & \vdots \\
 & & & & & & a_{ik}^{(k)}x_k & + & \cdots & + & a_{in}^{(k)}x_n & = & b_i^{(k)}, \\
 & & & & & & \vdots & & \vdots & & \vdots & & \vdots \\
 & & & & & & a_{nk}^{(k)}x_k & + & \cdots & + & a_{nn}^{(k)}x_n & = & b_n^{(k)}.
\end{array}$$

Now, *the central element* is first chosen, as that from elements $a_{jk}^{(k)}$ $(k \le j \le n)$ which has the biggest absolute value (it is denoted as ik-th one), i.e.

$$\left|a_{ik}^{(k)}\right| = \max_j \left\{ \left|a_{kk}^{(k)}\right|, \left|a_{k+1,k}^{(k)}\right|, \ldots, \left|a_{nk}^{(k)}\right| \right\}. \qquad (2.28)$$

Next, the i-th row (the *pivot row*) and the k-th row are interchanged, and the matrix transformation is performed as previously at Step k. As the matrix $\mathbf{A}$ is assumed to be nonsingular, then always $a_{ik}^{(k)} \neq 0$ – if all elements $a_{jk}^{(k)}$, $j = k, k+1, ..., n$ of the k-th column were zero, then $\mathbf{A}$ would be singular.

The Gauss elimination algorithm should be implemented *with pivoting at every step*, regardless of the value of $a_{kk}^{(k)}$, as *this leads to smaller numerical errors.*

The interchange of the k-th and p-th row (in the k-th step of the algorithm) is equivalent to multiplication of the actual matrix $\mathbf{A}^{(k)}$ by a certain matrix $\mathbf{P}^{(k)}$, differing from the identity matrix only in the location of two ones, namely

$$
\mathbf{P}^{(k,i)} = \begin{array}{c}\left[\begin{array}{cccccccccc}
1 & 0 & & & \cdots & & & & 0 & 0 \\
0 & \ddots & & & & & & & & 0 \\
 & & 1 & & & & & & & \\
 & & & 0 & 0 & \cdots & 0 & 1 & & \\
 & & & 0 & 1 & & & 0 & & \\
\vdots & & & \vdots & & \ddots & & \vdots & & \vdots \\
 & & & 0 & & & 1 & 0 & & \\
 & & & 1 & 0 & \cdots & 0 & 0 & & \\
 & & & & & & & & 1 & \\
0 & & & & & & & & \ddots & 0 \\
0 & 0 & & & \cdots & & & & 0 & 1
\end{array}\right]
\begin{array}{l}
\\ \\ \\ \cdots \text{ row } k \\ \\ \\ \\ \cdots \text{ row } i \\ \\ \\ \\
\end{array}
\\
\begin{array}{cc} \vdots & \vdots \\ \text{column } k & \text{column } i \end{array}
\end{array}
\tag{2.29}
$$

Multiplication of a matrix $\mathbf{A}$ by the matrix $\mathbf{P}^{(k,i)}$:

– *left-sided*, interchanges the k-th and i-th row in $\mathbf{A}$,
– *right-sided*, interchanges the k-th and i-th column in $\mathbf{A}$.

The matrix $\mathbf{P}^{(k,i)}$ has the following properties:

$$
\begin{aligned}
\det(\mathbf{P}^{(k,i)}) &= -1, \\
(\mathbf{P}^{(k,i)})^{-1} &= \mathbf{P}^{(k,i)}.
\end{aligned}
$$

The transformations of the matrix $\mathbf{A}$ by the Gaussian elimination with partial (column) pivoting lead to the following result:

$$
\mathbf{A}^{(n)} = \mathbf{L}^{(n-1)}\mathbf{P}^{(n-1)} \cdots \mathbf{L}^{(2)}\mathbf{P}^{(2)}\mathbf{L}^{(1)}\mathbf{P}^{(1)}\mathbf{A}^{(1)}. \tag{2.30}
$$

Due to the row interchanges, *the final factorization is not for the original matrix* $\mathbf{A}$, *but for the matrix* $\mathbf{PA}$, where $\mathbf{P}$ is the matrix representing all interchanges of rows, i.e.,

$$
\mathbf{LU} = \mathbf{PA}\ , \quad \text{where } \mathbf{P} = \mathbf{P}^{(n-1)}\mathbf{P}^{(n-2)} \cdots \mathbf{P}^{(1)}. \tag{2.31}
$$

We shall prove (2.31), together with the derivation of the formula describing the matrix $\mathbf{L}$. Let us assume, temporarily to shorten the description, that $n = 4$ (4×4 matrix $\mathbf{A}$). Because $\mathbf{A}^{(n)} = \mathbf{U}$ and $\mathbf{A}^{(1)} = \mathbf{A}$, then from (2.30), we have:

$$
\begin{aligned}
\mathbf{U} &= (\mathbf{L}^{(3)}\mathbf{P}^{(3)})(\mathbf{L}^{(2)}\mathbf{P}^{(2)})(\mathbf{L}^{(1)}\mathbf{P}^{(1)})\mathbf{A} \\
&= \mathbf{L}^{(3)}\mathbf{P}^{(3)}\mathbf{L}^{(2)}[\mathbf{P}^{(3)}\mathbf{P}^{(3)}]\mathbf{P}^{(2)}\mathbf{L}^{(1)}[\mathbf{P}^{(2)}\mathbf{P}^{(3)}\mathbf{P}^{(3)}\mathbf{P}^{(2)}]\mathbf{P}^{(1)}\mathbf{A},
\end{aligned}
$$

where matrices within the square brackets are, in fact, the unit matrices. Further,

$$\begin{aligned}\mathbf{U} &= \mathbf{L}^{(3)}(\mathbf{P}^{(3)}\mathbf{L}^{(2)}\mathbf{P}^{(3)})(\mathbf{P}^{(3)}\mathbf{P}^{(2)}\mathbf{L}^{(1)}\mathbf{P}^{(2)}\mathbf{P}^{(3)})(\mathbf{P}^{(3)}\mathbf{P}^{(2)}\mathbf{P}^{(1)})\mathbf{A}\\ &= \tilde{\mathbf{L}}^{(3)}(\tilde{\mathbf{L}}^{(2)})(\tilde{\mathbf{L}}^{(1)})(\mathbf{P})\mathbf{A},\end{aligned}$$

therefore

$$\mathbf{L} = (\tilde{\mathbf{L}}^{(3)}\tilde{\mathbf{L}}^{(2)}\tilde{\mathbf{L}}^{(1)})^{-1} = (\tilde{\mathbf{L}}^{(1)})^{-1}(\tilde{\mathbf{L}}^{(2)})^{-1}(\tilde{\mathbf{L}}^{(3)})^{-1}. \tag{2.32}$$

In general,

$$\tilde{\mathbf{L}}^{(k)} = \mathbf{P}^{(n-1)} \cdots \mathbf{P}^{(k+2}\mathbf{P}^{(k+1)}\mathbf{L}^{(k)}\mathbf{P}^{(k+1)}\mathbf{P}^{(k+2)} \cdots \mathbf{P}^{(n-1)}, \tag{2.33}$$

$$(\tilde{\mathbf{L}}^{(k)})^{-1} = \mathbf{P}^{(n-1)} \cdots \mathbf{P}^{(k+2)}\mathbf{P}^{(k+1)}(\mathbf{L}^{(k)})^{-1}\mathbf{P}^{(k+1)}\mathbf{P}^{(k+2)} \cdots \mathbf{P}^{(n-1)}, \tag{2.34}$$

where $\mathbf{L}^{(k)}$ is evaluated in a standard way in the actual step of the Gaussian elimination, and $\tilde{\mathbf{L}}^{(k)}$ (or $(\tilde{\mathbf{L}}^{(k)})^{-1}$) is the matrix $\mathbf{L}^{(k)}$ (or $(\mathbf{L}^{(k)})^{-1}$) with the elements *interchanged only in the k-th column*, consistently with the row interchanges in next steps of the algorithm (from (k+1)-th step until the last one). It will be illustrated below using a simple example, for $n = 3$.

Assume that in the second step (and the last one for $n = 3$) the rows 2 and 3 are interchanged, which implies the transformation of the matrix $(\mathbf{L}^{(1)})^{-1}$ evaluated in the first step, to the matrix $(\tilde{\mathbf{L}}^{(1)})^{-1}$:

$$\mathbf{P}^{(2)}(\mathbf{L}^{(1)})^{-1}\mathbf{P}^{(2)} = \begin{bmatrix} 1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix} 1 & 0 & 0\\ l_{21} & 1 & 0\\ l_{31} & 0 & 1\end{bmatrix}\begin{bmatrix} 1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & 0\\ l_{31} & 0 & 1\\ l_{21} & 1 & 0\end{bmatrix}\begin{bmatrix} 1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0\\ l_{31} & 1 & 0\\ l_{21} & 0 & 1\end{bmatrix} = (\tilde{\mathbf{L}}^{(1)})^{-1}.$$

This means that the multiplication of the matrix $(\mathbf{L}^{(1)})^{-1}$ left-sided by $\mathbf{P}^{(2)}$ (interchange of rows 2 i 3) and right-sided by $\mathbf{P}^{(2)}$ (interchange of columns 2 i 3) is equivalent to the *interchange of elements only in the first column* of the matrix $(\mathbf{L}^{(1)})^{-1}$, corresponding to the row interchange. Concluding, we have a practical rule:

> when performing the LU factorization with partial (column) pivoting, if in the k-th step the rows of $\mathbf{A}^{(k)}$ are interchanged, then identical row interchanges should be applied to the part of the matrix $\mathbf{L}$ evaluated up to this point of the algorithm (consisting of its first k–1 columns), *as if these earlier calculated parts of the rows of* $\mathbf{L}$ *were connected with the rows of the matrix* $\mathbf{A}^{(k)}$ (as it is illustrated in the Example 2.2 below).

Practical implementation of this rule is extremely simple, if the newly calculated, in the k-th step, elements l_{ik} of $\mathbf{L}$ are written into the matrix $\mathbf{A}^{(k)}$, exactly into places of the just zeroed elements $a_{ik}^{(k)}$ with the same subscripts. It is then sufficient, in all the next steps, to interchange the whole (complete) rows of the such generated matrix "$\mathbf{A}^{(k)}$", i.e., together with the elements l_{ik} calculated in the previous steps. This way of proceeding is illustrated in Example 2.2 below.

Example 2.2. The system of linear equations as in Example will be solved, but now with partial (column) pivoting.

$$\begin{bmatrix} 3 & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 4 \end{bmatrix}.$$

The first step is the same as in the Example 2.1:

$$\left[\begin{array}{ccc|c} 3 & 1 & 6 & 2 \\ 2 & 1 & 3 & 7 \\ 1 & 1 & 1 & 4 \end{array}\right] \Rightarrow \begin{array}{c} l_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}} = \frac{2}{3} \\[2ex] l_{31} = \frac{a_{31}^{(1)}}{a_{11}^{(1)}} = \frac{1}{3} \end{array} \Rightarrow \left[\begin{array}{c|cc|c} 3 & 1 & 6 & 2 \\ \hline \frac{2}{3} & \frac{1}{3} & -1 & \frac{17}{3} \\ \frac{1}{3} & \frac{2}{3} & -1 & \frac{10}{3} \end{array}\right].$$

The second step:

$$\mathbf{P}^{(2)} \Rightarrow \left[\begin{array}{c|cc|c} 3 & 1 & 6 & 2 \\ \hline \frac{1}{3} & \mathbf{\frac{2}{3}} & -1 & \frac{10}{3} \\ \frac{2}{3} & \frac{1}{3} & -1 & \frac{17}{3} \end{array}\right] \Rightarrow l_{32} = \frac{1}{2} \Rightarrow \left[\begin{array}{ccc|c} 3 & 1 & 6 & 2 \\ \frac{1}{3} & \frac{2}{3} & -1 & \frac{10}{3} \\ \frac{2}{3} & \frac{1}{2} & -\frac{1}{2} & 4 \end{array}\right].$$

Certainly, we have obtained

$$\mathbf{LU} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{2}{3} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 & 6 \\ 0 & \frac{2}{3} & -1 \\ 0 & 0 & -\frac{1}{2} \end{bmatrix} = \begin{bmatrix} 3 & 1 & 6 \\ 1 & 1 & 1 \\ 2 & 1 & 3 \end{bmatrix} = \mathbf{PA}.$$

Finally, the solution of the original system of equations is calculated as in Example 2.1, applying the backsubstitution phase. Certainly, the matrix $\mathbf{U}$ and the vector $\mathbf{b}^{(3)}$ are now different. □

Gaussian elimination with full pivoting

Every (k-th) step of the Gaussian elimination algorithm starts with a selection of the central element, but now it is the one with maximal absolute value *chosen from all elements of the $k \times k$ lower right corner submatrix of the matrix* $\mathbf{A}^{(k)}$, i.e., we compare absolute values of all elements $a_{jp}^{(k)}$ $(k \le j, p \le n)$ to select the right one (it is denoted as the (i,l)-th one below),

$$\left|a_{il}^{(k)}\right| = \max_{j,p}\left\{\left|a_{jp}^{(k)}\right|,\ \ j,p = k, k+1, ..., n\right\}. \tag{2.35}$$

Having chosen the central element, i.e., knowing its column index "l" and row index "i", the k-th and l-th columns are first interchanged – which means also *identical change in the order of components of the vector* $\mathbf{x}$, which must be remembered (stored). Next, the algorithm proceeds exactly as in the Gaussian elimination with partial column pivoting: the rows k-th and i-th are interchanged and the elements in the k-th column of the transformed matrix, located below the diagonal, are brought to zero. After all such steps we obtain finally the system of equations with an upper-triangular matrix, of the form

$$\mathbf{U} = \mathbf{A}^{(n)} = (\mathbf{L}^{(n-1)}\mathbf{P}^{(n-1)})\cdots(\mathbf{L}^{(2)}\mathbf{P}^{(2)})(\mathbf{L}^{(1)}\mathbf{P}^{(1)})\mathbf{A}^{(1)}\bar{\mathbf{P}}^{1}\bar{\mathbf{P}}^{2}\cdots\bar{\mathbf{P}}^{n-1}, \tag{2.36}$$

where $\mathbf{P}$ and $\bar{\mathbf{P}}$ are matrices describing the interchanges of rows and columns:

$$\mathbf{P} \ = \ \mathbf{P}^{(n-1)}\mathbf{P}^{(n-2)}\ldots\mathbf{P}^{(1)}, \tag{2.37}$$

$$\bar{\mathbf{P}} \ = \ \bar{\mathbf{P}}^{(1)}\bar{\mathbf{P}}^{(\mathbf{2})}\ldots\bar{\mathbf{P}}^{(n-1)}. \tag{2.38}$$

The interchange of columns does not affect the transformation of the right-hand side vector $\mathbf{b}^{(k)}$, which is transformed exactly in the same way as in the algorithm with partial column pivoting. Therefore, the form of the final transformed system of equations is almost identical as in the algorithm with partial column pivoting,

$$\mathbf{LU}\bar{\mathbf{x}} = \mathbf{Pb}, \tag{2.39}$$

only the solution vector $\bar{\mathbf{x}}$ differs from the original vector $\mathbf{x}$, in the order of its components (due to the interchanges of columns). *The final factorization is not of the original matrix* $\mathbf{A}$, *but of the matrix* $\mathbf{PA\bar{P}}$,

$$\mathbf{LU} = \mathbf{PA\bar{P}}. \tag{2.40}$$

An additional computational load in the algorithm with full pivoting, when compared to the algorithm with partial column pivoting only, is connected first of all with the computations needed to calculate and compare the absolute values of the matrix elements (at each step k^2-1 comparisons, as compared to $k-1$ comparisons, $k = n, n-1, ..., 2$). We have also column interchanges and corresponding interchanges in the order of components of the solution vector $\mathbf{x}$ ($\bar{\mathbf{P}}\mathbf{x} = \bar{\mathbf{x}}$).

Example 2.3. The algorithm with full pivoting will be applied to the system of equations considered in the two previous Examples 2.1 and 2.2 (where the algorithms without pivoting and with partial column pivoting were applied). This system of equations has the form:

$$\begin{bmatrix} 3 & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 4 \end{bmatrix}.$$

The subsequent steps of the algorithm are as follows:

$$\left[\begin{array}{ccc|c} 3 & 1 & 6 & 2 \\ 2 & 1 & 3 & 7 \\ 1 & 1 & 1 & 4 \end{array}\right] \Rightarrow \bar{\mathbf{P}}^{(1)} \Rightarrow \left[\begin{array}{ccc|c} \mathbf{6} & 1 & 3 & 2 \\ 3 & 1 & 2 & 7 \\ 1 & 1 & 1 & 4 \end{array}\right] \Rightarrow$$

$$\Rightarrow \begin{array}{c} l_{21} = \frac{3}{6} = \frac{1}{2} \\ l_{31} = \frac{1}{6} \end{array} \Rightarrow \left[\begin{array}{c|cc|c} 6 & 1 & 3 & 2 \\ \hline \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 6 \\ \frac{1}{6} & \frac{5}{6} & \frac{1}{2} & \frac{11}{3} \end{array}\right] \Rightarrow \mathbf{P}^{(2)} \Rightarrow$$

$$\Rightarrow \left[\begin{array}{c|cc|c} 6 & 1 & 3 & 2 \\ \hline \frac{1}{6} & \mathbf{\frac{5}{6}} & \frac{1}{2} & \frac{11}{3} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 6 \end{array}\right] \Rightarrow l_{32} = \frac{3}{5} \Rightarrow \left[\begin{array}{cc|c|c} 6 & 1 & 3 & 2 \\ \hline \frac{1}{6} & \frac{5}{6} & \frac{1}{2} & \frac{11}{3} \\ \hline \frac{1}{2} & \frac{3}{5} & \frac{1}{5} & \frac{19}{5} \end{array}\right].$$

Having the matrix $\mathbf{U} = \mathbf{A}^{(3)} = \mathbf{PA}\bar{\mathbf{P}}$ and the transformed vector of the right-hand side $\mathbf{b}^{(3)} = [2\ \frac{11}{3}\ \frac{19}{5}]^{\mathrm{T}}$, we can calculate the solution solving the system of equations $\mathbf{U}\bar{\mathbf{x}} = \mathbf{b}^{(3)}$, where $\bar{\mathbf{x}} = \bar{\mathbf{P}}\mathbf{x} = [x_3\ x_2\ x_1]^{\mathrm{T}}$, as below:

$$\begin{bmatrix} 6 & 1 & 3 \\ 0 & \frac{5}{6} & \frac{1}{2} \\ 0 & 0 & \frac{1}{5} \end{bmatrix} \begin{bmatrix} x_3 \\ x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 2 \\ \frac{11}{3} \\ \frac{19}{5} \end{bmatrix}.$$

The result is $\hat{\mathbf{x}} = [19\ \ -7\ \ -8]^{\mathrm{T}}$, certainly the same as in the Example 2.1. □

Numerical errors of the LU factorization

Numerical errors of the LU decomposition can be analyzed using the method of equivalent perturbations (see Chapter 1, Section 1.5), i.e. estimating an equivalent perturbation $\mathbf{E}$ of the factorized matrix $\mathbf{PA}$. For instance, for the factorization with partial column pivoting (more critical case than with full pivoting), the equivalent perturbation of the matrix $\mathbf{PA}$ is:

$$\widetilde{\mathbf{L}}\widetilde{\mathbf{U}} = \mathbf{PA} + \mathbf{E}, \tag{2.41}$$

where $\widetilde{\mathbf{L}}$ and $\widetilde{\mathbf{U}}$ denote the matrices obtained in floating-point arithmetic and, in the presented analysis, as a result of an exact factorization of the matrix **PA+E**. It can be shown that

$$\|\mathbf{E}\|_\infty \underset{1}{\leq} O(n^2) \cdot eps \cdot g_n, \tag{2.42}$$

where

$$g_n = \max_{1\leq i,j,k\leq n} \left|a_{ij}^{(k)}\right| \leq 2^{n-1} \cdot a,$$

$$a = \max_{1\leq i,j\leq n} |a_{ij}|\,.$$

The given estimate of g_n is known to be very conservative, in practice we assume

$$g_n \leq \beta_n \cdot a, \tag{2.43}$$

where β_n is a constant, in general also dependent on n, but much smaller than 2^{n-1}, e.g., when applying the full pivoting, g_n very rarely exceeds $8a$. The formulae (2.41) and (2.42) prove that the analyzed LU factorization algorithm is a *numerically correct* algorithm.

Analyzing the numerical properties of the Gaussian elimination algorithm with pivoting, for the solution of the system of equations $\mathbf{Ax} = \mathbf{b}$, it can be shown (see below) that this solution is equivalent to the exact solution with a perturbed matrix $\mathbf{A}$,

$$(\mathbf{A} + \delta\mathbf{A})\tilde{x} = \mathbf{b},$$

where

$$\frac{\|\delta\mathbf{A}\|_\infty}{\|\mathbf{A}\|_\infty} \leq O\left(n^3\right) \cdot \beta_n \cdot eps. \tag{2.44}$$

This proves the numerical correctness (thus the numerical stability) of the algorithm with partial (column) pivoting – therefore, also of the algorithm with full pivoting (as numerically less critical). The relation (2.44) shows that the equivalent error bound is proportional at least to the third power of the problem dimension n.

Error estimation[*] for a solution of a system of linear equations using the Gauss elimination – inequality (2.44). We have after the $\mathbf{LU}$ decomposition

$$\mathbf{PAx} = \mathbf{Pb},$$
$$\mathbf{LUx} = \mathbf{Pb}.$$

Two triangular systems of linear equations are solved to find the solution:

$$\mathbf{Ly} = \mathbf{Pb} \text{ and then } \mathbf{Ux} = \mathbf{y}.$$

[*]Optional.

Denoting by tilde all quantities obtained in the floating-point arithmetic ($\widetilde{\mathbf{L}}$, $\widetilde{\mathbf{U}}$, $\widetilde{\mathbf{y}}$, $\widetilde{\mathbf{x}}$, etc.), we can write:

$$\begin{aligned}\widetilde{\mathbf{L}}\widetilde{\mathbf{U}} &= \mathbf{PA} + \mathbf{E},\\ (\widetilde{\mathbf{L}}+\widetilde{\delta\mathbf{L}})\widetilde{\mathbf{y}} &= \mathbf{Pb},\\ (\widetilde{\mathbf{U}} + \widetilde{\delta\mathbf{U}})\widetilde{\mathbf{x}} &= \widetilde{\mathbf{y}},\end{aligned}$$

where $\widetilde{\delta\mathbf{L}}$ and $\widetilde{\delta\mathbf{U}}$ denote the equivalent perturbations corresponding to all numerical errors in the solutions of the triangular systems of equations. Combining the last two sets of equations we get

$$(\widetilde{\mathbf{L}}+\widetilde{\delta\mathbf{L}})(\widetilde{\mathbf{U}}+\widetilde{\delta\mathbf{U}})\widetilde{\mathbf{x}} = \mathbf{Pb},$$

which, using elementary algebraic operations, can be easily transformed to the formula

$$(\mathbf{A} + \delta\mathbf{A})\,\widetilde{\mathbf{x}} = \mathbf{b},$$

with

$$\delta\mathbf{A} = \mathbf{P}^{-1}(\mathbf{E}+\widetilde{\mathbf{L}}\cdot\widetilde{\delta\mathbf{U}}+\widetilde{\delta\mathbf{L}}\cdot\widetilde{\mathbf{U}}+\widetilde{\delta\mathbf{L}}\cdot\widetilde{\delta\mathbf{U}}).$$

Estimating the perturbation $\delta\mathbf{A}$ we have

$$\|\delta\mathbf{A}\| \le +\|\mathbf{E}\| + \|\widetilde{\mathbf{L}}\|\|\widetilde{\delta\mathbf{U}}\| + \|\widetilde{\delta\mathbf{L}}\|\|\widetilde{\mathbf{U}}\| + \|\widetilde{\delta\mathbf{L}}\|\|\widetilde{\delta\mathbf{U}}\|.$$

Due to $|l_{ij}| \le 1$ and $\widetilde{\mathbf{U}} = \mathbf{A}^{(n)}$ we get

$$\begin{aligned}\|\widetilde{\mathbf{L}}_\infty\| &\le n,\\ \|\widetilde{\mathbf{U}}\|_\infty &\le n\,g_n.\end{aligned}$$

Further, using the error estimate for a triangular system of equations we get

$$\begin{aligned}\|\widetilde{\delta\mathbf{L}}\|_\infty &\le O(\frac{1}{2}n^2)\,eps,\\ \|\delta\widetilde{\mathbf{U}}\|_\infty &\le O(\frac{1}{2}n^2)\,eps\,g_n.\end{aligned}$$

Taking into account the estimate (2.42) we obtain

$$\|\delta\mathbf{A}\|_\infty \le \left[O(n^2) + O(\frac{1}{2}n^3) + O(\frac{1}{2}n^3)\right] eps\,g_n.$$

Because $\|\mathbf{A}\|_\infty \ge a$ and assuming $g_n \le \beta_n a$ ($\beta_n \le 2^{n-1}$) we obtain finally, for a sufficiently strong arithmetic ($eps^2 \ll eps$):

$$\frac{\|\delta\mathbf{A}\|_\infty}{\|\mathbf{A}\|_\infty} \le O\left(n^3\right)\beta_n\,eps.$$

□

2.3.5. Residual correction (iterative improvement)

It can easily happen that after finding a numerical solution, say $\mathbf{x}^{(1)}$, of a system of linear equations, the solution accuracy is not satisfactory, i.e.

$$\mathbf{r}^{(1)} \overset{\text{df}}{=} \mathbf{A}\mathbf{x}^{(1)} - \mathbf{b} \neq 0,$$

and the error (called often a *residuum*) $\mathbf{r}^{(1)}$ is not acceptable – too large in absolute value. Denoting the exact solution by $\widehat{\mathbf{x}}$ we have

$$\begin{gathered}
\mathbf{x}^{(1)} = \widehat{\mathbf{x}} + \delta\mathbf{x}, \\
\mathbf{A}\left(\mathbf{x}^{(1)} - \delta\mathbf{x}\right) = \mathbf{b}, \\
\mathbf{A}\delta\mathbf{x} = \mathbf{A}\mathbf{x}^{(1)} - \mathbf{b}, \\
\mathbf{A}\delta\mathbf{x} = \mathbf{r}^{(1)}.
\end{gathered}$$

The last system of linear equations indicates a possibility to improve the solution $\mathbf{x}^{(1)}$, by solving it with respect to the correction $\delta\mathbf{x}$. The whole procedure is as follows:

1. The residuum $\mathbf{r}^{(1)} = \mathbf{A}\mathbf{x}^{(1)} - \mathbf{b}$ is calculated (prefarably with an increased precision).
2. The set $\mathbf{A}\delta\mathbf{x} = \mathbf{r}^{(1)}$ is solved, using the factorization (e.g., $\mathbf{LU}$) previously obtained while finding the first solution $\mathbf{x}^{(1)}$. In this way, the corrected solution $\mathbf{x}^{(2)}$ is obtained:
$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \delta\mathbf{x}.$$
3. The residuum $\mathbf{r}^{(2)} = \mathbf{A}\mathbf{x}^{(2)} - \mathbf{b}$ is calculated (prefarably with an increased precision). If it is smaller than $\mathbf{r}^{(1)}$ and still too large, the procedure is repeated, etc.

The calculation of the improved values $\mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \ldots$ is inexpensive when compared to the calculation of the original value $\mathbf{x}^{(1)}$, because the most expensive matrix factorization is made only once (when calculated $\mathbf{x}^{(1)}$). The presented procedure is known as *the residual correction method* or *the iterative improvement method.*

2.3.6. Full elimination method (Gauss-Jordan method)

Step 1: The procedure is precisely as in the Gaussian elimination (with partial pivoting) – all elements in the first column, with the exception of the first one, are transformed to zero.

Step 2: The second central element is chosen. Then the procedure of the transformation of the elements of the second column to zero is the same as in the Gaussian elimination – but now all the elements of the second column, with the exception of the diagonal one only, are transformed to zero, i.e., *also the element above the diagonal is transformed to zero.* We obtain

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

etc., until finally after $n-1$ steps

$$\begin{array}{cccccc}
x_1 & & & & = & b_1^{(n)}, \\
 & x_2 & & & = & b_2^{(n)}, \\
 & & \ddots & & & \vdots \\
 & & & x_n & = & b_n^{(n)}.
\end{array}$$

The number of elementary operations performed: A= $O(\frac{1}{2}n^3)$, M= $O(\frac{1}{2}n^3)$. The method can be convenient when the system of linear equations is solved only once (as the matrix factorization is not obtained), and especially when a solution of a restricted system of equations is of interest.

2.4. Cholesky-Banachiewicz (LL$^{\mathrm{T}}$) factorization

2.4.1. LL$^{\mathrm{T}}$ factorization

A symmetric matrix $\mathbf{A}$ is *positive definite*, if

$$\forall \mathbf{x} \neq 0 \quad \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0. \tag{2.45}$$

Theorem 2.1. *For any symmetric positive definite matrix* $\mathbf{A}$ *there is a unique lower-triangular matrix* $\mathbf{L}$ *with positive diagonal elements, such that*

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathrm{T}}. \tag{2.46}$$

The factorization (2.46) is called the *Cholesky* (in the Polish literature: *Cholesky-Banachiewicz) factorization*, or briefly the LL$^{\mathrm{T}}$ factorization. To derive the formulae showing how to calculate elements of the matrix $\mathbf{L}$, let us write the factorization in more detail:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & \cdot & \cdot \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdot \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdots & l_{nn} \end{bmatrix}.$$

This matrix equality can be treated as a system of linear equations (with unknown elements l_{ij}), having a special structure which leads to a simple procedure consisting of subsequent solutions of scalar equations only, each with only one unknown element. The order of the solution is by moving down along subsequent columns of the matrix **A**, starting from its diagonal, i.e. the algorithm starts from the element a_{11} in the first column:

$$\begin{aligned}
&a_{11} = l_{11}^2 \quad \Rightarrow \quad l_{11} = \sqrt{a_{11}},\\
&a_{j1} = l_{j1} l_{11} \quad \Rightarrow \quad l_{j1} = a_{j1} \,/\, l_{11}, \qquad j = 2, 3, ..., n,\\
&a_{22} = l_{21}^2 + l_{22}^2 \quad \Rightarrow \quad l_{22} = \sqrt{a_{22} - l_{21}^2},\\
&a_{j2} = l_{j1} \cdot l_{21} + l_{j2} \cdot l_{22} \quad \Rightarrow \quad l_{j2} = (a_{j2} - l_{j1} \cdot l_{21}) \,/\, l_{22}, \qquad j = 3, 4, ..., n,\\
&\quad \text{etc.}
\end{aligned}$$

In general, for $i = 1, 2, 3, ..., n$:

$$\begin{aligned}
a_{ii} &= l_{i1}^2 + l_{i2}^2 + ... + l_{ii}^2,\\
a_{ji} &= l_{j1} \cdot l_{i1} + l_{j2} \cdot l_{i2} + ... + l_{ji} \cdot l_{ii}, \qquad j = i+1, i+2, ..., n,
\end{aligned}$$

therefore, the final formulae are:

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}, \tag{2.47a}$$

$$l_{ji} = (a_{ji} - \sum_{k=1}^{i-1} l_{jk} \cdot l_{ik}) / l_{ii}, \quad i = 1, 2, ..., n, \; j = i{+}1, i{+}2, ..., n. \tag{2.47b}$$

The calculation of the matrix **L** requires A= $O(\frac{1}{6}n^3)$ additions, M= $O(\frac{1}{6}n^3)$ multiplications and n calculations of square roots.

Example 2.4. The LL$^{\mathrm{T}}$ factorization of the following matrix **A** will be calculated:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 13 & 23 \\ 4 & 23 & 77 \end{bmatrix}.$$

We have:

$$\begin{bmatrix} 1 & 2 & 4 \\ 2 & 13 & 23 \\ 4 & 23 & 77 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}.$$

Therefore, the subsequent scalar equations are:

$$\begin{aligned} 1 &= l_{11}^2 &\Rightarrow\quad & l_{11} = 1, \\ 2 &= l_{21}l_{11} &\Rightarrow\quad & l_{21} = 2 \,/\, 1 = 2, \\ 4 &= l_{31}l_{11} &\Rightarrow\quad & l_{31} = 4 \,/\, 1 = 4, \end{aligned}$$

$$\begin{aligned} 13 &= l_{21}^2 + l_{22}^2 &\Rightarrow\quad & l_{22} = \sqrt{13-4} = 3, \\ 23 &= l_{31}l_{21} + l_{32}l_{22} &\Rightarrow\quad & l_{32} = (23-8)/3 = 5, \\ 77 &= l_{31}^2 + l_{32}^2 + l_{33}^2 &\Rightarrow\quad & l_{33} = \sqrt{77-16-25} = 6, \end{aligned}$$

and the obtained matrix $\mathbf{L}$ is:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{bmatrix}.$$

□

If in the system of linear equations $\mathbf{Ax} = \mathbf{b}$ the matrix $\mathbf{A}$ satisfies the assumptions for LL^{T} factorization (symmetric, positive definite), then the factorization LL^{T}, and not LU, should be used – as twice more effective and generating less numerical errors.

Remark. Matrices **L** from LU and LL^{T} factorizations are *not the same matrices*, in spite of an identical notation "**L**", which indicates a lower-triangular matrix only. Relations between these matrices will be discussed in the next point.

2.4.2. $\mathbf{LDL^{T}}$ factorization, relations between triangular factorizations

The $\mathrm{LDL}^{\mathrm{T}}$ factorization is closely connected with the LL^{T} factorization (it is sometimes treated as a version of the LL^{T} factorization). It is as follows:

$$\mathbf{A} = \overline{\mathbf{L}}\mathbf{D}\overline{\mathbf{L}}^{\mathrm{T}}$$

$$= \begin{bmatrix} 1 & & & \mathbf{0} \\ \bar{l}_{21} & 1 & & \\ \vdots & & \ddots & \\ \bar{l}_{n1} & \bar{l}_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_{11} & & & \mathbf{0} \\ & d_{22} & & \\ & & \ddots & \\ \mathbf{0} & & & d_{nn} \end{bmatrix} \begin{bmatrix} 1 & \bar{l}_{21} & \cdots & \bar{l}_{n1} \\ & 1 & & \bar{l}_{n2} \\ & & \ddots & \\ \mathbf{0} & & & 1 \end{bmatrix}. \tag{2.48}$$

For symmetric and positive-definite (see (2.45)) matrices both factorizations exist and in the $\mathrm{LDL}^{\mathrm{T}}$ factorization we have $d_{ii} > 0,\ i = 1, ..., n$, which will be shown below. Denote $d_{ii} = l_{ii}^2$, $i = 1, ..., n$, then we can write

$$\overline{\mathbf{L}}\mathbf{D}\overline{\mathbf{L}}^{\mathrm{T}} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \bar{l}_{21} & 1 & & 0 \\ \vdots & & \ddots & \\ \bar{l}_{n1} & \bar{l}_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} l_{11}^2 & 0 & \cdots & 0 \\ 0 & l_{22}^2 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & l_{nn}^2 \end{bmatrix} \begin{bmatrix} 1 & \bar{l}_{21} & \cdots & \bar{l}_{n1} \\ 0 & 1 & & \bar{l}_{n2} \\ \vdots & & \ddots & \\ 0 & 0 & & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \bar{l}_{21} & 1 & & 0 \\ \vdots & & \ddots & \\ \bar{l}_{n1} & \bar{l}_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ 0 & l_{22} & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ 0 & l_{22} & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & \bar{l}_{21} & \cdots & \bar{l}_{n1} \\ 0 & 1 & & \bar{l}_{n2} \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$= \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdot \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdots & l_{nn} \end{bmatrix} = \mathbf{LL}^{\mathrm{T}}.$$

Therefore, we have the following transition from the LL$^{\mathrm{T}}$ to the LDL$^{\mathrm{T}}$:

$$\begin{aligned} \mathbf{D} &= \operatorname{diag}\{l_{ii}{}^2\}, \\ \overline{\mathbf{L}} &= \mathbf{L}\,[\operatorname{diag}\{l_{ii}\}]^{-1} \end{aligned} \tag{2.49}$$

and in the reversed direction

$$\mathbf{L} = \overline{\mathbf{L}} \cdot \operatorname{diag}\{\sqrt{\mathrm{d_{ii}}}\}. \tag{2.50}$$

The easiest way to derive the formulae for elements of the LDL$^{\mathrm{T}}$ factorization is to proceed in the way applied for the LL$^{\mathrm{T}}$ factorization. Multiplying the two last matrices of the LDL$^{\mathrm{T}}$ factorization we get it as a matrix equation in the form:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & \cdot & \cdot \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & & & \mathbf{0} \\ \bar{l}_{21} & 1 & & \\ \vdots & & \ddots & \\ \bar{l}_{n1} & \bar{l}_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_{11} & d_{11}\bar{l}_{21} & \cdots & d_{11}\bar{l}_{n1} \\ 0 & d_{22} & & d_{22}\bar{l}_{n2} \\ & & \ddots & \\ 0 & 0 & \cdots & d_{nn} \end{bmatrix}$$

and we solve scalar equations, subsequently along the columns of the matrix $\mathbf{A}$, starting from the diagonal elements (as it was for LL$^{\mathrm{T}}$ factorization):

$$\begin{aligned} & a_{11} = d_{11}^2 \quad \Rightarrow \quad d_{11} = a_{11}, \\ & a_{j1} = d_{11}\bar{l}_{j1} \quad \Rightarrow \quad \bar{l}_{j1} = a_{j1} \,/\, d_{11}, \qquad j = 2, 3, ..., n, \\ & a_{22} = d_{11}\bar{l}_{21}^2 + d_{22} \quad \Rightarrow \quad d_{22} = a_{22} - d_{11}\bar{l}_{21}^2, \\ & a_{j2} = \bar{l}_{j1}d_{11}\bar{l}_{21} + \bar{l}_{j2}d_{22} \quad \Rightarrow \quad l_{j2} = (a_{j2} - \bar{l}_{j1}d_{11}\bar{l}_{21}) \,/\, d_{22}, \qquad j = 3, 4, ..., n, \end{aligned}$$

etc.

This results in the following algorithm:

$$d_{ii} = a_{ii} - \sum_{k=1}^{i-1} \bar{l}_{ik}^2 d_{kk}, \tag{2.51a}$$

$$\bar{l}_{ji} = (a_{ji} - \sum_{k=1}^{i-1} \bar{l}_{jk} d_{kk} \bar{l}_{ik}) / d_{ii}, \quad i = 1, ..., n, \; j = i + 1, ..., n, \tag{2.51b}$$

which is *well defined for all symmetric nonsingular matrices*, as then $d_{ii} \neq 0$, $i = 1, ..., n$. The number of elementary operations is as for the LL$^{\mathrm{T}}$ factorization: M, A $= O(\frac{1}{6}n^3)$.

In general, the LDL$^{\mathrm{T}}$ *factorization can be obtained for any symmetric matrices* (definite, semi-definite or non-definite). For the non-definite matrices, elements d_{ii} of the diagonal matrix $\mathbf{D}$ may have any values: positive, negative or zero. Good numerical algorithms for the calculation of the LDL$^{\mathrm{T}}$ factorization apply an appropriate pivoting (interchanges of rows and also columns, to preserve the symmetry), we get then the factorization

$$\mathbf{PAP}' = \mathbf{LDL}^{\mathrm{T}},$$

where the matrix $\mathbf{P}$ describes pivot operations. A LDL$^{\mathrm{T}}$ block-factorization is an alternative, where matrix elements of dimension 1×1 or 2×2 are on the diagonal of the matrix $\mathbf{D}$. The function "`ldl`" of the MATLAB package enables the LDL$^{\mathrm{T}}$ factorization in the both mentioned versions (for any symmetric matrices).

The LDL$^{\mathrm{T}}$ factorization is used, e.g., in numerical optimization algorithms utilizing second-order derivatives of the optimized (performance) function.

The LDL$^{\mathrm{T}}$ factorization is a linking element between the LU i LL$^{\mathrm{T}}$ factorizations, being directly connected with both of them, certainly for matrices for which all factorizations exist (symmetric, positive definite). The relation between the LDL$^{\mathrm{T}}$ and LU factorizations has been just discussed, the relation between LDL$^{\mathrm{T}}$ and LU factorizations is as follows:

$$\mathbf{A} = \overline{\mathbf{L}}\mathbf{D}\overline{\mathbf{L}}^{\mathrm{T}} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \bar{l}_{21} & 1 & & 0 \\ \vdots & & \ddots & \\ \bar{l}_{n1} & \bar{l}_{n2} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & d_{nn} \end{bmatrix} \begin{bmatrix} 1 & \bar{l}_{21} & \cdots & \bar{l}_{n1} \\ 0 & 1 & & \bar{l}_{n2} \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \bar{l}_{21} & 1 & & 0 \\ \vdots & & \ddots & \\ \bar{l}_{n1} & \bar{l}_{n2} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} d_{11} & d_{11}\bar{l}_{21} & \cdots & d_{11}\bar{l}_{n1} \\ 0 & d_{22} & & d_{22}\bar{l}_{n2} \\ & & \ddots & \\ 0 & 0 & \cdots & d_{nn} \end{bmatrix} = \overline{\mathbf{L}}\mathbf{U}\,.$$

2.5. Calculation of determinants and inverse matrices

Recall that for any square matrices $\mathbf{B}$ and $\mathbf{C}$ of the same dimension we have

$$\det(\mathbf{BC}) = \det\mathbf{B} \cdot \det\mathbf{C}$$

and that the determinant of a triangular matrix is equal to the product of its diagonal elements.

To calculate *the determinant*, numerically correctly, matrix factorizations should be used:

$$\begin{aligned}
\text{using LU}: &\qquad \det \mathbf{A} = \det\left(\mathbf{P}^T\mathbf{L}\mathbf{U}\right) = \det \mathbf{P} \det \mathbf{U} = \det \mathbf{P} \prod_{i=1}^{n} u_{ii},\\
\text{using LL}^{\mathrm{T}}: &\qquad \det \mathbf{A} = \det\left(\mathbf{L}\mathbf{L}^{\mathrm{T}}\right) = (\det \mathbf{L})^2 = (\prod_{i=1}^{n} l_{ii})^2,\\
\text{using LDL}^{\mathrm{T}}: &\qquad \det \mathbf{A} = \det\left(\mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}\right) = \det \mathbf{D} = \prod_{i=1}^{n} d_{ii}.
\end{aligned}$$

To calculate *the inverse of a matrix* by a numerically correct algorithm, a triangular factorization should be used. We have two main approaches here.

In the first approach, *a direct inversion of triangular matrices* is applied, e.g.,

$$(\mathbf{L}\mathbf{U})^{-1} = \mathbf{U}^{-1} \cdot \mathbf{L}^{-1}, \tag{2.52}$$

$$\left(\mathbf{L}\mathbf{L}^{\mathrm{T}}\right)^{-1} = \left(\mathbf{L}^{\mathrm{T}}\right)^{-1} \cdot \mathbf{L}^{-1}. \tag{2.53}$$

This method is numerically effective, because for any triangular matrices:

– the inverse of an upper (lower) triangular matrix preserves the triangular structure,
– inverting a triangular matrix can be numerically effective, by solving a sequence of scalar equations only – the procedure stems directly from a matrix equality defining the inverse.

For example, for a lower triangular matrix $\mathbf{L}$, the matrix equality is ($\mathbf{Y} = \{y_{ij}\}$ denotes the inverse):

$$\mathbf{I} = \mathbf{L}\mathbf{Y} \;\Leftrightarrow\; \begin{bmatrix} 1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0\\ l_{21} & l_{22} & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} y_{11} & 0 & \cdots & 0\\ y_{21} & y_{22} & & 0\\ \vdots & \vdots & \ddots & \vdots\\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{bmatrix}.$$

The elements of the inverse matrix are calculated subsequently along the columns, starting from the first one, at each column starting from the diagonal element and moving down (analogously as it was for the LL$^{\mathrm{T}}$ factorization):

$$\begin{aligned}
&1 = l_{11}y_{11} \quad\Rightarrow\quad y_{11} = 1/l_{11},\\
&0 = l_{21}y_{11} + l_{22}y_{21} \quad\Rightarrow\quad y_{21} = (-l_{21}y_{11})\;/\;l_{22},\\
&0 = l_{31}y_{11} + l_{32}y_{21} + l_{33}y_{31} \quad\Rightarrow\quad y_{31} = -(l_{31}y_{11} + l_{32}y_{21})\;/\;l_{33},\\
&\qquad\qquad\qquad\vdots
\end{aligned}$$

$$
\begin{aligned}
&0 = l_{n1}y_{11} + \cdots + l_{nn}y_{n1} \quad \Rightarrow \quad y_{n1} = -(l_{n1}y_{11} + \cdots + l_{n,n-1}y_{n-1,1}) \;/\; l_{nn},\\
&1 = l_{22}y_{22} \quad \Rightarrow \quad y_{22} = 1/l_{22},\\
&0 = l_{32}y_{22} + l_{33}y_{32} \quad \Rightarrow \quad y_{32} = (-l_{32}y_{22}) \;/\; l_{33},\\
&\quad \text{etc.}
\end{aligned}
$$

It is left to the reader, to formulate general formulae (analogous to (2.47a) and (2.47b)), needed for an effective computer implementation.

The required number of elementary operations to calculate the inverse of a single triangular matrix: M, A $= O(\frac{1}{6}n^3)$. The number of operations to calculate the product of triangular matrices: M, A $= O(\frac{1}{3}n^3)$.

Therefore, a calculation of the inverse matrix, e.g..:

- using the LU factorization: requires M, A = $O(n^3)$ ($O(\frac{1}{3}n^3)$ for the matrix factorization and $2\cdot O(\frac{1}{6}n^3)$ for inverting two triangular matrices and $O(\frac{1}{3}n^3)$ for the matrix multiplication),
- using the LL$^\mathrm{T}$ factorization: requires M, A = $O(\frac{5}{6}n^3)$ and n square root calculations.

The second approach to the problem of the matrix inversion consists of a *solution of n systems of linear equations* with the same factorized matrix, e.g., for $\mathbf{LU} = \mathbf{PA}$, we have:

$$
\begin{aligned}
\mathbf{AA}^{-1} &= \mathbf{I},\\
\mathbf{PAA}^{-1} &= \mathbf{PI},\\
\mathbf{LUA}^{-1} &= \mathbf{PI},
\end{aligned}
$$

where $\mathbf{P}$ is a permutation matrix from the Gaussian elimination with pivoting. Denote

$$
\mathbf{A}^{-1} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{bmatrix},
$$

then

$$
\mathbf{LU} \begin{array}{c} \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{bmatrix} \\ \begin{array}{cccc} \mathbf{y}_1 & \mathbf{y}_2 & & \mathbf{y}_n \end{array} \end{array} = \mathbf{P} \begin{array}{c} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & & \cdots & 1 \end{bmatrix} \\ \begin{array}{cccc} \mathbf{e}_1 & \mathbf{e}_2 & & \mathbf{e}_n \end{array} \end{array}.
$$

This matrix equation is equivalent to n sets of standard linear equations:

$$\begin{aligned} \mathbf{LUy}_1 &= \mathbf{Pe}_1, \\ \mathbf{LUy}_2 &= \mathbf{Pe}_2, \\ \vdots &= \vdots \\ \mathbf{LUy}_n &= \mathbf{Pe}_n, \end{aligned}$$

each using the same $\mathbf{LU}$ factorization of $\mathbf{A}$. Solving the above sets of equations requires $O(2{\cdot}n{\cdot}\frac{1}{2}n^2) = O(n^3)$ additions and multiplications, not counting $O(\frac{1}{3}n^3)$ additions and multiplications needed for a (single) LU factorization.

Remark. It is wasteful to produce an inverse matrix if it is really not necessary. Comparing expenses when solving $\mathbf{Ay} = \mathbf{b}$ using the $\mathbf{LU}$ factorization and solving then two triangular sets of equations, with the procedure of calculating first the inverse $\mathbf{A}^{-1} = (\mathbf{LU})^{-1}$ and then $\mathbf{A}^{-1}\mathbf{b}$, we have:

$$\begin{array}{ccccc} & & \mathbf{Ly} = \mathbf{b} & & \mathbf{Ux} = \mathbf{y} \\ & \diagup & O\left(\frac{1}{2}n^2\right) & + & O\left(\frac{1}{2}n^2\right) \\ \mathbf{LU} & & & & \\ & \diagdown & (\mathbf{LU})^{-1} & & \mathbf{x} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{b} \\ & & O\left(n^3\right) & + & O\left(n^2\right) \end{array}$$

Moreover, the first approach generates less numerical errors.

2.6. Iterative methods for systems of linear equations

Iterative methods for solving a system of linear equations $\mathbf{Ax} = \mathbf{b}$ are used, first of all, for problems of large dimension and with a sparse matrix $\mathbf{A}$.

Consider a sequence of vectors $\mathbf{x}^{(n)}$, where $n = 0, 1, ...$ and $\mathbf{x}^{(0)}$ is a given initial point (usually, the best known approximation of the solution point), generated according to:

$$\mathbf{x}^{(i+1)} = \mathbf{Mx}^{(i)} + \mathbf{w}, \tag{2.54}$$

where $\mathbf{M}$ is a certain matrix.

Theorem 2.2. *A sequence* $\{\mathbf{x}^{(i)}\}$ *defined by* $\mathbf{x}^{(i+1)} = \mathbf{Mx}^{(i)} + \mathbf{w}$ *is, for any initial point* $\mathbf{x}^{(0)}$*, convergent to the point* $\widehat{\mathbf{x}}$ *satisfying the equality* $\mathbf{x} = \mathbf{Mx} + \mathbf{w}$*, if and only if*

$$\mathrm{sr}\,(\mathbf{M}) < 1, \tag{2.55}$$

where $\mathrm{sr}\,(\mathbf{M})$ *denotes the spectral radius of the matrix* $\mathbf{M}$.

Remark.[*] The Theorem 2.2 implies that an iterative method will work properly with any matrix $\mathbf{M}$ and vector $\mathbf{w}$ satisfying:

a) the convergence condition: $\mathrm{sr}\,(\mathbf{M}) < 1$,

b) the coincidence condition: $\widehat{\mathbf{x}} = \mathbf{M}\widehat{\mathbf{x}} + \mathbf{w}$.

Theoretically, any matrix $\mathbf{M}$ having the spectral radius less than 1 can be assumed, and then the vector **w** can be calculated from the coincidence condition:

$$\mathbf{w} = (\mathbf{I} - \mathbf{M})\,\mathbf{A}^{-1}\mathbf{b},$$

which certainly is *not sensible*, because the calculation of $\mathbf{A}^{-1}\mathbf{b}$ immediately gives the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, moreover, the matrix $\mathbf{A}$ may be large.

A reverse reasoning is more sensible: we choose a square matrix $\mathbf{N}$ assuming $\mathbf{w} = \mathbf{N}\mathbf{b}$. Then from the coincidence condition:

$$\mathbf{A}^{-1}\mathbf{b} = \mathbf{M}\mathbf{A}^{-1}\mathbf{b} + \mathbf{N}\mathbf{b},$$

which implies

$$\begin{aligned} \mathbf{I} &= \mathbf{M} + \mathbf{N}\mathbf{A}, \ \text{i.e.,} \\ \mathbf{M} &= \mathbf{I} - \mathbf{N}\mathbf{A}. \end{aligned}$$

In this way we obtain a family of iterative methods:

$$\mathbf{x}^{(i+1)} = (\mathbf{I} - \mathbf{N}\mathbf{A})\,\mathbf{x}^{(i)} + \mathbf{N}\mathbf{b}, \tag{2.56}$$

where the matrix **N** should be chosen in a way satisfying $\mathrm{sr}\,(\mathbf{I} - \mathbf{N}\mathbf{A}) < 1$. The choice of a matrix **N** is certainly not easy, it is not arbitrarily chosen in practical cases – we usually choose an iterative algorithm from among known algorithms. However, the above reasoning shows significant level of arbitrariness in the construction of the iterative methods. □

Measures of effectiveness of the iterative methods:

1. *the number of elementary arithmetic operations* needed to perform a single iteration $\mathbf{x}^{(i)} \to \mathbf{x}^{(i+1)}$, also the *compter memory requirements*,
2. *the speed of convergence,* i.e. a measure indicating how fast the error

$$\mathbf{e}^{(i)} = \mathbf{x}^{(i)} - \widehat{\mathbf{x}}$$

decreases. It should be mentioned that the spectral radius $\mathrm{sr}(\mathbf{M})$ (if known) is a certain measure of an asymptotic convergence – the smaller the radius $\mathrm{sr}(\mathbf{M})$ the better the *asymptotic convergence* of the method.

[*] Optional.

2.6.1. Jacobi's method

Decompose a matrix $\mathbf{A}$ as follows

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U},$$

where $\mathbf{L}$ is the subdiagonal matrix, $\mathbf{D}$ the diagonal matrix and $\mathbf{U}$ the matrix with entries over the diagonal. Such a decomposition is always possible, as the following example explains:

$$\underset{\mathbf{A}}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}} = \underset{\mathbf{L}}{\begin{bmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 7 & 8 & 0 \end{bmatrix}} + \underset{\mathbf{D}}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{bmatrix}} + \underset{\mathbf{U}}{\begin{bmatrix} 0 & 2 & 3 \\ 0 & 0 & 6 \\ 0 & 0 & 0 \end{bmatrix}}.$$

The system of linear equations $\mathbf{Ax} = \mathbf{b}$ can now be written as

$$\mathbf{Dx} = -(\mathbf{L} + \mathbf{U})\mathbf{x} + \mathbf{b}.$$

Assuming that diagonal entries of the matrix $\mathbf{A}$ are nonzero (i.e., the matrix $\mathbf{D}$ jest nonsingular) the following iterative method can be *intuitively* proposed:

$$\mathbf{Dx}^{(i+1)} = -\left(\mathbf{L} + \mathbf{U}\right)\mathbf{x}^{(i)} + \mathbf{b}, \qquad i = 0, 1, 2, ... \tag{2.57}$$

The method is known as the *Jacobi's method* and is often encountered in an equivalent form

$$\mathbf{x}^{(i+1)} = -\mathbf{D}^{-1}\left(\mathbf{L} + \mathbf{U}\right)\mathbf{x}^{(i)} + \mathbf{D}^{-1}\mathbf{b}, \qquad i = 0, 1, 2, ... \tag{2.58}$$

The Jacobi's method is a *parallel computational scheme*, because the matrix equation (2.58) can be written in the form of n independent scalar equations:

$$x_j^{(i+1)} = -\frac{1}{d_{jj}}\left(\sum_{k=1}^{n}(l_{jk} + u_{jk})x_k^{(i)} + b_j\right), \qquad j = 1, 2, ..., n, \tag{2.59}$$

which can be computed in parallel, totally or partially, when using a computer which enables a parallelization of the computations.

A strong diagonal dominance of the matrix $\mathbf{A}$ is the *sufficient convergence condition* for the Jacobi's method, i.e.,

1. $|a_{ii}| > \sum\limits_{j=1, j\neq i}^{n} |a_{ij}|, \quad i = 1, 2, ..., n$ – row strong dominance,
2. $|a_{jj}| > \sum\limits_{i=1, i\neq j}^{n} |a_{ij}|, \quad j = 1, 2, ..., n$ – column strong dominance.

2.6.2. Gauss-Seidel method

Decompose the matrix of a system of equations as previously, $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$. But, now we shall write the system of equations in the form

$$(\mathbf{L} + \mathbf{D})\mathbf{x} = -\mathbf{U}\mathbf{x} + \mathbf{b}.$$

Assuming again that $\mathbf{D}$ is nonsingular, the following iterative method can be now proposed

$$(\mathbf{D} + \mathbf{L})\mathbf{x}^{(i+1)} = -\mathbf{U}\mathbf{x}^{(i)} + \mathbf{b}, \qquad i = 0, 1, 2, ...$$

called the *Gauss-Seidel method.* The idea of the method relies on the fact that a single iteration of the method can be written in the form

$$\mathbf{D}\mathbf{x}^{(i+1)} = -\mathbf{L}\mathbf{x}^{(i+1)} - \mathbf{U}\mathbf{x}^{(i)} + \mathbf{b}, \qquad i = 0, 1, 2, ... \tag{2.60}$$

Due to the subdiagonal structure of the matrix $\mathbf{L}$, the fact that $\mathbf{x}^{(i+1)}$ is also present on the right-hand side of the system of equations does not harm, provided the calculation of elements of the updated vector $\mathbf{x}^{(i+1)}$ is appropriately organised. Namely, taking into account the structure of $\mathbf{D}$ and $\mathbf{L}$, we have

$$\begin{bmatrix} d_{11}x_1^{(i+1)} \\ d_{22}x_2^{(i+1)} \\ d_{33}x_3^{(i+1)} \\ \vdots \\ d_{nn}x_n^{(i+1)} \end{bmatrix} = - \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ l_{21} & 0 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 0 & & 0 \\ \vdots & \vdots & & & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1^{(i+1)} \\ x_2^{(i+1)} \\ x_3^{(i+1)} \\ \vdots \\ x_n^{(i+1)} \end{bmatrix} - \mathbf{w}^{(i)},$$

where

$$\mathbf{w}^{(i)} = \mathbf{U}\mathbf{x}^{(i)} - \mathbf{b}.$$

Therefore, the following order of calculations follows, in a natural way:

$$\begin{aligned} x_1^{(i+1)} &= -w_1^{(i)}/d_{11}, \\ x_2^{(i+1)} &= (-l_{21} \cdot x_1^{(i+1)} - w_2^{(i)})/d_{22}, \\ x_3^{(i+1)} &= (-l_{31} \cdot x_1^{(i+1)} - l_{32} \cdot x_2^{(i+1)} - w_3^{(i)})/d_{33}, \end{aligned}$$
etc.

The computational structure of the Gauss-Seidel method is called *sequential*, as the computations cannot be made in parallel, they must be performed sequentially in a well prescribed order – every subsequent scalar equation uses results of the computations of the previous equations.

The method is convergent if the matrix $\mathbf{A}$ is strongly row or column diagonally dominant (a *sufficient convergence condition*). Moreover, for symmetric matrices the method is convergent if the matrix $\mathbf{A}$ is *positive definite*. The Gauss-Seidel method is usually faster convergent than the Jacobi's method (if both are convergent, making a comparison possible).

2.6.3. Stop tests

Criteria to check when to terminate iterations of an iterative method are necessary. Practically, the following termination criteria can be proposed:

1. Checking differences between two subsequent iteration points:
$$\left\|\mathbf{x}^{(i+1)} - \mathbf{x}^{(i)}\right\| \le \delta, \tag{2.61}$$
wher δ is an assumed tolerance. However, as we are primarily interested in the solution of the system of equations with a required accuracy, then after fulfillment of the above test we can, additionally, perform (as a test of higher level, computationally more demanding):
2. Checking a norm (e.g., Euclidean) of the solution error vector:
$$\left\|\mathbf{A}\mathbf{x}^{(i+1)} - \mathbf{b}\right\| \le \delta_2, \tag{2.62}$$
where δ_2 is an assumed tolerance. If this test is not satisfied with the assumed accuracy, then the value of δ in (2.61) can be diminished and the iterations continued. Certainly, the value of δ_2 cannot be too small, as maximal attainable solution accuracy is limited by numerical errors.

Problems

1. Derive the formula describing the 1-norm of a matrix.
2. Evaluate the formula describing conditioning of a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ in the case of a simultaneous perturbations both in the right-hand side vector $\mathbf{b}$ and in the matrix $\mathbf{A}$.
3. Solve the following systems of linear equations:
$$\begin{array}{lrcrcrcrcr} \text{a)} & 3x_1 & + & x_2 & - & 2x_3 & - & x_4 & = & 3, \\ & 3x_1 & + & x_2 & + & 2x_3 & + & 3x_4 & = & -1, \\ & x_1 & + & 5x_2 & - & 4x_3 & - & x_4 & = & 3, \\ & 2x_1 & - & 2x_2 & + & 2x_3 & + & 3x_4 & = & -8, \end{array}$$
$$\begin{array}{lrcrcrcrcr} \text{b)} & x_1 & + & 2x_2 & & & - & x_4 & = & 9, \\ & 2x_1 & + & 3x_2 & - & x_3 & & & = & 9, \\ & x & & 4x_2 & + & 2x_3 & - & 5x_4 & = & 26, \\ & 5x_1 & + & 5x_2 & + & 2x_3 & - & 4x_4 & = & 32, \end{array}$$
using the Gauss elimination method with (partial) pivoting, evaluate also the LU factorization.

4. Solve the first system of linear equations from the previous problem using the Gauss-Jordan (full elimination) method.

5. Determine the LL^{T} factorization of the matrices:

$$\begin{bmatrix} 4 & 2 & 4 \\ 2 & 5 & 10 \\ 4 & 10 & 84 \end{bmatrix}, \quad \begin{bmatrix} 4 & 2 & 8 & 4 \\ 2 & 17 & 16 & 6 \\ 8 & 16 & 29 & 21 \\ 4 & 6 & 21 & 39 \end{bmatrix}.$$

6. Evaluate $\mathrm{LDL}^{\mathrm{T}}$ factorization for matrices from previous problem, using the algorithm given in Section 2.4.2.

7. Prove that the inverse of a lower (upper) triangular matrix is also a lower (upper) triangular matrix.

8. Formulate general (for any dimension n of a matrix) formulae describing elements of the inverse of a lower triangular matrix, write these formulae in MATLAB's package language.

9. Evaluate the inverse of the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & 3 \\ 2 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix},$$

applying different algorithms using the LU factorization:
– inverting triangular matrices from the LU factorization,
– solving systems of linear equations.

10. Using MATLAB language, write a general program for solving a system of linear equations of any (n) dimension, using the Gaussian elimination with (partial) pivoting. Using this program, solve the following systems of linear equations:

a)
$$\begin{aligned}
x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 + \frac{1}{4}x_4 + \frac{1}{5}x_5 &= 1, \\
\frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 + \frac{1}{5}x_4 + \frac{1}{6}x_5 &= 0, \\
\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 + \frac{1}{6}x_4 + \frac{1}{7}x_5 &= 0, \\
\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 + \frac{1}{7}x_4 + \frac{1}{8}x_5 &= 0, \\
\frac{1}{5}x_1 + \frac{1}{6}x_2 + \frac{1}{7}x_3 + \frac{1}{8}x_4 + \frac{1}{9}x_5 &= 0,
\end{aligned}$$

b)
$$\begin{aligned}
x_1 + 0.5x_2 + 0.33333x_3 + 0.25x_4 + 0.2x_5 &= 1,\\
0.5x_1 + 0.33333x_2 + 0.25x_3 + 0.2x_4 + 0.16667x_5 &= 0,\\
0.33333x_1 + 0.25x_2 + 0.2x_3 + 0.16667x_4 + 0.14286x_5 &= 0,\\
0.25x_1 + 0.2x_2 + 0.16667x_3 + 0.14286x_4 + 0.125x_5 &= 0,\\
0.2x_1 + 0.16667x_2 + 0.14286x_3 + 0.125x_4 + 0.11111x_5 &= 0.
\end{aligned}$$

compare the results, check the condition number of the matrix.

11. Using MATLAB language, write a general program for solving a system of linear equations of any (n) dimension, using the Gaussian elimination with (partial) pivoting. Using this program, solve numerically the systems of linear equations $\mathbf{Hx} = \mathbf{b}$, where $\mathbf{H} = \{h_{ij}\}$ is a Hilbert matrix,

$$h_{ij} = \frac{1}{i+j-1}, \; i, j = 1, ..., n,$$

and $\mathbf{b} = [1\,1\,\cdots\,1]^{\mathrm{T}}$, subsequently for n = 5, 10, 20, 30. Evaluate and compare solution errors.

12.[*] Determine a matrix **M** and a vector **w** of the linear iterative scheme (2.54), corresponding to the Jacobi's and Gaussa-Seidel methods. Determine the corresponding matrices **N** of the linear iterative scheme (2.56).

[*]Optional.

Chapter 3

QR factorization, eigenvalues, singular values

3.1. Orthogonal-triangular (QR) matrix factorizations

Two (or more) vectors are called *orthonormal*, if they are mutually orthogonal and each is of unity length (the Euclidean vector norm will be used throughout this section, unless otherwise explicitly stated).

If all columns of the matrix $\mathbf{Q} \in \mathbb{R}^{m\times n}$ $(m \geq n)$ are mutually orthogonal, then

$$\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{D}, \tag{3.1}$$

where $\mathbf{D} \in \mathbb{R}^{n\times n}$ is a diagonal matrix. If all columns of the matrix $\mathbf{Q}$ are, additionally, orthonormal, then $\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}$ (but in general, for $m > n$, $\mathbf{Q}\mathbf{Q}^{\mathrm{T}} \neq \mathbf{I}$).

A square matrix $\mathbf{Q}$ is called an *orthogonal matrix*, if

$$\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}, \tag{3.2}$$

i.e., it is a square matrix with all columns being mutually orthonormal vectors. Reasoning strictly, the above definition describes the orthonormal matrix, but it is common in the literature to use the description ”orthogonal” for orthonormal matrices, *defining the orthogonal matrix by* (3.2).

It follows directly from the definitio (3.2), that

$$\mathbf{Q}^{\mathrm{T}} = \mathbf{Q}^{-1}, \tag{3.3}$$

therefore

$$\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{Q}\mathbf{Q}^{\mathrm{T}} = \mathbf{I}. \tag{3.4}$$

This equality shows that an orthogonal matrix is a square matrix with all columns being mutually orthonormal, and with all rows being mutually orthonormal.

Lemma 3.1. *If* $\mathbf{Q} \in \mathbb{R}^{n\times n}$ *is a matrix with all columns being mutually orthonormal vectors, then*

$$\|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2\,. \tag{3.5}$$

Proof. From definition of the Euclidean norm and from $\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}$, we have

$$\forall \mathbf{x} \quad \|\mathbf{Q}\mathbf{x}\|_2 = \sqrt{(\mathbf{Q}\mathbf{x})^{\mathrm{T}}\mathbf{Q}\mathbf{x}} = \sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{Q}\mathbf{x}} = \sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{x}} = \|\mathbf{x}\|_2\,.$$

Corollary. For any orthogonal matrix, the following holds:

$$\|\mathbf{Qx}\|_2 = \|\mathbf{x}\|_2, \quad \|\mathbf{Q}^{\mathrm{T}}\mathbf{x}\|_2 = \|\mathbf{x}\|_2. \tag{3.6}$$

Theorem 3.1. *Every matrix $\mathbf{A}_{m\times n}$ ($\mathbf{A} \in \mathbb{R}^{m\times n}$, $m \geq n$) with all columns linearly independent (i.e., of full rank n) can be presented in the form of the following orthogonal-triangular factorizations:*

1. $$\mathbf{A}_{m\times n} = \bar{\mathbf{Q}}_{m\times n}\bar{\mathbf{R}}_{n\times n}, \tag{3.7}$$
   where $\bar{\mathbf{Q}}_{m\times n}$ is a matrix with orthogonal (not orthonormal) columns, and $\bar{\mathbf{R}}_{n\times n}$ is an upper triangular matrix with unity elements on the diagonal;
2. $$\mathbf{A}_{m\times n} = \mathbf{Q}_{m\times n}\mathbf{R}_{n\times n}, \tag{3.8}$$
   where $\mathbf{Q}_{m\times n}$ is a matrix with orthonormal columns, and $\mathbf{R}_{n\times n}$ is an upper triangular matrix with all positive elements on the diagonal;
3. $$\mathbf{A}_{m\times n} = \mathbf{Q}_{m\times m}\begin{bmatrix}\mathbf{R}_{n\times n}\\ \mathbf{0}\end{bmatrix}_{m\times n} = \begin{bmatrix}\mathbf{Q}^1_{m\times n} & \mathbf{Q}^2_{m\times(m-n)}\end{bmatrix}\begin{bmatrix}\mathbf{R}_{n\times n}\\ \mathbf{0}\end{bmatrix}_{m\times n}, \tag{3.9}$$
   where $\mathbf{Q}_{m\times m}$ is an orthogonal matrix.

Proof – by construction of the factorizations:
Ad 1. Let $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2\ \cdots\ \mathbf{a}_n]$. The columns $\mathbf{a}_i$ of the matrix $\mathbf{A}$ will be made orthogonal using the standard Gram-Schmidt algorithm; denoting by $\bar{\mathbf{q}}_i$ the vectors after orthogonalization, we have:

$$\begin{aligned}
\bar{\mathbf{q}}_1 &= \mathbf{a}_1, \ \bar{r}_{11} \overset{\mathrm{df}}{=} 1,\\
\bar{\mathbf{q}}_2 &= \mathbf{a}_2 - \frac{\bar{\mathbf{q}}_1^{\mathrm{T}}\mathbf{a}_2}{\bar{\mathbf{q}}_1^{\mathrm{T}}\bar{\mathbf{q}}_1}\bar{\mathbf{q}}_1 = \mathbf{a}_2 - \bar{r}_{12}\bar{\mathbf{q}}_1, \ \bar{r}_{22} \overset{\mathrm{df}}{=} 1,\\
\bar{\mathbf{q}}_i &= \mathbf{a}_i - \sum_{j=1}^{i-1}\frac{\bar{\mathbf{q}}_j^{\mathrm{T}}\mathbf{a}_i}{\bar{\mathbf{q}}_j^{\mathrm{T}}\bar{\mathbf{q}}_j}\bar{\mathbf{q}}_j = \mathbf{a}_i - \sum_{j=1}^{i-1}\bar{r}_{ji}\bar{\mathbf{q}}_j, \ \bar{r}_{ii} \overset{\mathrm{df}}{=} 1, \ i = 3, ..., n.
\end{aligned} \tag{3.10}$$

The above equations can be rewritten in the form:

$$\begin{aligned}
\mathbf{a}_1 &= \bar{r}_{11}\bar{\mathbf{q}}_1, \ \bar{r}_{11} \overset{\mathrm{df}}{=} 1,\\
\mathbf{a}_2 &= \bar{r}_{12}\bar{\mathbf{q}}_1 + \bar{r}_{22}\bar{\mathbf{q}}_2 = \sum_{j=1}^{2}\bar{r}_{j2}\bar{\mathbf{q}}_j, \quad \bar{r}_{12} = \frac{\bar{\mathbf{q}}_1^{\mathrm{T}}\mathbf{a}_2}{\bar{\mathbf{q}}_1^{\mathrm{T}}\bar{\mathbf{q}}_1}, \ \bar{r}_{22} \overset{\mathrm{df}}{=} 1,\\
\mathbf{a}_i &= \sum_{j=1}^{i-1}\bar{r}_{ji}\bar{\mathbf{q}}_j + \bar{r}_{ii}\bar{\mathbf{q}}_i = \sum_{j=1}^{i}\bar{r}_{ji}\bar{\mathbf{q}}_j, \ \bar{r}_{ji} = \frac{\bar{\mathbf{q}}_j^{\mathrm{T}}\mathbf{a}_i}{\bar{\mathbf{q}}_j^{\mathrm{T}}\bar{\mathbf{q}}_j}, \ \bar{r}_{ii} \overset{\mathrm{df}}{=} 1, \ i=3, ..., n.
\end{aligned} \tag{3.11}$$

We shall apply now a known property of the product of two matrices:

> *i-th column* $\mathbf{a}_i$ *of the product matrix* $\mathbf{A} = \mathbf{BC}$ *is a linear combination of columns of the first matrix* $\mathbf{B}$*, with coefficients being elements of the i-th column of the second matrix* $\mathbf{C}$*, i.e.,*

$$\mathbf{a}_i = \mathbf{Bc_i} = [\mathbf{b}_1\ \mathbf{b}_2 \cdots \mathbf{b}_n] \begin{bmatrix} c_{1i} \\ c_{2i} \\ \vdots \\ c_{ni} \end{bmatrix} = \sum_{j=1}^{n} \mathbf{b}_j c_{ji}.$$

Using this general property of the matrix product we can write:

$$\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2 \cdots \mathbf{a}_n] = [\bar{\mathbf{q}}_1\ \bar{\mathbf{q}}_2 \cdots \bar{\mathbf{q}}_n] \begin{bmatrix} 1 & \bar{r}_{12} & \dots & \bar{r}_{1n} \\ 0 & 1 & \dots & \bar{r}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} = \bar{\mathbf{Q}}\bar{\mathbf{R}}, \quad (3.12)$$

where $\bar{\mathbf{Q}} = \bar{\mathbf{Q}}_{m\times n} = [\bar{\mathbf{q}}_1\ \bar{\mathbf{q}}_2 \cdots \bar{\mathbf{q}}_n]$ has orthogonal (not orthonormal) columns, and $\bar{\mathbf{R}}_{n\times n}$ is upper triangular with unity elements on the diagonal. This concludes the proof of the first part of the Theorem.

2. If we normalize the columns of $\bar{\mathbf{Q}}$,

$$\mathbf{Q} = \left[\frac{\bar{\mathbf{q}}_1}{\|\bar{\mathbf{q}}_1\|}, \frac{\bar{\mathbf{q}}_2}{\|\bar{\mathbf{q}}_2\|}, \dots, \frac{\bar{\mathbf{q}}_n}{\|\bar{\mathbf{q}}_n\|}\right], \quad (3.13)$$

and then perform the following linear transformation:

$$\mathbf{R} = \mathbf{N}\bar{\mathbf{R}}, \quad (3.14)$$

where

$$\mathbf{N} = \begin{bmatrix} \|\bar{\mathbf{q}}_1\| & 0 & \dots & 0 \\ 0 & \|\bar{\mathbf{q}}_2\| & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \|\bar{\mathbf{q}}_n\| \end{bmatrix}, \quad (3.15)$$

then the factorization (3.8) will be obtained.

3. If we add $m-n$ columns to the matrix $\mathbf{Q}_{m\times n}$ to make it a square and orthogonal matrix $\mathbf{Q}_{m\times m}$ and additional rows containing only zeros to the matrix $\mathbf{R}_{n\times n}$ to make it of dimension $m \times n$, then the factorization (3.9) will result. The columns added to $\mathbf{Q}_{m\times n}$ can be any columns orthogonal mutually and with columns of $\mathbf{Q}_{m\times n}$ (i.e., we add a matrix $\mathbf{Q}^2_{m\times(m-n)}$ from (3.9), having the stated orthogonality properties). $\square$

A simple way to augment $\mathbf{Q}_{m\times n}$: add any $m-n$ new columns to $\mathbf{A}_{m\times n}$ such that all columns of the resulting square matrix $\mathbf{A}_{m\times m}$ are linearly independent, then continue (add) the orthogonalization of these columns using the same Gram-Schmidt algorithm and normalize the obtained additional columns.

The Theorem 3.1 is generally true for any $\mathbf{A}_{m\times n}$ matrix, *i.e., also for rank-defficient matrices (with $k<n$).* Thus, QR factorization exists for any matrix, for rank-defficient matrices it can be obtained appropriately augmenting the basic Gram-Schmidt algorithm, or using algorithms based on Householder's reflections or Givens rotations.

We have three QR factorizations stated in the Theorem 3.1. The most popular is the factorization (3.9) with an orthogonal matrix $\mathbf{Q}$, commonly described as *the QR factorization.* The factorization (3.8) is called the *thin, or economy-size (MATLAB) QR factorization*, in the MATLAB procedure it is an option of the factorization (3.9). Certainly, for any square matrix both factorizations are identical.

Concluding, calculations of the QR matrix factorizations can be performed using:

1. The standard Gram-Schmidt orthogonalization algorithm, as applied in the proof of the Theorem 3.1, but for full-rank matrices only. A *modified Gram-Schmidt algorithm* is recommended, as it has better numerical properties. For a general case the Gram-Schmidt algorithm can also be applied, but must be appropriately augmented to deal with rank-defficient matrices – other methods, indicated below, are then more recommended.
2. The algorithm using the Householder's reflections – the factorization (3.9) is originally generated, then the thin factorization (3.8) is obtained by removing last $m-n$ columns from the matrix $\mathbf{Q}_{m\times m}$ and $m-n$ last rows from the matrix $\mathbf{R}_{m\times n}$, see Section 3.6. Most popular approach.
3. The algorithm using the Givens rotations, see Section 3.5 – the QR factorization (3.9) is originally generated, as it is in the case with the Householder rotations. Recommended first of all for sparse matrices.

Example 3.1. The QR factorizations of the following matrices will be calculated:

$$\text{a)}\;\; \mathbf{A}=[\mathbf{a}_1\; \mathbf{a}_2]=\begin{bmatrix} 1 & 1 \\ 2 & -1 \\ -2 & 4 \end{bmatrix},$$

$$\text{b)}\;\; \mathbf{B}=[\mathbf{a}_1\; \mathbf{a}_2\; \mathbf{a}_3]=\begin{bmatrix} 1 & 1 & 9 \\ 2 & -1 & 0 \\ -2 & 4 & 0 \end{bmatrix}.$$

Ad a). We have

$$\begin{aligned}
\bar{\mathbf{q}}_1 &= \mathbf{a}_1 = [1\ 2\ -2]^{\mathrm{T}},\\
\bar{\mathbf{q}}_2 &= \mathbf{a}_2 - \frac{\bar{\mathbf{q}}_1^{\mathrm{T}}\mathbf{a}_2}{\bar{\mathbf{q}}_1^{\mathrm{T}}\bar{\mathbf{q}}_1}\bar{\mathbf{q}}_1 = \mathbf{a}_2 - \bar{r}_{12}\bar{\mathbf{q}}_1 = \begin{bmatrix} 1\\ -1\\ 4 \end{bmatrix} - \frac{-9}{9}\begin{bmatrix} 1\\ 2\\ -2 \end{bmatrix} = \begin{bmatrix} 2\\ 1\\ 2 \end{bmatrix}.
\end{aligned}$$

This results in the "narrow" factorization (3.7)

$$\bar{\mathbf{Q}}_{\mathbf{A}} = \begin{bmatrix} 1 & 2\\ 2 & 1\\ -2 & 2 \end{bmatrix}, \quad \bar{\mathbf{R}}_{\mathbf{A}} = \begin{bmatrix} 1 & -1\\ 0 & 1 \end{bmatrix}. \tag{3.16}$$

Normalizing the columns of the matrix $\bar{\mathbf{Q}}_{\mathbf{A}}$ according to (3.13) - (3.15), we get

$$\mathbf{Q}_{\mathbf{A}} = \begin{bmatrix} \frac{1}{3} & \frac{2}{3}\\ \frac{2}{3} & \frac{1}{3}\\ -\frac{2}{3} & \frac{2}{3} \end{bmatrix}, \quad \mathbf{R}_{\mathbf{A}} = \begin{bmatrix} 3 & -3\\ 0 & 3 \end{bmatrix}. \tag{3.17}$$

Ad b). Two first columns of the matrix $\mathbf{B}$ are the same as in the matrix $\mathbf{A}$. Hence, only the third column must be normalized. According to (3.11), we have

$$\begin{aligned}
\bar{\mathbf{q}}_3 &= \mathbf{a}_3 - \frac{\bar{\mathbf{q}}_1^{\mathrm{T}}\mathbf{a}_3}{\bar{\mathbf{q}}_1^{\mathrm{T}}\bar{\mathbf{q}}_1}\bar{\mathbf{q}}_1 - \frac{\bar{\mathbf{q}}_2^{\mathrm{T}}\mathbf{a}_3}{\bar{\mathbf{q}}_2^{\mathrm{T}}\bar{\mathbf{q}}_2}\bar{\mathbf{q}}_2 = \mathbf{a}_3 - \bar{r}_{13}\bar{\mathbf{q}}_1 - \bar{r}_{23}\bar{\mathbf{q}}_2\\
&= \begin{bmatrix} 9\\ 0\\ 0 \end{bmatrix} - \frac{9}{9}\begin{bmatrix} 1\\ 2\\ -2 \end{bmatrix} - \frac{18}{9}\begin{bmatrix} 2\\ 1\\ 2 \end{bmatrix} = \begin{bmatrix} 4\\ -4\\ -2 \end{bmatrix}.
\end{aligned}$$

This results in the factorization (3.7)

$$\bar{\mathbf{Q}}_{\mathbf{B}} = \begin{bmatrix} 1 & 2 & 4\\ 2 & 1 & -4\\ -2 & 2 & -2 \end{bmatrix}, \quad \bar{\mathbf{R}}_{\mathbf{B}} = \begin{bmatrix} 1 & -1 & 1\\ 0 & 1 & 2\\ 0 & 0 & 1 \end{bmatrix}. \tag{3.18}$$

Normalizing columns of the matrix $\bar{\mathbf{Q}}_{\mathbf{B}}$ according to (3.13) - (3.15), we get

$$\mathbf{Q}_{\mathbf{B}} = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & \frac{2}{3}\\ \frac{2}{3} & \frac{1}{3} & -\frac{2}{3}\\ -\frac{2}{3} & \frac{2}{3} & -\frac{1}{3} \end{bmatrix}, \quad \mathbf{R}_{\mathbf{B}} = \begin{bmatrix} 3 & -3 & 3\\ 0 & 3 & 6\\ 0 & 0 & 6 \end{bmatrix}. \tag{3.19}$$

It can be easily seen that the matrices

$$\mathbf{Q}_{\mathbf{B}} = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & \frac{2}{3}\\ \frac{2}{3} & \frac{1}{3} & -\frac{2}{3}\\ -\frac{2}{3} & \frac{2}{3} & -\frac{1}{3} \end{bmatrix}, \quad \mathbf{R}_{\mathbf{A}+} = \begin{bmatrix} 3 & -3\\ 0 & 3\\ 0 & 0 \end{bmatrix} \tag{3.20}$$

constitute the factorization QR (3.9) of the matrix $\mathbf{A}$. $\square$

The modified Gram-Schmidt algorithm has better numerical properties than the standard Gram-Schmidt algorithm used in the proof of Theorem 3.1. The modified algorithm performs the process of orthogonalization in a different order. The standard algorithm (3.11) orthogonalizes vectors (columns of $\mathbf{A}$) one after another, the modified algorithm performes in another way: when a new column is orthogonalized, then all next columns are immediately orthogonalized with respect to this column. Thus, when performing the process of QR factorization, the modified algorithm creates a sequence of matrices $\{\mathbf{A} = \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(n)} = \bar{\mathbf{Q}}\}$, where $\mathbf{A}^{(i)} = [\bar{\mathbf{q}}_1\ \bar{\mathbf{q}}_2 \cdots \bar{\mathbf{q}}_{i-1}\ \mathbf{a}_i^{(i)}\ \mathbf{a}_{i+1}^{(i)} \cdots \mathbf{a}_n^{(i)}]$. Using the "for" loop from the MATLAB language, the modified Gram-Schmidt algorithm can be written in the form:

```
$\mathbf{A}^{(1)} = \mathbf{A}$;
for i = 1:n,
    $\bar{\mathbf{q}}_i = \mathbf{a}_i^{(i)}$;  $\bar{r}_{ii} = 1$;
    $d_i = \bar{\mathbf{q}}_i^{\mathrm{T}}\bar{\mathbf{q}}_i$;
    for j = i+1:n                                        (3.21)
        $\bar{r}_{ij} = \frac{\bar{\mathbf{q}}_i^{\mathrm{T}}\mathbf{a}_j^{(i)}}{d_i}$;
        $\mathbf{a}_j^{(i+1)} = \mathbf{a}_j^{(i)} - \bar{r}_{ij}\bar{\mathbf{q}}_i$;
    end
end
```

The m-function "qrmgs" given below implements in MATLAB the algorithm (3.21) for the QR (thin) factorization, together with the normalization (the last "for" loop). Therefore, the QR factorization (3.8) is the output from this procedure. The program is written for both real-valued and complex-valued matrices (notice that the order of elements in a scalar product of complex numbers is essential, thus for complex-valued vectors it is essential in the formula for $\bar{r}_{ij}$ coefficients).

```
function [Q,R]=qrmgs(A)
%QR (thin) factorization using modified Gram-Schmidt algorithm
%for full rank  real-valued and complex-valued matrices
[m n]=size(A);
Q=zeros(m,n);
R=zeros(n,n);
d=zeros(1,n);
%factorization with orthogonal (not orthonormal) columns of Q:
for i=1:n
   Q(:,i)=A(:,i);
   R(i,i)=1;
   d(i)=Q(:,i)'*Q(:,i);
   for j=i+1:n
```

```
        R(i,j)=(Q(:,i)'*A(:,j))/d(i);
        A(:,j)=A(:,j)-R(i,j)*Q(:,i);
   end
end
%column normalization (columns of Q orthonormal):
for i=1:n
    dd=norm(Q(:,i));
    Q(:,i)=Q(:,i)/dd;
    R(i,i:n)=R(i,i:n)*dd;
end
```

3.2. Eigenvalues

3.2.1. Preliminaries

Eigenvalues play an important role in science and technology, in particular in computer science and in automatic control.

An eigenvalue and a corresponding eigenvector of a real-valued square matrix $\mathbf{A}_n$ are defined as a pair consisting of a number (generally complex valued) $\lambda \in \mathbb{C}$ and a vectors $\mathbf{v} \in \mathbb{C}^n$ such that

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}, \tag{3.22}$$

where λ – an eigenvalue, $\mathbf{v}$ – a corresponding eigenvector. Thus, the eigenvalues and the eigenvectors satisfy the equation

$$(\mathbf{A} - \lambda\mathbf{I})\,\mathbf{v} = 0. \tag{3.23}$$

Therefore, λ is an eigenvalue of $\mathbf{A}$ if and only if it satisfies the following *characteristic equation:*

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0. \tag{3.24}$$

A square n-dimensional matrix has exactly n eigenvalues and corresponding eigenvectors. The eigenvalues are uniquely defined, whereas the eigenvectors are in fact *eigendirections*. This follows from the fact that the definition (3.22) implies directly that if $\mathbf{v}$ is an eigenvector, then also $\alpha\mathbf{v}$, $\alpha \in \mathbb{R}$, is an eigenvector corresponding to the same eigenvalue. Therefore, it is usually assumed that eigenvectors are normed (of unity length).

The set of all eigenvalues of a matrix $\mathbf{A}$ is called *the spectrum* of $\mathbf{A}$. The spectrum of a matrix $\mathbf{A}$ will denoted by $\mathrm{sp}(\mathbf{A})$. The following implication is practically useful

$$\lambda \in \mathrm{sp}(\mathbf{A}) \;\Rightarrow\; \forall\alpha \in \mathbb{C}\;\; (\lambda + \alpha) \in \mathrm{sp}(\mathbf{A} + \alpha\mathbf{I}), \tag{3.25}$$

it follows directly from the definition of the eigenvalue:

$$(\mathbf{A} + \alpha\mathbf{I})\mathbf{v} = \mathbf{A}\mathbf{v} + \alpha\mathbf{I}\mathbf{v} = \lambda\mathbf{v} + \alpha\mathbf{v} = (\lambda + \alpha)\mathbf{v}.$$

Definition. Two square matrices (of the same dimension) $\mathbf{A}$ and $\mathbf{B}$ are called *similar matrices*, if there is a nonsingular matrix $\mathbf{S}$ such that

$$\mathbf{S}^{-1}\mathbf{A}\mathbf{S} = \mathbf{B}. \tag{3.26}$$

Similar matrices have the same eigenvalues, because:

$$\begin{aligned}\det\left(\mathbf{S}^{-1}\mathbf{A}\mathbf{S}-\lambda\mathbf{I}\right) &= \det\left(\mathbf{S}^{-1}\mathbf{A}\mathbf{S}-\lambda\mathbf{S}^{-1}\mathbf{S}\right)\\ &= \det\mathbf{S}^{-1}\left(\mathbf{A}-\lambda\mathbf{I}\right)\mathbf{S}\\ &= \det\mathbf{S}^{-1}\det\left(\mathbf{A}-\lambda\mathbf{I}\right)\det\mathbf{S}\\ &= \det\left(\mathbf{A}-\lambda\mathbf{I}\right).\end{aligned}$$

Theorem 3.2. *If a matrix* $\mathbf{A}$: $\mathbb{R}^n \to \mathbb{R}^n$ *is symmetric, then all its eigenvectors and eigenvalues are real.*

Proof. let $\lambda \in \mathbb{C}$ be any eigenvalue of matrix a $\mathbf{A}$, and $\mathbf{v} \in \mathbb{C}_n$ the corresponding eigenvector. We have:

$$\bar{\mathbf{v}}^{\mathrm{T}}\mathbf{A}\mathbf{v} = \bar{\mathbf{v}}^{\mathrm{T}}\lambda\mathbf{v} = \lambda\bar{\mathbf{v}}^{\mathrm{T}}\mathbf{v} = \lambda\|\mathbf{v}\|^2.$$

On the other hand, due to the symmetry of $\mathbf{A}$, we have:

$$\bar{\mathbf{v}}^{\mathrm{T}}\mathbf{A}\mathbf{v} = \bar{\mathbf{v}}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{v} = [\mathbf{A}\bar{\mathbf{v}}]^{\mathrm{T}}\mathbf{v} = [\bar{\mathbf{A}\mathbf{v}}]^{\mathrm{T}}\mathbf{v} = [\bar{\lambda\mathbf{v}}]^{\mathrm{T}}\mathbf{v} = [\bar{\lambda}\bar{\mathbf{v}}]^{\mathrm{T}}\mathbf{v} = \bar{\lambda}\bar{\mathbf{v}}^{\mathrm{T}}\mathbf{v} = \bar{\lambda}\|\mathbf{v}\|^2.$$

Thus, it was proved that $\lambda\|\mathbf{v}\|^2 = \bar{\lambda}\|\mathbf{v}\|^2$. This implies $\lambda = \bar{\lambda}$, which is possible only for a real eigenvalue.

Further, the facts that the matrix $(\mathbf{A}-\lambda\mathbf{I})$ of the linear equation $(\mathbf{A}-\lambda\mathbf{I})\mathbf{v} = \mathbf{0}$ is real and singular and the right-hand side of this equation is zero directly imply that a non-zero solution of this equation exists and must be real, hence the eigenvector $\mathbf{v}$ is real. This concludes the proof. $\square$

It can be easily shown, reasoning similarly as in the proof of Theorem 3.2, that if all eigenvalues of a real matrix are different then all eigenvectors are mutually orthogonal (orthonormal). For multiple eigenvalues the normed eigenvectors are in general not unique, in fact *eigensubspaces* correspond then generally to multiple eigenvalues (different linearly independent eigenvectors corresponding to a multiple eigenvalue define its eigensubspace) – but always a set of n orthonormal eigenvectors can be selected. Concluding, *for each symmetric matrix there is a set of n orthonormal eigenvectors, creating an orthonormal base of the space* $\mathbb{R}^n$.

Theorem 3.3. *If all eigenvectors of a matrix are mutually orthogonal, then this matrix can be transformed to the diagonal matrix* $\operatorname{diag}\{\lambda_1, \dots, \lambda_n\}$*, by a similarity transformation.*

Proof. Let $\{\mathbf{v}_1, ..., \mathbf{v}_n\}$ be the set of all mutually orthogonal eigenvectors of a matrix $\mathbf{A}$, assume also that these vectors are orthonormal (are normalized). From the basic definition (3.22):

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad i = 1, ..., n.$$

Define the matrix

$$\mathbf{V} \overset{\text{df}}{=} [\mathbf{v}_1, ..., \mathbf{v}_n],$$

then we can write the above set of equalities in the following equivalent matrix form:

$$\mathbf{AV} = \mathbf{V} \operatorname{diag}(\lambda_i).$$

Multiplying now both sides by $\mathbf{V}^{\mathrm{T}}$ we get

$$\mathbf{V}^{\mathrm{T}}\mathbf{AV} = \mathbf{V}^{\mathrm{T}}\mathbf{V} \operatorname{diag}\{\lambda_i\}. \tag{3.27}$$

The vectors $\mathbf{v}_1, ..., \mathbf{v}_n$ are orthonormal, which means

$$\begin{aligned} \mathbf{v}_i^{\mathrm{T}}\mathbf{v}_j &= 0, \quad i \neq j, \\ \mathbf{v}_i^{\mathrm{T}}\mathbf{v}_i &= 1, \quad i, j = 1, ..., n. \end{aligned}$$

This implies that

$$\mathbf{V}^T\mathbf{V} = \mathbf{I},$$

which means that the matrix $\mathbf{V}$ is orthogonal, $\mathbf{V}^T = \mathbf{V}^{-1}$. Therefore, the equality (3.27) can be written in the form

$$\mathbf{V}^{-1}\mathbf{AV} = \operatorname{diag}\{\lambda_i\},$$

which concludes the proof. □

Corollary. Every real symmetric matrix can be transformed to the diagonal matrix $\operatorname{diag}\{\lambda_1, \dots, \lambda_n\}$ by a similarity transformation, and the eigenvectors of the original matrix are columns of the transformation matrix.

Passing to the general case of a *non-symmetric matrix*, *complex-valued eigenvalues and eigenvectors* must be considered as well, hence also *complex-valued matrices*.

Let the asterisk superscript (*) denote the operations of transposition and conjugation of elements of a square complex-valued matrix $\mathbf{A}$, i.e., $\mathbf{A}^* = \bar{\mathbf{A}}^{\mathrm{T}}$. A square matrix $\mathbf{A}$ with, generally, complex elements is called:

– *hermitian*, if $\mathbf{A} = \mathbf{A}^*$,

– *unitary*, if $\mathbf{A}^*\mathbf{A} = \mathbf{I}$ (i.e., $\mathbf{A}^* = \mathbf{A}^{-1}$).

A square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ with, in general, complex elements is called a *normal matrix*, if

$$\mathbf{A}^*\mathbf{A} = \mathbf{A}\mathbf{A}^*. \tag{3.28}$$

If a matrix $\mathbf{A}$ is normal, *then there is a set of n orthonormal eigenvectors of this matrix* (there is a set of its eigenvectors constituting a base of the space $\mathbb{R}^n$). Therefore, such matrices can be transformed to the diagonal form by a similarity transformation, similarly as it was shown in the Theorem 3.3 for symmetric matrices. Certainly, all hermitian matrices (i.e., symmetric in real-valued case) are normal.

Therefore, the possibility to diagonalize a square matrix is not a general property. However, for any square matrix (also nonsymmetric, also complex-valued) we have:

Theorem 3.4. . *Every square n-dimensional matrix* $\mathbf{A}$ *is similar to an upper triangular matrix* $\mathbf{R}$ *(in general complex-valued) with eigenvalues of* $\mathbf{A}$ *on the diagonal,*

$$\mathbf{U}^{-1}\mathbf{A}\mathbf{U} = \begin{bmatrix} \lambda_1 & x & \cdots & x & x \\ 0 & \lambda_2 & \cdots & x & x \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \lambda_n \end{bmatrix} = \mathbf{R},$$

where the similarity transformation matrix $\mathbf{U}$ *is unitary (i.e.,* $\mathbf{U} = \mathbf{U}^*$*) and all possibly non-zero elements are denoted by* x*.*

Selected methods for calculation of eigenvalues:

- *The determinant methods*, which exploit the fact that eigenvalues are roots of the characteristic polynomial, effective for a single eigenvalue or a limited number of eigenvalues.
- *The Jacobi's method*, a transformation of a symmetric matrix to the diagonal form using a series of Givens rotations (effective to the dimension $n \approx 10$).
- *The QR method*, most general and effective one.

Determinant methods for finding eigenvalues

Theoretically, the eigenvalues can be calculated as roots of the characteristic polynomial, i.e., as solutions of the equation $\det(\lambda\mathbf{I} - \mathbf{A}) = 0$. This means that single eigenvalues are subsequently calculated, as roots of the characteristic polynomial, using an appropriate root finding metod. Usually, the matrix is first transformed to a simpler form (the tridiagonal or the Hessenberg form). Methods implementing this strategy, e.g.:

1. *The bisection method using Sturm sequences* – for symmetric matrices. Every eigenvalue is found using the bisection method (see Section 6.1.1), decisions

about choices of subintervals containing the eigenvalue are easy, based on properties of the Sturm sequences. The original matrix should be first transformed to the *tridiagonal form*, i.e., having non-zero elements only on the diagonal, subdiagonal and superdiagonal (it is the Hessenberg form of symmetric matrices), see, e.g., Section 3.5, Section 3.6.

2. *The Hyman's method* – roots of the characteristic polynomial are found using the Newton's method (see Section 6.1.4), calculating in a specific way the value and the derivative of the polynomial. The original matrix should be first transformed to the *upper Hessenberg form*, i.e., to an almost upper triangular matrix, with non-zero elements on and above the diagonal and directly below the diagonal only (on the first subdiagonal), see, e.g., Section 3.6.

3.2.2. The QR method for finding eigenvalues

The core of this method is a subsequent use of the QR factorization of a square matrix.

The QR method for symmetric matrices

Before starting the method, it is recommended to transform the given matrix $\mathbf{A}$ to the *tridiagonal form* (the Hessenberg form of symmetric matrices). However, this is not necessary, but only recommended, as it increases then the effectiveness of calculations (QR factorizations and products of matrices are then more effectively calculated, as the $\mathbf{QR}$ factorization preserves the tridiagonal form).

The idea of a single step of the QR method (transformation of $\mathbf{A}^{(k)}$ into $\mathbf{A}^{(k+1)}$):

$$\begin{aligned}\mathbf{A}^{(k)} &= \mathbf{Q}^{(k)}\mathbf{R}^{(k)} \quad \text{(factorization)},\\ \mathbf{A}^{(k+1)} &= \mathbf{R}^{(k)}\mathbf{Q}^{(k)}.\end{aligned}$$

Because the matrix $\mathbf{Q}^{(k)}$ is orthogonal, we have

$$\mathbf{R}^{(k)} = (\mathbf{Q}^{(k)})^{-1}\mathbf{A}^{(k)} = \mathbf{Q}^{(k)\mathrm{T}}\mathbf{A}^{(k)},$$

thus

$$\mathbf{A}^{(k+1)} = \mathbf{Q}^{(k)T}\mathbf{A}^{(k)}\mathbf{Q}^{(k)},$$

hence, the matrix $\mathbf{A}^{(k+1)}$ is a transformation of the matrix $\mathbf{A}^{(k)}$ by similarity. Therefore, $\mathbf{A}^{(k+1)}$ has the same eigenvalues.

Basic algorithm (the QR method without shifts):

$$\begin{aligned}\mathbf{A}^{(1)} &= \mathbf{A},\\ \mathbf{A}^{(1)} &= \mathbf{Q}^{(1)}\mathbf{R}^{(1)} \quad \text{(factorization)},\\ \mathbf{A}^{(2)} &= \mathbf{R}^{(1)}\mathbf{Q}^{(1)} \ \ (= \mathbf{Q}^{(1)\mathrm{T}}\mathbf{A}^{(1)}\mathbf{Q}^{(1)}),\end{aligned}$$

$$\mathbf{A}^{(2)} = \mathbf{Q}^{(2)}\mathbf{R}^{(2)} \quad \text{(factorization)},$$
$$\mathbf{A}^{(3)} = \mathbf{R}^{(2)}\mathbf{Q}^{(2)} \ \ (= \mathbf{Q}^{(2)\mathrm{T}}\mathbf{A}^{(2)}\mathbf{Q}^{(2)}) = \mathbf{Q}^{(2)\mathrm{T}}\mathbf{Q}^{(1)\mathrm{T}}\mathbf{A}^{(1)}\mathbf{Q}^{(1)}\mathbf{Q}^{(2)}),$$
etc.
$$\mathbf{A}^{(k)} \longrightarrow \ \mathbf{V}^{-1}\mathbf{A}\mathbf{V} = \mathrm{diag}\,\{\lambda_\mathrm{i}\}\,.$$

Thus, for a symmetric matrix $\mathbf{A}$, the matrix $\mathbf{A}^{(k)}$ converges to the diagonal matrix $\mathrm{diag}\,(\lambda_i)$.

Rate of convergence: if $\mathbf{A}$ has all eigenvalues with distinct absolute values,

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n| > 0,$$

then it can be shown that

$$a_{i,i}^{(k)} \underset{k\to\infty}{\longrightarrow} \lambda_i, \ \ i-1,...,n,$$

$$a_{i+1,i}^{(k)} \underset{k\to\infty}{\longrightarrow} 0, \ \ i=1,...,n-1$$

and the convergence of an off-diagonal element $a_{i+1,i}^{(k)}$ to zero is linear with the convergence ratio $\left|\frac{\lambda_{i+1}}{\lambda_i}\right|$, i.e.,

$$\frac{|a_{i+1,i}^{(k+1)}|}{|a_{i+1,i}^{(k)}|} \approx \left|\frac{\lambda_{i+1}}{\lambda_i}\right|.$$

Therefore, the method can be slowly convergent if certain eigenvalues have similar values. A remedy is to use the method *with shifts*.

A single iteration of the QR method with shifts:

$$\begin{aligned}
\mathbf{A}^{(k)} - p_k\mathbf{I} &= \mathbf{Q}^{(k)}\mathbf{R}^{(k)},\\
\mathbf{A}^{(k+1)} &= \mathbf{R}^{(k)}\mathbf{Q}^{(k)} + p_k\mathbf{I}\\
&= \mathbf{Q}^{(k)T}(\mathbf{A}^{(k)} - p_k\mathbf{I})\mathbf{Q}^{(k)}{+}p_k\mathbf{I}\\
&= \mathbf{Q}^{(k)T}\mathbf{A}^{(k)}\mathbf{Q}^{(k)},
\end{aligned}$$

because the matrix $\mathbf{Q}^{(k)}$ is orthogonal. The convergence ratio is then

$$\left|\frac{\lambda_{i+1} - p_k}{\lambda_i - p_k}\right|.$$

Therefore, the best shift p_k should be chosen as an actual estimate of λ_{i+1}. Practically, the following procedure is applied: if

$$\mathbf{A}^{(k)} = \left[\begin{array}{cccccc} d_1^{(k)} & e_1^{(k)} & \dots & 0 & 0 & 0 \\ e_1^{(k)} & d_2^{(k)} & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & d_{n-2}^{(k)} & e_{n-2}^{(k)} & 0 \\ \cline{5-6} 0 & 0 & \dots & e_{n-2}^{(k)} & \multicolumn{1}{|c}{d_{n-1}^{(k)}} & e_{n-1}^{(k)} \\ 0 & 0 & \dots & 0 & \multicolumn{1}{|c}{e_{n-1}^{(k)}} & d_n^{(k)} \end{array}\right] \tag{3.29}$$

(a tridiagonal matrix form is considered) then $d_n^{(k)}$ itself or, much better, a closer to $d_n^{(k)}$ eigenvalue of the 2×2 submatrix of $\mathbf{A}^{(k)}$ located at its lower right corner, i.e., of the matrix

$$\begin{bmatrix} d_{n-1}^{(k)} & e_{n-1}^{(k)} \\ e_{n-1}^{(k)} & d_n^{(k)} \end{bmatrix},$$

is taken as the shift p_k for the next iteration. The procedure is then repeated until $d_n^{(k)} = \lambda_n$, which is indicated by $e_{n-1}^{(k)} = 0$. In the case of a full (not tridiagonal) matrix transformations, the equality $d_n^{(k)} = \lambda_n$ will be true if all elements of the last row of $\mathbf{A}^{(k)}$, except the diagonal one, are zero.

Next, the procedure can be repeated for the $(n-1) \times (n-1)$ matrix obtained after dropping the last row and column corresponding to the obtained eigenvalue λ_n (*a deflation procedure*). Thus, we have the following:

Structure of the QR algorithm with shifts

1. The eigenvalue λ_n is found, as a closer to $d_n^{(k)}$ eigenvalue of the 2×2 submatrix from the right lower corner of $\mathbf{A}^{(k)}$,
2. The last row and the last column of the actual matrix $\mathbf{A}^{(k)}$ are deleted, i.e., only the submatrix $\mathbf{A}_{n-1}^{(k)}$ is further taken into account (compare it with full matrix (3.29),

$$\mathbf{A}_{n-1}^{(k)} = \begin{bmatrix} d_1^{(k)} & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & d_{n-2}^{(k)} & e_{n-2}^{(k)} \\ 0 & \dots & e_{n-2}^{(k)} & d_{n-1}^{(k)} \end{bmatrix}.$$

3. Next eigenvalue λ_{n-1} is found using the same procedure – i.e., the matrix $\mathbf{A}_{n-1}^{(k)}$ is transformed using the QR procedure until $e_{n-2}^{(k)} = 0$. In the case of full (not tridiagonal) matrix transformations, the iterations are performed until all the elements of the last matrix row, except the diagonal one ($d_{n-1}^{(k)}$), are zero.

4. The last row and column of the actual matrix $\mathbf{A}_{n-1}^{(k)}$ are deleted (deflation),

   $\vdots$

   etc., until all eigenvalues are found.

An outline of the QR method for nonsymmetric matrices

The QR method is always convergent and very effective (when using the algorithm with shifts) for symmetric matrices. The method is also applicable and very effective for *nonsymmetric matrices*. Similarly as it was for the symmetric matrices, it is recommended to transform the initial matrix to a simpler form, before starting the basic QR algorithm. This time it is the full *upper Hessenberg form*. The initial simplifying transformation it is not necessary, but recommended, as it reduces the computational burden, exploiting the fact that the QR factorization of the upper Hessenberg matrix is more effective, and the transformations of each single iteration of the QR algotithm preserve the Hessenberg form.

The implementation of the QR algorithm for general (also nonsymmetric) matrices can be exactly the same as the one discussed above, but the possibility to arrive at a complex eigenvalue must be taken into account. Therefore, the shift must be always chosen basing on the 2×2 lower right corner submatrix of $\mathbf{A}_i^{(k)}$ $(2 \le i \le n)$ and the complex arithmetic must be used.

The complex arithmetic can be avoided when implementing an algorithm version operating only with real-valued matrices, such a version exists and is based on changes of the shifts every two iterations of the method (i.e., every two transformations of the matrix) – the formulae and programs can be found in, e.g., [3].

However, the QR method in the basic version presented up to now, may fail for nonsymmetric matrices. For instance, for the matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

the QR algorithm without shifts in not convergent, because

$$\mathbf{A} = \mathbf{QR} = \mathbf{AI}, \quad \text{i.e.,} \quad \mathbf{A} = \mathbf{A}^{(1)} = \mathbf{A}^{(2)} = \mathbf{A}^{(3)} = \ldots$$

Moreover, it is not convergent also with the shifts, because the eigenvalues of 2×2 submatrix from the bottom right corner of $\mathbf{A}$ are both zero. In such cases, a way to achieve the convergence can be to apply a preliminary similarity transformation of the original matrix.

Example 3.2. We shall present and compare the numbers of iterations required to calculate the eigenvalues of several different matrices, using the QR algorithm without shifts and with shifts. For simplicity of the programs, the algorithms will be implemented without preliminary matrix transformation to the upper Hessenberg form (the tridiagonal form for symmetric matrices).

The following simple program of the QR algorithm without shifts, written in the MATLAB environment, was used to calculate the eigenvalues of the symmetric matrices:

```
function eigenvalues=EigvalQRNoShift(D,tol,imax)
%tol - tolerance (upper bound) for nulled elements
%imax - maximal number of iterations
n=size(D,1);
i=1;
while i <= imax & max(max(D-diag(diag(D)))) > tol
   [Q1,R1]=qr(D);
   D=R1*Q1;  %transformed matrix
   i=i+1;
end
if i > imax
      error('imax exceeded program terminated');
   end
eigenvalues=diag(D);
```

The implemented program of the QR method with shifts is as follows:

```
function eigenvalues=EigvalQRshifts(A,tol,imax)
%tol - tolerance (upper bound) for nulled elements
%imax - max number of iterations to calculate an eigenvalue
n=size(A,1);
eigenvalues=diag(ones(n));
INITIALsubmatrix=A; % initial matrix (original)
for k=n:-1:2,
    DK=INITIALsubmatrix; %initial matrix to calculate
                                      % a single eigenvalue
    i=0;
    while i<=imax & max(abs(DK(k,1:k-1)))>tol,
      DD=DK(k-1:k,k-1:k); % 2x2 bottom right corner submatrix
      [ev1,ev2]=quadpolynroots(1,-(DD(1,1)+DD(2,2)),...
                        ...DD(2,2)*DD(1,1)-DD(2,1)*DD(1,2));
      if abs(ev1-DD(2,2)) < abs(ev2-DD(2,2)),
         shift=ev1; %shift - DD eigenvalue closest to DK(k,k)
      else
         shift=ev2;
```

```
        end
          DP=DK-eye(k)*shift; %shifted matrix
          [Q1,R1]=qr(DP); %QR factorization
          DK=R1*Q1+eye(k)*shift; %transformed matrix
          i=i+1;
      end
      if i > imax
          error('imax exceeded program terminated');
      end
      eigenvalues(k)=DK(k,k);
      if k > 2,
         INITIALsubmatrix=DK(1:k-1,1:k-1); %matrix deflation
      else
         eigenvalues(1)=DK(1,1); %last eigenvalue
      end
end
```

The MATLAB procedure ”`qr`” was applied in the programs for the QR factorization (the m-function ”`qrmgs`” presented previously in Section 3.1 was designed for full rank matrices only). To calculate eigenvalues of the 2×2 submatrix from the bottom right corner of the transformed matrix in the second program, a simple m-function ”`quadpolynroots`” was used, calculating roots of a quadratic polynomial with coefficients being its arguments.

The procedure ”`EigvalQRshifts`”, when applied to the symmetric matrices, operates on real numbers and calculates all the (real) eigenvalues. However, it can also be applied to calculate eigenvalues of a nonsymmetric matrix, if operating with complex numbers. MATLAB switches itself automatically to the complex arithmetic when a complex number occurs during the calculations, i.e., if a complex eigenvalue occurs – and this can happen when calculating the eigenvalues of the nonsymmetric 2×2 lower right corner submatrix of the transformed matrix (using function ”`quadpolynroots`”).

Calculations of eigenvalues were made for several sample matrices, of dimension 5×5 and 10×10, generated by the MATLAB instruction ”`rand`”. Every symmetric matrix was obtained as a sums of a random matrix and its transpose. The results, in the form of numbers of iteration counted as numbers of calls of ”`qr`” function (the QR factorization), are presented in Table 3.1, for three sample matrices of each dimension and class (symmetric or nonsymmetric). Additionally, the numbers of complex eigenvalues are shown in the last column of the table. The presented results clearly show that the QR algorithm with shifts is far superior and more reliable. Not only significantly lower numbers of iterations are needed, but in several cases, only this algorithm enables successful solution.

Table 3.1. Numbers of iterations of the QR algorithm without shifts and with shifts

tolerance	tol=0.00001		tol=0.0000001		
algorithm with shifts	no	yes	no	yes	number of complex eigenvalues
symmetric matrix	35	7	47	8	0
5×5	154	6	failed	23	0
	91	8	124	9	0
symmetric matrix	93	16	129	16	0
10×10	132	24	193	76	0
	failed	15	failed	18	0
nonsymmetric matrix	–	7	–	8	2
5×5	–	55	–	78	4
	–	10	–	11	2
nonsymmetric matrix	–	24	–	27	6
10×10	–	44	–	54	4
	–	26	–	45	4

□

3.3. Singular values, SVD decomposition

Let us consider a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, where $m \geq n$. The matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ has dimension $n\times n$ and is symmetric and positive semidefinite, i.e.,

$$\forall\lambda \in \mathrm{sp}\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right) \qquad \lambda_i \geq 0.$$

Therefore, we can present its eigenvalues as

$$\lambda_i = (\sigma_i)^2, \qquad \text{where} \quad \sigma_i \geq 0, \qquad i = 1, ..., n.$$

The numbers σ_i are called *singular values* of the matrix $\mathbf{A}$.

Corollary. If a square matrix $\mathbf{A}$ is symmetric and positive semidefinite, then its eigenvalues are identical with its singular values.

If a square matrix $\mathbf{A}$ is only symmetric (not necessarily positive semidefinite), then

$$\sigma_i = |\lambda_i|, \quad \lambda_i \in \mathrm{sp}(\mathbf{A}), \qquad i = 1, ..., n.$$

A matrix factorization performed along its singular values, called the *singular value decomposition* (SVD) has many important applications. The following theorem formulates this decomposition.

Theorem 3.5. . *For any matrix* $\mathbf{A}_{m\times n}$ *(*$\mathbf{A}\in\mathbb{R}^{m\times n}$, $m\geq n$*) there are orthogonal matrices* $\mathbf{U}_{m\times m}$, $\mathbf{V}_{n\times n}$ *and a matrix* $\mathbf{\Sigma}_{m\times n}$, *such that:*

$$\mathbf{A}=\mathbf{U\Sigma V}^{\mathrm{T}},$$

where

$$\mathbf{\Sigma}=\begin{bmatrix}\mathbf{D}_{n\times n}\\ \mathbf{0}_{(m-n)\times n}\end{bmatrix}_{m\times n}=\left[\begin{array}{cccc}\sigma_1 & 0 & \cdots & 0\\ 0 & \ddots & & \\ \vdots & & \sigma_k & \vdots\\ & & \ddots & 0\\ 0 & \cdots & 0 & \sigma_n\\ \hline & \multicolumn{2}{c}{\mathbf{0}} & \end{array}\right],$$

- $\mathbf{D}_{n\times n}$= diag $(\sigma_i,\ i=1,...,n)$, *where* $\sigma_i\geq 0$. *If some eigenvalues* σ_i *are zero, then* $\mathbf{D}$ *is singular (and* $\mathbf{A}$ *is not of full rank – the number of nonzero singular values is equal to the rank* k *of* $\mathbf{A}$*),*
- $\mathbf{V}_{m\times m}$ *is a matrix with columns consisting of orthonormal eigenvectors of the matrix* $\mathbf{A}^{\mathrm{T}}\mathbf{A}$,
- $\mathbf{U}_{n\times n}$ *is a matrix with columns consisting of orthonormal eigenvectors of the matrix* $\mathbf{A}\mathbf{A}^{\mathrm{T}}$.

Comment. It should be pointed out that calculation of σ_i directly as the square roots of the eigenvalues of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ *may lead to a loss of significant digits and thus to significant errors*, as illustrated by a simple example below.

Example 3.3. We shall calculate singular values of a matrix $\mathbf{A}$, exactly and numerically for a given precision *eps*, directly according to the definition of the singular values as the square roots of the eigenvalues of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, where

$$\mathbf{A}=\begin{bmatrix}1 & 1\\ \varepsilon & 0\\ 0 & \varepsilon\end{bmatrix},\quad |\varepsilon|<\sqrt{eps}.$$

The exact eigenvalues of this matrix are

$$\sigma_1(\mathbf{A})=\sqrt{2+\varepsilon^2},\quad \sigma_2(\mathbf{A})=|\varepsilon|\,.$$

On the other hand,

$$\mathbf{A}^T\mathbf{A}=\begin{bmatrix}1+\varepsilon^2 & 1\\ 1 & 1+\varepsilon^2\end{bmatrix},$$

$$fl\left(\mathbf{A}^T\mathbf{A}\right)=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix},\quad \text{because } \varepsilon^2<eps.$$

Therefore, the eigenvalues of $\mathbf{A}$ calculated numerically as the square roots of the eigenvalues of $\mathbf{A}^T\mathbf{A}$ are equal to $\widetilde{\sigma}_1 = \sqrt{2}$, $\widetilde{\sigma}_2 = 0$, where tilde $(\widetilde{\ })$ denotes numerical values obtained in the floating point arithmetic. But we know that $\sigma_2(\mathbf{A}) = |\varepsilon|$, and this shows that a very large numerical error emerged, several orders of magnitude larger than a roundoff error of the number representation. For instance, for $eps = 10^{-12}$ and $\varepsilon = 10^{-7}$ we have $|\varepsilon| < \sqrt{eps} = 10^{-6}$ and $\tilde{\sigma}_2(\mathbf{A}) = 0$ – but $\sigma_2(\mathbf{A}) = 10^{-7} \gg 0$. □

To calculate the singular value decomposition, the algorithms avoiding the loss of accuracy should be used, e.g., the Golub-Reinsch algorithm, see, e.g., [3]. The SVD decomposition is of a key importance for many numerical problems, e.g., it is applied for an exact calculation of the rank of a matrix, for a solution of a linear least-squares problem with a rank-defficient matrix.

3.4. Linear least-squares problem

The *linear least-squares problem* (LLSP) can be defined in the following way: *For a given matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *(*$m > n$*) and vector* $\mathbf{b} \in\mathbb{R}^m$*, find a vector* $\widehat{\mathbf{x}} \in \mathbb{R}^n$ *such that*

$$\forall \mathbf{x} \in\mathbb{R}^n \qquad \|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\|_2 \le \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2 . \tag{3.30}$$

Thus, the LLSP can be treated as a problem of solving a set of m linear equations with n variables (unknowns), where $n < m$ (an overdetermined system of equations). As in this case, in general, the system of equations cannot be solved accurately, we look for a solution minimizing the Euclidean norm of the solution error.

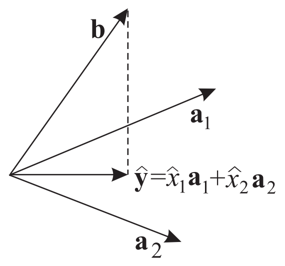
Geometrical interpretation: $\mathbf{A}\widehat{\mathbf{x}}$ is an orthogonal projection of $\mathbf{b}$ onto the subspace spanned by all columns of the matrix $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2\ \cdots\ \mathbf{a}_n]$, i.e., onto the subspace

$$\mathbb{Y} \subset \mathbb{R}^m = \{\mathbf{y} \in \mathbb{R}^m :\ \mathbf{y} = \mathbf{A}\mathbf{x} = x_1\mathbf{a}_1 + \cdots + x_n\mathbf{a}_n, \mathbf{x} = [x_1\ \cdots\ x_n]^{\mathrm{T}} \in \mathbb{R}^n\},$$

which is illustrated in Fig. 3.1 for the case $n = 2$. It should be noticed, that the norm of the error (a difference between left and right sides of the equations) can be zero only if the right-hand side vector $\mathbf{b}$ is linearly dependent on the columns of the matrix $\mathbf{A}$, i.e., when it belongs to the subspace $\mathbb{Y}$. This happens very rarely for $m > n$.

The LLSP can have a nonunique solution, therefore a minimal norm of the solution is usually required, to make the solution unique, i.e.,

$$(\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2 = \|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\|_2) \Rightarrow \|\widehat{\mathbf{x}}\|_2 \le \|\mathbf{x}\|_2 . \tag{3.31}$$



$\mathbf{b}$
$\mathbf{a}_1$
$\hat{\mathbf{y}}=\hat{x}_1\mathbf{a}_1+\hat{x}_2\mathbf{a}_2$
$\mathbf{a}_2$

Figure 3.1. Geometrical interpretation of the solution $\hat{\mathbf{x}} = [\hat{x}_1\ \hat{x}_2]^{\mathrm{T}}$ of the LLSP, for $n = 2$

Another possible approach is to require a solution vector with a maximal number of elements equal to zero. Let us observe that the LLSP is equivalent to the minimization of the following quadratic function $J(x)$:

$$J(x) = (\mathbf{b} - \mathbf{A}\mathbf{x})^{\mathrm{T}}(\mathbf{b} - \mathbf{A}\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} - 2\mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{b} + \mathbf{b}^{\mathrm{T}}\mathbf{b}. \tag{3.32}$$

Methods for solving the LLSP problem

Denote by k the rank of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, $k \leq n$.

First, assume that $k = n$, i.e., the *matrix* $\mathbf{A}$ *has full rank.* Then the square nad symmetric matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is positive definite (has all the eigenvalues positive), hence the function $J(x)$ is strictly convex and has a unique minimum at the point satisfying the necessary optimality condition – equating the function gradient to zero:

$$J'(x)^{\mathrm{T}} = 2\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} - 2\mathbf{A}^{\mathrm{T}}\mathbf{b} = \mathbf{0}.$$

(the gradient of a multivariable function is a column vector – while the derivative is a row vector, being its transpose). This results in a linear system of equations

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{A}^{\mathrm{T}}\mathbf{b}, \tag{3.33}$$

which is known as the set of *normal equations*. In the considered case, it has a unique solution. However, for a badly conditioned matrix $\mathbf{A}$, the condition number for the matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ becomes even much worse, because

$$\mathrm{cond}_2(\mathbf{A}^{\mathrm{T}}\mathbf{A}) = (\mathrm{cond}_2\mathbf{A})^2 = (\frac{\sigma_1}{\sigma_{\mathrm{n}}})^2,$$

where σ_1 and σ_n are the largest and the smallest singular values of $\mathbf{A}$. In such cases, it is recommended to find the solution using the QR factorization of the matrix $\mathbf{A}$, and it is convenient to use the thin (economical) QR factorization (3.8), $\mathbf{A}_{m\times n} = \mathbf{Q}_{m\times n}\mathbf{R}_{n\times n}$. The set of normal equations (3.33) can then be written in the form

$$\mathbf{R}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{Q}\mathbf{R}\mathbf{x} = \mathbf{R}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{b}. \tag{3.34}$$

Due to the orthonormality of columns of the matrix $\mathbf{Q}$, we have $\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}$. Taking also into account that the matrix $\mathbf{R}$ is nonsingular (because $k = n$), we obtain finally the following well defined system of linear equations

$$\mathbf{R}\mathbf{x} = \mathbf{Q}^{\mathrm{T}}\mathbf{b}. \tag{3.35}$$

It should be noted that if the QR factorization is evaluated to solve the LLSP only, then the narrow, not normalized QR factorization $\bar{\mathbf{Q}}\bar{\mathbf{R}}$ (3.7) can be applied. Then, instead of (3.35), we solve the following system of linear equations

$$\bar{\mathbf{R}}\mathbf{x} = (\bar{\mathbf{Q}}^{\mathrm{T}}\bar{\mathbf{Q}})^{-1}\bar{\mathbf{Q}}^{\mathrm{T}}\mathbf{b}, \tag{3.36}$$

where the term $(\bar{\mathbf{Q}}^{\mathrm{T}}\bar{\mathbf{Q}})^{-1}$ can be used with no cost, because

$$\bar{\mathbf{Q}}^{\mathrm{T}}\bar{\mathbf{Q}} = \mathrm{diag}\{d_1, ..., d_n\}$$

and the values $d_i = \bar{\mathbf{q}}_i^{\mathrm{T}}\bar{\mathbf{q}}_i$ are calculated during the QR factorization process using the Gram-Schmidt algorithm (prefarably a modified one), see (3.21).

Consider now the case with $k < n$, i.e., when the *matrix* $\mathbf{A}$ *is of reduced rank.* In this case, an algorithm based on the SVD decomposition is recommended. Applying the SVD decomposition to the matrix $\mathbf{A}$ and then the Lemma 3.1, we have

$$\begin{aligned} ||\mathbf{b} - \mathbf{A}\mathbf{x}||_2 &= \left\|\mathbf{b} - \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}\mathbf{x}\right\|_2 \\ &= \left\|\mathbf{U}^{\mathrm{T}}\mathbf{b} - \boldsymbol{\Sigma}\left(\mathbf{V}^{\mathrm{T}}\mathbf{x}\right)\right\|_2 \\ &= \left\|\widetilde{\mathbf{b}} - \boldsymbol{\Sigma}\widetilde{\mathbf{x}}\right\|_2, \end{aligned}$$

where $\widetilde{\mathbf{b}} = \mathbf{U}^{\mathrm{T}}\mathbf{b}$, $\widetilde{\mathbf{x}} = \mathbf{V}^{\mathrm{T}}\mathbf{x}$.

Denote: $\widetilde{\mathbf{b}} = [\widetilde{\mathbf{b}}_1^{\mathrm{T}}\ \widetilde{\mathbf{b}}_2^{\mathrm{T}}]^{\mathrm{T}},\ \widetilde{\mathbf{b}}_1 \in \mathbb{R}^n, \widetilde{\mathbf{b}}_2 \in \mathbb{R}^{m-n}$, then

$$||\mathbf{b} - \mathbf{A}\mathbf{x}||_2 = \left\|\widetilde{\mathbf{b}} - \boldsymbol{\Sigma}\widetilde{\mathbf{x}}\right\|_2 = \left\| \begin{array}{c} \widetilde{\mathbf{b}}_1 - \mathbf{D}\widetilde{\mathbf{x}} \\ \widetilde{\mathbf{b}}_2 \end{array} \right\|_2 = \left\| \begin{array}{c} \widetilde{b}_1 - \sigma_1\widetilde{x}_1 \\ \vdots \\ \widetilde{b}_k - \sigma_k\widetilde{x}_k \\ \widetilde{b}_{k+1} \\ \vdots \\ \widetilde{b}_m \end{array} \right\|_2,$$

where $k \le n$ is the number of positive singular values (is the rank of $\mathbf{A}$). Thus, the norm$||\mathbf{b} - \mathbf{A}\mathbf{x}||_2$ attains its mimimum equal to $\sqrt{\sum_{i=k+1}^{m}(\widetilde{b}_i)^2}$, for every $\widetilde{\mathbf{x}}$ of the form:

$$\widetilde{\mathbf{x}} = \left[\begin{array}{c} \widetilde{x}_1 = \widetilde{b}_1/\sigma_1 \\ \vdots \\ \widetilde{x}_k = \widetilde{b}_k/\sigma_k \\ \widetilde{x}_{k+1} = \text{any value} \\ \vdots \\ \widetilde{x}_n = \text{any value} \end{array} \right]. \tag{3.37}$$

Therefore, a uniquely defined solution $\widehat{\widehat{\mathbf{x}}}$ is

$$\widehat{\widehat{\mathbf{x}}} = \begin{bmatrix} \sigma_1/\widetilde{b}_1 \\ \vdots \\ \sigma_k/\widetilde{b}_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{3.38}$$

which can be written in the form

$$\widehat{\widehat{\mathbf{x}}} = \mathbf{\Sigma}^+\widetilde{\mathbf{b}}, \text{ where } \mathbf{\Sigma}^+ = \left[\mathbf{D}^+_{n\times n} \ \mathbf{0}_{n\times(m-n)}\right], \ \mathbf{D}^+ = \operatorname{diag}\left\{\frac{1}{\sigma_1}, ..., \frac{1}{\sigma_\mathrm{k}}, 0, ..., 0\right\}.$$

Hence

$$\mathbf{V}^\mathrm{T}\widehat{\mathbf{x}} = \mathbf{\Sigma}^+\mathbf{U}^\mathrm{T}\mathbf{b},$$

therefore

$$\widehat{\mathbf{x}} = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^\mathrm{T}\mathbf{b}, \tag{3.39}$$

where the matrix

$$\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^\mathrm{T} \tag{3.40}$$

is called *a pseudoinverse of* $\mathbf{A}$.

Example 3.4. The system of linear equations $\mathbf{Ax} = \mathbf{b}$ will be solved, minimizing the solution error norm, where

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & -1 \\ -2 & 4 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}.$$

The matrix $\mathbf{A}$ has full rank (all columns are linearly independent), hence the solution can be found from the set of normal equations (3.33), or using the QR factorization.

The set of normal equations has the form

$$\begin{bmatrix} 9 & -9 \\ -9 & 18 \end{bmatrix}\mathbf{x} = \begin{bmatrix} 3 \\ 6 \end{bmatrix},$$

which has the solution $\mathbf{x} = [\frac{4}{3}, 1]^\mathrm{T}$.

To apply the QR factorization, the narrow factorization is sufficient – such factorization of our matrix $\mathbf{A}$ was calculated in Example 3.1, thus the following system of equations results

$$\mathbf{Rx} = \mathbf{Q}^\mathrm{T}\mathbf{b} \quad \Leftrightarrow \quad \begin{bmatrix} 3 & -3 \\ 0 & 3 \end{bmatrix}\mathbf{x} = \begin{bmatrix} 1 \\ 3 \end{bmatrix},$$

which certainly leads to the same solution vector.

The system of equations $\mathbf{Rx} = \mathbf{Q}^{\mathrm{T}}\mathbf{b}$ needs less computations then the normal system of equations, because it is with (upper) triangular matrix. Moreover, the matrix $\mathbf{R}$ is better conditioned than the matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$. However, the QR factorization must be first performed, and in the case of normal equations the product of matrices $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ must be calculated. □

3.5. Givens transformation, with applications[*]

3.5.1. Givens transformation (rotation)

The Givens rotation matrix, rotating by an angle θ in the subplane given by the versors $\mathbf{e}_p$ and $\mathbf{e}_q$, is defined as follows:

$$\mathbf{G}_{pq} = \begin{array}{cc} \left[\begin{array}{ccccccccccc} 1 & 0 & & & & \cdots & & & & 0 & 0 \\ 0 & \ddots & & & & & & & & & 0 \\ & & 1 & & & & & & & & \\ & & & c & 0 & \cdots & 0 & s & & & \\ & & & 0 & 1 & & 0 & 0 & & & \\ \vdots & & & \vdots & & \ddots & & \vdots & & & \vdots \\ & & & 0 & 0 & & 1 & 0 & & & \\ & & & -s & 0 & \cdots & 0 & c & & & \\ & & & & & & & & 1 & & \\ 0 & & & & & & & & & \ddots & 0 \\ 0 & 0 & & & & \cdots & & & & 0 & 1 \end{array}\right] & \begin{array}{l} \\ \\ \\ \text{row } p \\ \\ \\ \\ \text{row } q \\ \\ \\ \\ \end{array} \\ \begin{array}{ccc} \vdots & \qquad & \vdots \\ \text{column } p & & \text{column } q \end{array} & \end{array} \tag{3.41}$$

where

$$c = \cos\theta,\ s = \sin\theta,$$

i.e., this matrix differs from a unity matrix in four elements $(c, s, -s, c)$, located in rows and columns indexed by p and q, as it can be seen in (3.41). Because $c^2 + s^2 = 1$, then $\mathbf{G}_{pq}\left(\mathbf{G}_{pq}\right)^{\mathrm{T}} = \mathbf{I}$, hence the matrix $\mathbf{G}_{pq}$ is orthogonal.

When a vector $\mathbf{a}$ is transformed,

$$\mathbf{b} = \mathbf{G}_{pq}\mathbf{a},$$

[*]Optional.

then the formulae for its components are as follows:

$$\begin{aligned} b_j &= a_j, \qquad \text{for } j \neq p, q\ , \\ b_p &= c a_p + s a_q, \\ b_q &= -s a_p + c a_q. \end{aligned}$$

Zeroing the coordinate b_p of the vector $\mathbf{b}$ using the Givens rotation is as follows:

$$\begin{aligned} b_q &= -s a_p + c a_q = 0, \\ 1 &= s^2 + c^2, \end{aligned}$$

which leads to
$$c = \frac{a_p}{\sqrt{a_p^2 + a_q^2}}, \quad s = \frac{a_q}{\sqrt{a_p^2 + a_q^2}}\ ,$$

whereas $c = s = 0$ when $a_p = a_q = 0$.

After a Givens rotation defined by the matrix $\mathbf{G}_{pq}$:

– in $\mathbf{G}_{pq}\mathbf{A}$ or $(\mathbf{G}_{pq})^{\mathrm{T}}\mathbf{A}$, the changes are only in the p-th and q-th rows of $\mathbf{A}$,
– in $\mathbf{A}\mathbf{G}_{pq}$ or $\mathbf{A}(\mathbf{G}_{pq})^{\mathrm{T}}$, the changes are only in the p-th and q-th column of$\mathbf{A}$.

Givens rotations preserving matrix similarity

It is easy to observe that differences between the matrices $\mathbf{A}' = (\mathbf{G}_{pq})^{\mathrm{T}}\mathbf{A}\mathbf{G}_{pq}$ and $\mathbf{A}$ *are only in the p-th and q-th rows and columns*. Therefore, it is not difficult to derive adequate relations:

$$\begin{aligned} a'_{ij} &= a_{ij}, \qquad \text{for } i, j \neq p, q, \\ a'_{pj} &= c a_{pj} - s a_{qj}, \qquad \text{for } j \neq p, q, \\ a'_{qj} &= s a_{pj} + c a_{qj}, \qquad \text{for } j \neq p, q, \\ a'_{jp} &= c a_{jp} - s a_{jq}, \qquad \text{for } j \neq p, q, \\ a'_{jq} &= s a_{jp} + c a_{jq}, \qquad \text{for } j \neq p, q, \end{aligned}$$

$$\begin{aligned} a'_{pp} &= (c a_{pp} - s a_{qp}) c - s\,(c a_{pq} - s a_{qq}) \\ &= c^2 a_{pp} + s^2 a_{qq} - sc\,(a_{pq} + a_{qp})\,, \\ a'_{qq} &= (s a_{pp} + c a_{qp}) s + c\,(s a_{pq} + c a_{qq}) \\ &= s^2 a_{pp} + c^2 a_{qq} + sc\,(a_{pq} + a_{qp})\,, \\ a'_{pq} &= c^2 a_{pq} - s^2 a_{qp} + sc\,(a_{pp} - a_{qq})\,, \\ a'_{qp} &= c^2 a_{qp} - s^2 a_{pq} + sc\,(a_{pp} - a_{qq})\,. \end{aligned}$$

When matrix $\mathbf{A}$ is *symmetric*, then these relations reduce to:

$$\begin{aligned}
a'_{ij} &= a_{ij}, \qquad \text{for } i,j \neq p,q,\\
a'_{pj} &= a'_{jp} = ca_{jp} - sa_{jq}, \qquad \text{for } j \neq p,q,\\
a'_{qj} &= a'_{jq} = sa_{jp} + ca_{jq}, \qquad \text{for } j \neq p,q,\\
a'_{pp} &= c^2 a_{pp} + s^2 a_{qq} - 2sca_{pq},\\
a'_{qq} &= s^2 a_{pp} + c^2 a_{qq} + 2sca_{pq},\\
a'_{pq} &= a'_{qp} = \left(c^2 - s^2\right) a_{pq} + sc\left(a_{pp} - a_{qq}\right).
\end{aligned}$$

The two-sided Givens rotations, preserving matrix similarity, are the basis of the Jacobi's method for finding eigenvalues and eigenvectors of a symmetric matrix – the idea is to transform to zero all the off-diagonal matrix elements, sequentially (see the next section). An algorithm of transformation of a symmetric matrix to a tridiagonal similar form can also be based on the Givens rotations – all elements except those on the diagonal, superdiagonal and subdiagonal are, sequentially, transformed to zero. The Givens rotations can also be used to construct an algorithm for the QR factorization (see Section 3.5.3).

3.5.2. Jacobi's method for finding eigenvalues of a symmetric matrix

The method finds all eigenvalues (and eigenvectors, when needed) of a symmetric matrix, applying a series of Givens similarity transformations:

$$\mathbf{A}'_{k+1} = \mathbf{G}_{pq}^{\mathrm{T}} \mathbf{A}_k \mathbf{G}_{pq},$$

which lead to a diagonal matrix with the eigenvalues on the diagonal.

In a single iteration, when an element a_{pq} is transformed to zero, only the elements with indexes p and q are changed (comp. Section 3.5.1):

$$\begin{aligned}
a'_{jp} &= ca_{jp} - sa_{jq}, \qquad j = 1,...,n, \quad j \neq p,q,\\
a'_{jq} &= ca_{jq} + sa_{jp}, \qquad j = 1,...,n, \quad j \neq p,q,\\
a'_{pp} &= c^2 a_{pp} + s^2 a_{qq} - 2sca_{pq},\\
a'_{qq} &= s^2 a_{pp} + c^2 a_{qq} + 2sca_{pq},\\
a'_{pq} &= \left(c^2 - s^2\right) a_{pq} + sc\left(a_{pp} - a_{qq}\right).
\end{aligned}$$

It is required that $a'_{pq} = 0$ (and from the symmetry $a'_{qp} = 0$), therefore

$$\left(c^2 - s^2\right) a_{pq} + sc\left(a_{pp} - a_{qq}\right) = 0.$$

Hence, we obtain

$$\mathrm{ctg} 2\theta = \frac{c^2 - s^2}{2sc} = \frac{a_{qq} - a_{pp}}{2a_{pq}}, \ |\theta| \leq \frac{\pi}{4}, \ \theta = \pm\frac{\pi}{4} \text{ for } a_{pp} = a_{qq}.$$

Denoting (see [3])

$$t = \operatorname{tg}\theta = \frac{s}{c},$$

$$\alpha = \operatorname{ctg} 2\theta,$$

we obtain the equation for t:

$$t^2 + 2\alpha t - 1 = 0.$$

We take the root with a smaller absolute value as a solution:

$$t = \frac{\operatorname{sgn}(\alpha)}{|\alpha| + \sqrt{\alpha^2+1}},$$

or

$$t = \frac{1}{2\alpha} \quad \text{when } \alpha \gg 1 \quad - \text{ to avoid an overflow.}$$

From the definitions, $c = \frac{1}{\sqrt{1+t^2}}$, $s = tc$. Denoting also $r = \frac{s}{1+c}$, we get more stable numerical formulae:

$$\begin{aligned}
a'_{jp} &= a_{jp} - s\left(a_{jq} + r a_{jp}\right), \qquad & j = 1, ..., n, \;\; j \neq p, q, \\
a'_{jq} &= a_{jq} + s\left(a_{jp} - r a_{jq}\right), \qquad & j = 1, ..., n, \;\; j \neq p, q, \\
a'_{pp} &= a_{pp} - t a_{pq}, \\
a'_{qq} &= a_{qq} + t a_{pq}.
\end{aligned}$$

If we define

$$S = \sum_{i,j=1, i\neq j}^{n} |a_{ij}|^2, \quad S' = \sum_{i,j=1, i\neq j}^{n} |a'_{ij}|^2,$$

then it can be shown that

$$S' = S - 2|a_{pq}|^2,$$

i.e., the sum of absolute values of all off-diagonal elements is smaller from step to step, thus the method is convergent. A matrix being the product of the rotation matrices converges to the matrix with columns being the eigenvectors of $\mathbf{A}$.

In the original Jacobi's method, at each step of the algorithm the indices p and q are selected as those corresponding to the off-diagonal element with a largest absolute value. A more practical is the *cyclic Jacobi's method*, where all off-diagonal elements are brought to zero in an a priori prescribed order – e.g., choosing elements when moving along subsequent rows of the matrix. Typical matrices require 6 to 10 cycles, the corresponding number of computations (additions and multiplications) is of order from $12n^3$ to $20n^3$. In practice, the Jacobi's method is effective for matrices not exceeding the dimension 10×10.

3.5.3. The QR matrix factorization using the Givens rotations

A matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ will be factorized. The following *notation* will be used:

x – elements not transformed,
$*$ – elements after transformation (changed).

First step: $n-1$ Givens rotations $\mathbf{G}_{1j}$, $j = 2, ..., n$, resulting in a transformation matrix $\mathbf{G}^{(1)}$ of the first step (a superscript indicates the step):

$$\mathbf{G}^{(1)}\mathbf{A} = \mathbf{G}_{1n} \cdots \mathbf{G}_{13}\mathbf{G}_{12}\mathbf{A}.$$

These rotations bring to zero all the elements of the first column, except the first one, sequentially, i.e., the second one, third one, etc.:

$$\mathbf{G}^{(1)}\mathbf{A} = \mathbf{G}_{1n} \cdots \mathbf{G}_{13}\left(\mathbf{G}_{12}\begin{bmatrix} \text{x} & \text{x} & \text{x} & \\ \text{x} & \text{x} & \text{x} & \\ \text{x} & \text{x} & \text{x} & \\ \text{x} & \text{x} & \text{x} & \cdots \\ \vdots & \vdots & \vdots & \\ \text{x} & \text{x} & \text{x} & \end{bmatrix}\right)$$

$$= \mathbf{G}_{1n} \cdots \mathbf{G}_{13}\begin{bmatrix} * & * & * & \\ 0 & * & * & \\ \text{x} & \text{x} & \text{x} & \\ \text{x} & \text{x} & \text{x} & \cdots \\ \vdots & \vdots & \vdots & \\ \text{x} & \text{x} & \text{x} & \end{bmatrix} = \cdots = \begin{bmatrix} * & * & * & \\ 0 & * & * & \\ 0 & * & * & \\ 0 & * & * & \cdots \\ \vdots & \vdots & \vdots & \\ 0 & * & * & \end{bmatrix}.$$

Second step: transformations of the elements of the second column located under the diagonal to zero, sequentially, starting from the third one:

$$\mathbf{G}^{(2)}\left(\mathbf{G}^{(1)}\mathbf{A}\right) = \mathbf{G}_{2n} \cdots \mathbf{G}_{24}\mathbf{G}_{23}\left(\mathbf{G}^{(1)}\mathbf{A}\right),$$

$$\mathbf{G}^{(2)}\left(\mathbf{G}^{(1)}\mathbf{A}\right) = \mathbf{G}_{2n}\cdots\mathbf{G}_{24}\mathbf{G}_{23}\left(\begin{bmatrix} \text{x} & \text{x} & \text{x} & \\ 0 & \text{x} & \text{x} & \\ 0 & \text{x} & \text{x} & \\ 0 & \text{x} & \text{x} & \cdots \\ \vdots & \vdots & \vdots & \\ 0 & \text{x} & \text{x} & \end{bmatrix}\right) = \begin{bmatrix} \text{x} & \text{x} & \text{x} & \\ 0 & * & * & \\ 0 & 0 & * & \\ 0 & 0 & * & \cdots \\ \vdots & \vdots & \vdots & \\ 0 & 0 & * & \end{bmatrix},$$

etc. in the next steps, until

$$\mathbf{G}^{(n-1)} \cdots \mathbf{G}^{(2)}\mathbf{G}^{(1)}\mathbf{A} = \bar{\mathbf{Q}}\mathbf{A} = \mathbf{R},$$

where the matrix $\mathbf{R}$ is upper triangular and the matrix $\bar{\mathbf{Q}}$ is orthogonal, therefore $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q} = \bar{\mathbf{Q}}^{\mathrm{T}}$ is orthogonal.

The number of computations needed to evaluate the matrix the $\mathbf{R}$: $\mathrm{M} = O(\frac{4}{3}n^3)$, $\mathrm{A} = O(\frac{2}{3}n^3)$ and $O(\frac{1}{2}n^2)$ calculations of square roots.

3.6. Householder transformation, with applications[*]

3.6.1. Householder transformation (reflection)

The Householder transformation (Householder reflection) is defined by the following matrix $\mathbf{P}$:

$$\mathbf{P} = \mathbf{I} - 2\mathbf{w}\mathbf{w}^{\mathrm{T}}, \qquad \text{where } \mathbf{w} \in \mathbb{R}^n,\ \|\mathbf{w}\| = 1.$$

We have immediately the following properties:

$$\begin{aligned} \mathbf{P} &= \mathbf{P}^{\mathrm{T}} \quad (\text{symmetry}), \\ \mathbf{P}\mathbf{P}^{\mathrm{T}} &= \mathbf{I} \quad (\text{i.e., } \mathbf{P} = \mathbf{P}^{-1}, \text{orthogonality}), \end{aligned}$$

because

$$\left(\mathbf{I} - 2\mathbf{w}\mathbf{w}^{\mathrm{T}}\right)\left(\mathbf{I} - 2\mathbf{w}\mathbf{w}^{\mathrm{T}}\right) = \mathbf{I} - 4\mathbf{w}\mathbf{w}^{\mathrm{T}} + 4\mathbf{w}\left(\mathbf{w}^{\mathrm{T}}\mathbf{w}\right)\mathbf{w}^{\mathrm{T}} = \mathbf{I}.$$

Geometrical interpretation:

$$\mathbf{y} = \mathbf{P}\mathbf{a} = \left(\mathbf{I} - 2\mathbf{w}\mathbf{w}^{\mathrm{T}}\right)\mathbf{a} = \mathbf{a} - 2\left(\mathbf{w}^{\mathrm{T}}\mathbf{a}\right)\mathbf{w} = \mathbf{a} - 2\mathbf{r},$$

where $\mathbf{r}$ is the orthogonal projection of $\mathbf{a}$ onto $\mathbf{w}$. That is, $\mathbf{Pa}$ is a mirror image of $\mathbf{a}$ with respect to a plane orthogonal to the vector $\mathbf{w}$, see Fig. 3.2. Appropriately choosing $\mathbf{w}$ we can obtain any desired location (direction) of the vector $\mathbf{y} = \mathbf{Pa}$.

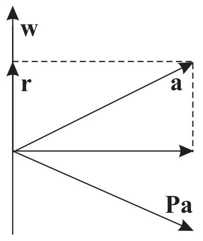


Figure 3.2. Geometrical interpretation of the Householder transformation (the Householder reflection)

[*]Optional.

A most frequent application of the Householder transformation is to obtain, for a given vector $\mathbf{a}$, a vector $\mathbf{Pa}$ parallel to the versor $\mathbf{e}_1 = [1\ 0\ 0\ ...0]^{\mathrm{T}}$. To this end, the matrix $\mathbf{P}$ is represented as follows:

$$\mathbf{P} = \mathbf{I} - \frac{1}{K}\mathbf{u}\mathbf{u}^{\mathrm{T}}, \tag{3.42}$$

where
$$\mathbf{u} = \mathbf{a} \pm \|\mathbf{a}\|\,\mathbf{e}_1, \quad K = \frac{1}{2}\|\mathbf{u}\|^2,$$

where the sign is chosen as the one maximizing the norm of $\mathbf{u}$, i.e.,

$$\mathbf{u} = \mathbf{a} + \operatorname{sgn}(a_1)\,\|\mathbf{a}\|\,\mathbf{e}_1,$$

where
$$a_1 = (\mathbf{e}_1)^{\mathrm{T}}\mathbf{a}.$$

Then
$$\begin{aligned}
\mathbf{Pa} &= \mathbf{a} - \frac{\mathbf{u}}{K}\left(\mathbf{a} \pm \|\mathbf{a}\|\,\mathbf{e}_1\right)^{\mathrm{T}}\mathbf{a} \\
&= \mathbf{a} - \frac{2\mathbf{u}\left(\|\mathbf{a}\|^2 \pm \|\mathbf{a}\|\,a_1\right)}{\|\mathbf{a} \pm \|\mathbf{a}\|\,\mathbf{e}_1\|^2} \\
&= \mathbf{a} - \frac{2\mathbf{u}\left(\|\mathbf{a}\|^2 \pm \|\mathbf{a}\|\,a_1\right)}{\|\mathbf{a}\|^2 + \|\mathbf{a}\|^2 \pm 2\,\|\mathbf{a}\|\,a_1} \\
&= \mathbf{a} - \mathbf{u} \\
&= \mp\|\mathbf{a}\|\,\mathbf{e}_1.
\end{aligned}$$

In practice, the elements of the matrix $\mathbf{P}$ should not be computed, but only the product of this matrix and the vector $\mathbf{a}$:

$$\mathbf{Pa} = \left(\mathbf{I} - 2\mathbf{w}\mathbf{w}^{\mathrm{T}}\right)\mathbf{a} = \mathbf{a} - (2\mathbf{w}^{\mathrm{T}}\mathbf{a})\mathbf{w}. \tag{3.43}$$

Using (3.42), we can write this equation in the form

$$\mathbf{Pa} = \left(\mathbf{I} - \frac{1}{K}\mathbf{u}\mathbf{u}^{\mathrm{T}}\right)\mathbf{a} = \mathbf{a} - \left(\frac{\mathbf{u}^{\mathrm{T}}\mathbf{a}}{K}\right)\mathbf{u}. \tag{3.44}$$

In applications, the Householder transformation is often applied to a whole matrix (see Section 3.6.2), and the results of the transformation of subsequent columns of the matrix are calculated.

In practice, a two-sided Householder transformation is also used, to obtain matrices preserving the similarity property (see, e.g., Section 3.6.3), i.e.,

$$\mathbf{A}' = \mathbf{PAP}.$$

Also in this case, the matrix multiplications should not be performed in practice, but the results should be calculated in a simpler way. If we define

$$\mathbf{r} = \frac{\mathbf{Au}}{K}, \quad \mathbf{s}^{\mathrm{T}} = \frac{\mathbf{u}^{\mathrm{T}}\mathbf{A}}{K}, \quad h = \frac{\mathbf{u}^{\mathrm{T}}\mathbf{r}}{2K},$$

then we have

$$\mathbf{AP} = \mathbf{A}\left(\mathbf{I} - \frac{\mathbf{uu}^{\mathrm{T}}}{K}\right) = \mathbf{A} - \mathbf{ru}^{\mathrm{T}}.$$

Therefore,

$$\begin{aligned}\mathbf{A}' = \mathbf{PAP} &= \left(\mathbf{I} - \frac{\mathbf{uu}^{\mathrm{T}}}{K}\right)\left(\mathbf{A} - \mathbf{ru}^{\mathrm{T}}\right) \\ &= \mathbf{A} - \mathbf{ru}^{\mathrm{T}} - \mathbf{us}^{\mathrm{T}} + 2h\mathbf{uu}^{\mathrm{T}} \\ &= \mathbf{A} - (\mathbf{r} - h\mathbf{u})\,\mathbf{u}^{\mathrm{T}} - \mathbf{u}\,(\mathbf{s} - h\mathbf{u})^{\mathrm{T}}. \end{aligned} \tag{3.45}$$

For *symmetric matrices* we have, additionally,

$$\mathbf{s} = \frac{\mathbf{A}^{\mathrm{T}}\mathbf{u}}{K} = \frac{\mathbf{Au}}{K} = \mathbf{r}$$

and the final formula can be simplified to

$$\mathbf{A}' = \mathbf{A} - (\mathbf{r} - h\mathbf{u})\,\mathbf{u}^{\mathrm{T}} - \mathbf{u}\,(\mathbf{r} - h\mathbf{u})^{\mathrm{T}}. \tag{3.46}$$

The Householder transformation is a numerically correct algorithm.

3.6.2. The QR matrix factorization using the Householder reflections

Let a square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ be given. The following *notation* will be used in the considerations to follow:

x – elements of the vector to be transformed to the prescribed direction (usually $\mathbf{e}_1$),
x – other matrix elements before a transformation,
$*$ – matrix elements after a transformation (the transformed elements).

First step: the first column of $\mathbf{A}$ is reflected to the direction $\mathbf{e}_1$:

$$\mathbf{P} = \mathbf{P}^{(1)}, \quad \mathbf{P}^{(1)}\mathbf{A} = \mathbf{P}^{(1)}\begin{bmatrix} x & \mathrm{x} & \mathrm{x} & & \\ x & \mathrm{x} & \mathrm{x} & & \\ x & \mathrm{x} & \mathrm{x} & \cdots & \\ \vdots & \vdots & \vdots & & \\ x & \mathrm{x} & \mathrm{x} & & \end{bmatrix} = \begin{bmatrix} * & * & * & & \\ 0 & * & * & & \\ 0 & * & * & \cdots & \\ \vdots & \vdots & \vdots & & \\ 0 & * & * & & \end{bmatrix}.$$

Second step: We proceed as in the first step, but this time with the first column of a square submatrix of dimension $n-1$, obtained after deleting the first column and row. The transformation matrix is $\mathbf{P}^{(2)}$ and we have

$$\mathbf{P}^{(2)}\left(\mathbf{P}^{(1)}\mathbf{A}\right)=\begin{bmatrix} 1 & 0\ 0 & \cdots \\ 0 & & \\ 0 & \widetilde{\mathbf{P}}^{(2)}_{(n-1)} & \\ \vdots & & \\ 0 & & \end{bmatrix}\begin{bmatrix} \mathrm{x} & \mathrm{x} & \mathrm{x} & \\ 0 & x & \mathrm{x} & \\ 0 & x & \mathrm{x} & \cdots \\ \vdots & \vdots & \vdots & \\ 0 & x & \mathrm{x} & \end{bmatrix}=\begin{bmatrix} \mathrm{x} & \mathrm{x} & \mathrm{x} & \\ 0 & * & * & \\ 0 & 0 & * & \cdots \\ \vdots & \vdots & \vdots & \\ 0 & 0 & * & \end{bmatrix},$$

where $\widetilde{\mathbf{P}}^{(2)}_{(n-1)}$ is the Householder transformation matrix for the vector of dimension $n-1$, transforming first column of $(n-1)$-dimensional submatrix of $\mathbf{P}^{(1)}\mathbf{A}$ onto the direction of the first versor of the space $\mathbb{R}^{n-1}$.

Next steps are performed in a similar way, finally we obtain

$$\mathbf{P}^{(n-1)}\mathbf{P}^{(n-2)}\cdots\mathbf{P}^{(2)}\mathbf{P}^{(1)}\mathbf{A}=\mathbf{Q}^{-1}\mathbf{A}=\mathbf{R},$$

where $\mathbf{R}$ is an upper-triangular matrix. Thus we get

$$\mathbf{A}=\mathbf{QR},$$
$$\mathbf{Q}=\mathbf{P}^{(1)}\mathbf{P}^{(2)}\cdots\mathbf{P}^{(n-2)}\mathbf{P}^{(n-1)},$$

where $\mathbf{R}$ is upper-triangular. Moreover, the matrix $\mathbf{Q}$ is orthogonal, because

$$\mathbf{P}^{(i)}\left(\mathbf{P}^{(i)}\right)^{\mathrm{T}}=\mathbf{P}^{(i)}\mathbf{P}^{(i)}=\mathbf{I},\quad i=1,...,n-1.$$

The number of computations for calculation of the matrix $\mathbf{R}$ is M,A= $O(\frac{2}{3}n^3)$ and $n-1$ square root calculations. To get explicit form of the matrix $\mathbf{Q}$, additional calculations of order n^3 are needed.

The presented algorithm can be, certainly, applied also to the QR factorization of a non-square matrix $\mathbf{A}\in\mathbb{R}^{m\times n}$, $m>n$.

3.6.3. Transformation of a matrix to the Hessenberg form using the Householder reflections, preserving matrix similarity

A square matrix has the Hessenberg form $\mathbf{H}$ if it is almost upper-triangular, i.e., (asterisks denote positions with a possibility of a non-zero element),

$$\mathbf{H}=\begin{bmatrix} * & * & * & * & \cdots & * \\ * & * & * & * & \cdots & * \\ 0 & * & * & * & \cdots & * \\ 0 & 0 & * & * & \cdots & * \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & * & * \end{bmatrix}.$$

We shall apply the Householder reflections, preserving similarity, i.e., twice – from both sides. The way of proceeding from the left side will be similar to that used in Section 3.6.2, but the transformed vectors (subvectors) will have the dimension lower by one.

First step: elements from first column of the matrix $\mathbf{A} = \mathbf{A}^{(1)}$ will be transformed to zero, starting from the third element.

To this end, a Householder matrix $\widetilde{\mathbf{P}}^{(1)} \in \mathbb{R}^{(n-1)\times(n-1)}$ is constructed, which reflects the vector $[a_{21}\ a_{31}\ \cdots\ a_{n1}]^{\mathrm{T}}$ onto the direction of the first versor of the space $\mathbb{R}^{n-1}$, i.e., to the form $[*\ 0\ \cdots\ 0]^{\mathrm{T}} \in \mathbb{R}^{n-1}$. Next, a transformation matrix of the first step is constructed, $\mathbf{P}^{(1)} \in \mathbb{R}^{n\times n}$, in the form

$$\mathbf{P}^{(1)} = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{P}}^{(1)} \end{bmatrix},$$

and two-sided transformation is performed

$$\mathbf{A}^{(2)} = \mathbf{P}^{(1)}\mathbf{A}^{(1)}\mathbf{P}^{(1)}.$$

It is easy to check that a left-side multiplication by $\mathbf{P}^{(1)}$ transforms appropriately the first column of $\mathbf{A}^{(1)}$ (not affecting the first row), and a right-side multiplication does not change this column, due to the structure of the matrix $\mathbf{P}^{(1)}$. Therefore, the matrix $\mathbf{A}^{(2)}$ is of a required form,

$$\mathbf{A}^{(2)} = \begin{bmatrix} * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & * & \dots & * & * \end{bmatrix}.$$

Second step: The elements of second column of the matrix $\mathbf{A}^{(2)}$ are transformed to zero, starting from the fourth element.

To this end, the matrix $\widetilde{\mathbf{P}}^{(2)} \in \mathbb{R}^{(n-2)\times(n-2)}$ is constructed, which reflects the subvector $[a_{32}^{(2)}\ a_{42}^{(2)}\ \cdots\ a_{n2}^{(2)}]^{\mathrm{T}}$ of the second column of the matrix $\mathbf{A}^{(2)}$ onto the direction of the first versor of the space $\mathbb{R}^{n-2}$, i.e., to the form $[*\ 0\ \cdots\ 0]^{\mathrm{T}} \in \mathbb{R}^{n-2}$. Next, a transformation matrix $\mathbf{P}^{(2)} \in \mathbb{R}^{n\times n}$ of the second step is constructed, in the form

$$\mathbf{P}^{(2)} = \begin{bmatrix} \mathbf{I}_2 & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{P}}^{(2)} \end{bmatrix},$$

where $\mathbf{I}_2$ is a two-dimensional unity matrix, and the two-sided transformation is performed:

$$\mathbf{A}^{(3)} = \mathbf{P}^{(2)}\mathbf{A}^{(1)}\mathbf{P}^{(2)}.$$

It is easy to check that a multiplication by $\mathbf{P}^{(2)}$ from the left side transforms appropriately the second column of $\mathbf{A}^{(1)}$, not affecting the first column (and the first two rows, too), and a subsequent multiplication from the right side does not change the first two columns – due to the structure of the matrix $\mathbf{P}^{(2)}$. Therefore, the resulting matrix is in the form

$$\mathbf{A}^{(3)} = \begin{bmatrix} * & * & * & * & \cdots & * \\ * & * & * & * & \cdots & * \\ 0 & * & * & * & \cdots & * \\ 0 & 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & * & \ldots & * & * \end{bmatrix}.$$

We proceed in an analogous way in the **third step**, i.e., we construct the matrix $\widetilde{\mathbf{P}}^{(3)} \in \mathbb{R}^{(n-3)\times(n-3)}$ which transforms to zero the elements of the third column of the matrix $\mathbf{A}^{(3)}$, starting from the fifth element, and next, the transformation matrix $\mathbf{P}^{(3)}$ of an analogous structure (with the submatrix $\mathbf{I}_3$ in the left upper corner).

We proceed in an analogous way in **next steps**, obtaining finally

$$\mathbf{A}^{(n-1)} = \mathbf{P}^{(n-2)}\mathbf{P}^{(n-3)} \ldots \mathbf{P}^{(2)}\mathbf{P}^{(1)}\mathbf{A}\mathbf{P}^{(1)}\mathbf{P}^{(2)} \ldots \mathbf{P}^{(n-3)}\mathbf{P}^{(n-2)} = \mathbf{H}.$$

Remark. All elements of the matrices $\widetilde{\mathbf{P}}^{(i)}$ should not be calculated, only the results of multiplications of subsequent vectors (columns) of the transformed matrices by these matrices, applying appropriately the relation (3.45).

The number of additions and multiplications is of order $\frac{5}{3}n^3$, it can be reduced to $\frac{2}{3}n^3$ for symmetric matrices.

Problems

1. Using the standard Gram-Schmidt ortogonalization algorithm calculate, sequentially, the QR factorizations (3.7), (3.8) and (3.9) of the matrix

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 1 & 4 \\ 1 & 3 & 4 \\ 1 & 1 & 2 \end{bmatrix}.$$

2. Calculate the eigenvalues and the eigenvectors of the following matrices:

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \begin{bmatrix} 1 & 2 & 0 \\ 0 & 2 & 0 \\ -2 & -2 & -1 \end{bmatrix}.$$

Check for the above matrices, that a matrix $\mathbf{V}$ with columns being orthogonal eigenvectors of the matrix $\mathbf{A}$, transforms $\mathbf{A}$ to the diagonal form, by the similarity transformation.

3. Show that the determinant of a symmetric and nonsingular matrix is equal to the product of its eigenvalues.
4. The Cayley-Hamilton's Theorem: Every square matrix $\mathbf{A}$ is a root of its characteristic polynomial $w(\lambda)$, i.e., $w(\mathbf{A}) = \mathbf{0}$.
   Applying this theorem, formulate a method for finding the inverse matrix (hint: multiply $w(\mathbf{A})$ by $\mathbf{A}^{-1}$).
5. Calculate the SVD decomposition and the pseudo-inverse matrix for

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

6. Prove that the condition number of a square nonsingular matrix, in the 2-norm, is equal to the ratio of its largest and smallest singular values (hint: applying the SVD decomposition show that singular values of the inverse matrix are inverses of singular values of the original matrix).
7. Find solution $\hat{\mathbf{x}}$ of the system of equations $\mathbf{Ax} = \mathbf{b}$, where

$$\mathbf{A} = \begin{bmatrix} 1 & 3.5 & 3 \\ 1 & -0.5 & 1 \\ 1 & 3.5 & 7 \\ 1 & -0.5 & -3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 17.5 \\ 1.5 \\ 25.5 \\ -6.5 \end{bmatrix},$$

   minimizing the Euclidean norm of the solution error vector $\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}$:
   a) using the QR factorization of the matrix $\mathbf{A}$ (calculating and using a most useful factorization form, from those given in the Theorem 3.1),
   b) using the set of normal equations.

Chapter 4

Approximation

Assume there are given values of a function $f(x)$ in a given interval, at every point or at a finite number of points. The aim of the approximation is to find a simpler function $F(x)$, from a chosen class of approximating functions, which is appropriately close to $f(x)$. The approximation of a continuous function over a finite interval can be applied when the original function $f(x)$ is too complicated to be effectively used in certain analysis or design methods. On the other hand, the approximation of functions known or calculated at a given finite number of points only is widely used in modeling techniques.

Denote:

$f\,(x)$ – an original function, to be approximated (unknown, or known but to be approximated by a simpler function),

$F\,(x)$ – an approximating function.

Assume that:

X – a linear function space, $f \in X$,

X_n – a (n+1)-dimensional subspace of X, with a basis $\phi_0\,(x)\,, ..., \phi_n\,(x)$, i.e.,

$$F\,(x) \in X_n \quad \Leftrightarrow \quad F\,(x) = a_0\phi_0\,(x) + a_1\phi_1\,(x) + ... + a_n\phi_n\,(x)\,.$$

where $a_i \in \mathbb{R},\ i = 0, 1, ..., n,$ are the coefficients.

The *approximation problem* can be defined in the following way: to find a function $F^* \in X_n$ closest to f in a certain sense, usually in the sense of a certain distance $\delta(f-F)$ defined by a norm $\|\cdot\|$,

$$\forall F \in X_n \quad \delta(f-F^*) \overset{\text{df}}{=} \|f - F^*\| \leq \|f - F\|\,.$$

Thus, approximation of the function f means finding the coefficients $a_0, ..., a_n$ of F such that the norm $\|f - F\|$ is minimized.

Typical examples, resulting from a choice of different norms:

– *A uniform continuous approximation* of a continuous function $f(x)$ defined on a closed interval $[a, b\,]$:

$$\|F - f\| = \sup_{x\in[a,b]} |F\,(x) - f\,(x)| \qquad \text{(Tschebyschev norm).} \qquad (4.1)$$

– *A continuous least-squares approximation* of a function $f(x)$, quadratically integrable over a closed interval $[a, b]$, i.e., $f(x) \in L_p^2[a, b]$:

$$\|F - f\| = \sqrt{\int_a^b p(x) \left[F(x) - f(x)\right]^2 dx}, \tag{4.2}$$

where $p(x)$ is a weighting function.

– *A uniform discrete approximation* of a function $f(x)$, known on a finite set of $N+1$ points only:

$$\|F - f\| = \max\{|F(x_0) - f(x_0)|, |F(x_1) - f(x_1)|, \cdots, |F(x_N) - f(x_N)|\}. \tag{4.3}$$

– *A discrete least-squares approximation* (the least-squares method) of a function $f(x)$ known on a finite set of $N + 1$ points only:

$$\|F - f\| = \sqrt{\sum_{j=0}^{N} p(x_j) \left[F(x_j) - f(x_j)\right]^2}, \tag{4.4}$$

where $p(x)$ is a weighting function.

A possible difference between two continuous approximations of a continuous function $f(x)$ is illustrated in Fig. 4.1, for the uniform and least-squares approximations. An approximation of a curve by a zero-order polynomial (a straight horizontal line) is performed. The straight line $F_c(x)$ results from the uniform approximation – maximal positive and negative deviations of the curve from the approximating line are equal, which results in the minimal Tschebyschev norm (4.1)). On the other hand, the straight line $F_s(x)$ results from the least-squares approximation

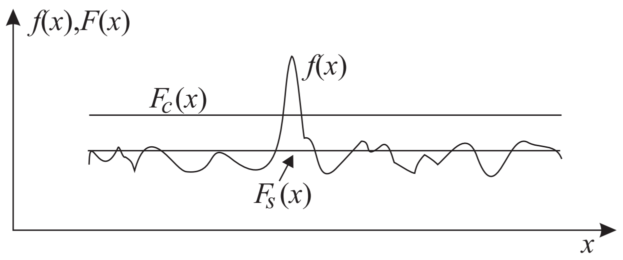


Figure 4.1. Geometric interpretation of a difference between two continuous approximations of a function $f(x)$ using a polynomial of zero order: a uniform approximation $F_c(x)$ and a least-squares approximation $F_s(x)$

– from minimizing the integral of the squared difference $f(x) - F_s(x)$. In the first case, the *maximal deviation* counts only, whereas in the second case, *all deviations* count, regardless of the sign and value.

4.1. Discrete least-squares approximation

Assume that for a given finite number of points, $x_0, x_1, ..., x_N$ $(x_i \neq x_j)$, the values $y_j = f(x_j)$, $j = 0, 1, 2, ..., N$, are known.

Let $\phi_i(x)$, $i = 0, 1, ..., n$, be a basis of a space $X_n \subseteq X$ of interpolating functions, i.e.,

$$\forall F \in X_n \quad F(x) = \sum_{i=0}^{n} a_i \phi_i(x) . \tag{4.5}$$

The approximation problem:
to find values of the parameters $a_0, a_1, ..., a_n$ defining the approximating function (4.5), which minimize the least-squares error defined by

$$H(a_0, ..., a_n) \overset{\text{df}}{=} \sum_{j=0}^{N} \left[f(x_j) - \sum_{i=0}^{n} a_i \phi_i(x_j) \right]^2 . \tag{4.6}$$

In the above formulation, the weighting function $p(\cdot)$ is not present, to simplify the presentation.

The formula for the coefficients $a_0, a_1, ..., a_n$ can be derived from the necessary condition for a minimum (being here also the sufficient condition, as the function is convex):

$$\frac{\partial H}{\partial a_k} = -2 \sum_{j=0}^{N} \left[f(x_j) - \sum_{i=0}^{n} a_i \phi_i(x_j) \right] \cdot \phi_k(x_j) = 0, \qquad k = 0, ..., n,$$

i.e.,

$$a_0 \sum_{j=0}^{N} \phi_0(x_j) \cdot \phi_0(x_j) + a_1 \sum_{j=0}^{N} \phi_1(x_j) \cdot \phi_0(x_j) + \cdots + a_n \sum_{j=0}^{N} \phi_n(x_j) \cdot \phi_0(x_j)$$
$$= \sum_{j=0}^{N} f(x_j) \cdot \phi_0(x_j) ,$$

$$a_0 \sum_{j=0}^{N} \phi_0(x_j) \cdot \phi_1(x_j) + a_1 \sum_{j=0}^{N} \phi_1(x_j) \cdot \phi_1(x_j) + \cdots + a_n \sum_{j=0}^{N} \phi_n(x_j) \cdot \phi_1(x_j)$$
$$= \sum_{j=0}^{N} f(x_j) \cdot \phi_1(x_j) ,$$

$$\vdots$$

$$\vdots$$

$$a_0 \sum_{j=0}^{N} \phi_0(x_j)\cdot\phi_n(x_j) + a_1 \sum_{j=0}^{N} \phi_1(x_j)\cdot\phi_n(x_j) + \cdots + a_n \sum_{j=0}^{N} \phi_n(x_j)\cdot\phi_n(x_j) = \sum_{j=0}^{N} f(x_j)\cdot\phi_n(x_j).$$

The above system of linear equations with the unknowns $a_0, ..., a_n$ is called the set of *normal equations*, and its matrix is known as *the Gram's matrix*. The normal equations can be written in a much simpler form if the scalar product is defined:

$$\langle \phi_i, \phi_k \rangle \overset{\text{df}}{=} \sum_{j=0}^{N} \phi_i(x_j)\,\phi_k(x_j). \tag{4.7}$$

The set of normal equations takes then a much simpler form:

$$\begin{bmatrix} \langle\phi_0,\phi_0\rangle & \langle\phi_1,\phi_0\rangle & \cdots & \langle\phi_n,\phi_0\rangle \\ \langle\phi_0,\phi_1\rangle & \langle\phi_1,\phi_1\rangle & \cdots & \langle\phi_n,\phi_1\rangle \\ \vdots & \vdots & \vdots & \vdots \\ \langle\phi_0,\phi_n\rangle & \langle\phi_1,\phi_n\rangle & \cdots & \langle\phi_n,\phi_n\rangle \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \langle\phi_0,f\rangle \\ \langle\phi_1,f\rangle \\ \vdots \\ \langle\phi_n,f\rangle \end{bmatrix}. \tag{4.8}$$

Let us define the following matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \cdots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_n(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_n(x_N) \end{bmatrix}, \tag{4.9}$$

define also

$$\begin{aligned} \mathbf{a} &= [a_0\, a_1 \cdots a_n]^{\mathrm{T}}, \\ \mathbf{y} &= [y_0\, y_1 \cdots y_N]^{\mathrm{T}}, \quad y_j = f(x_j), \;\; j = 0, 1, ..., N. \end{aligned}$$

The performance function (4.6) of the approximation problem can now be written as

$$H(\mathbf{a}) = (\| \mathbf{y} - \mathbf{A}\mathbf{a} \|_2)^2. \tag{4.10}$$

Therefore, the problem of the least-squares approximation is a linear least-squares problem (LLSP). Notice that all columns of the matrix $\mathbf{A}$ are linearly independent (which follows from linear independence of the basis functions). Hence, the matrix $\mathbf{A}$ has full rank.

Utilizing the definition (4.9) of the matrix $\mathbf{A}$, the set of normal equations (4.8) can be written in the form

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\,\mathbf{a} = \mathbf{A}^{\mathrm{T}}\mathbf{y}. \tag{4.11}$$

Because the matrix $\mathbf{A}$ has full rank, then the Gram's matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is nonsingular. This implies uniqueness of the solution of the set of normal equations. But, even being nonsingular, the matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ can be badly conditioned – its condition number is a square of the condition number of $\mathbf{A}$. In this case, it is recommended to solve the approximation problem using the method based on the QR factorization of $\mathbf{A}$, see the methods for solving the LLSP in Section 3.4.

Example 4.1. Consider the problem of a discrete least-squares approximation in a two-dimensional ($n = 1$) function basis consisting of the following basis functions:

$$\phi_0(x) = x, \quad \phi_1(x) = e^x,$$

for the data set

x_j	-2	-1	0	1	2
y_j	-3	-1.2	0.2	3	7.5

.

We have 5 points, i.e., N=4. First, let us calculate elements of the Gram's matrix and of the vector of the right-hand side of the normal equations:

$$\begin{aligned}
\langle\phi_0,\phi_0\rangle &= \sum_{j=0}^{4} x_j x_j = 4+1+0+1+4 = 10,\\
\langle\phi_0,\phi_1\rangle &= \sum_{j=0}^{4} x_j e^{x_j} = -2e^{-2} - e^{-1} + 0 + e^1 + 2e^2 = 16.8578,\\
\langle\phi_1,\phi_0\rangle &= \langle\phi_0,\phi_1\rangle = 16.8578,\\
\langle\phi_1,\phi_1\rangle &= \sum_{j=0}^{4} e^{x_j} e^{x_j} = (e^{-2})^2 + (e^{-1})^2 + (e^0)^2 + (e^1)^2 + (e^2)^2 = 63.1409,\\
\langle\phi_0,f\rangle &= \sum_{j=0}^{4} x_j y_j = 6 + 1.2 + 0 + 3 + 15 = 25.2,\\
\langle\phi_1,f\rangle &= \sum_{j=0}^{4} e^{x_j} y_j = -3e^{-2} - 1.2e^{-1} + 0.2e^0 + 3e^1 + 7.5e^2 = 62.9253.
\end{aligned}$$

Thus, we obtain the following normal equations:

$$\begin{bmatrix} 10 & 16.8578 \\ 16.8578 & 63.1409 \end{bmatrix}\begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 25.2 \\ 62.9253 \end{bmatrix},$$

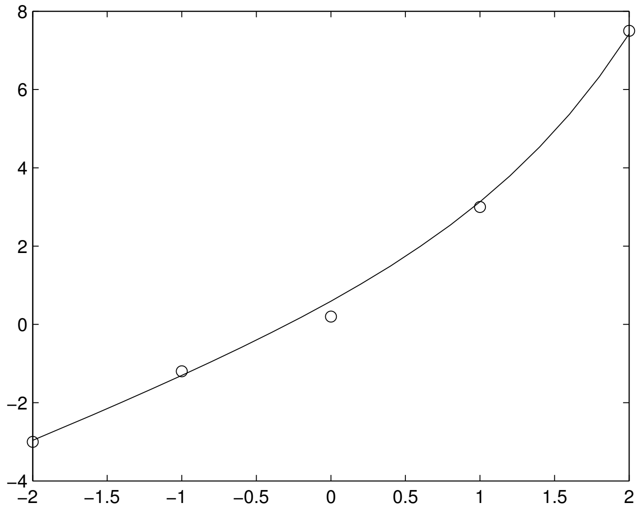


Figure 4.2. Location of the data points and the approximation function in Example 4.1

with the solution vector $[a_0 \ \ a_1]^{\mathrm{T}} = [1.5213 \ \ 0.5925]^{\mathrm{T}}$. Therefore, the approximation function is

$$F(x) = 1.5213x + 0.5925e^{x}.$$

Figure 4.2 presents the data (the circles) and the approximation function. □

4.1.1. Polynomial approximation

Polynomials $W_n(x)$ are often used as approximating functions (n will denote the order of a polynomial). The justification of this fact is the classic Weierstrass Theorem, about a uniform approximation of a continuous function $f(x)$ on a closed interval $[a, b]$ by an algebraic polynomial. The statement of this theorem can be as follow:

$$\forall \varepsilon > 0 \quad \exists n \quad \forall x \in [a, b] \quad |f(x) - W_n(x)| \leq \varepsilon. \tag{4.12}$$

The order n of the approximating polynomial is usually much lower than the number of points, at which the values of the original function are given, i.e.,

$$N \gg n.$$

Consider the following *natural polynomial basis (the power basis)*:

$$\phi_0(x) = 1, \ \ \phi_1(x) = x, \ \phi_2(x) = x^2, \ \ \cdots, \ \ \phi_n(x) = x^n, \tag{4.13}$$

i.e.,

$$F(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n. \tag{4.14}$$

Introducing auxiliary definitions:

$$g_{ik} \overset{\mathrm{df}}{=} \sum_{j=0}^{N} (x_j)^i (x_j)^k = \sum_{j=0}^{N} (x_j)^{i+k},$$

$$\varrho_k \overset{\mathrm{df}}{=} \sum_{j=0}^{N} f(x_j)(x_j)^k,$$

we obtain the system of normal equations in the following form:

$$\begin{array}{l} a_0 g_{00} + a_1 g_{10} + ... + a_n g_{n0} = \varrho_0 \\ a_0 g_{01} + a_1 g_{11} + ... + a_n g_{n1} = \varrho_1 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_0 g_{0n} + a_1 g_{1n} + ... + a_n g_{nn} = \varrho_n \end{array} \quad \equiv \quad \mathbf{G} \cdot \mathbf{a} = \varrho. \tag{4.15}$$

Example 4.2. For the following set of data:

x_j	0	0.2	0.4	0.6	0.8	1.0
y_j	1.15	0.7	0.5	0.4	0.25	0.2

,

we shall find the approximating polynomial of order: a) one; b) two. The approximation error, defined by the maximum norm, will be also evaluated.

a) After calculation of the coefficients g_{ij} and ρ_i, we obtain the following normal equations:

$$\begin{bmatrix} 6 & 3 \\ 3 & 2.2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 3.2 \\ 0.98 \end{bmatrix},$$

which lead to the following approximating polynomial of first order:

$$w_1(x) = 0.976 - 0.886x.$$

b) After calculation of the coefficients g_{ij} and ρ_i, we obtain the following normal equations:

$$\begin{bmatrix} 6 & 3 & 2.2 \\ 3 & 2.2 & 1.8 \\ 2.2 & 1.8 & 1.566 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 3.2 \\ 0.98 \\ 0.612 \end{bmatrix},$$

and, after finding the solution, the following approximating polynomial of second order:

$$w_2(x) = 1.11 - 1.886x + x^2.$$

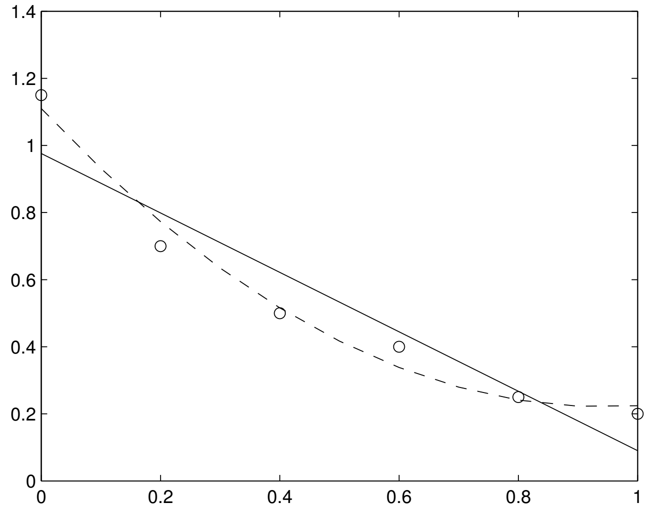


Figure 4.3. Location of the data and the approximating polynomials in Example 4.2

The errors of the approximation are as follows:

x_j	0	0.2	0.4	0.6	0.8	1.0	error
$w_1(x_j)-y_j$	-0.174	0.103	0.130	0.0564	0.033	-0.09	0.174
$w_2(x_j)-y_j$	-0.04	0.073	0.0156	-0.062	-0.009	0.024	0.073

Figure 4.3 presents the data (circles) and the approximating polynomials: of the first order (solid line) and of the second order (dashed line). □

Remark. The Gram's matrix **G** for the systems of normal equations obtained for the approximating polynomials with the natural basis ($\phi_0(x) = 1,\ \phi_1(x) = x,\ \cdots$ $\cdots,\ \phi_n(x) = x^n$) quickly becomes ill-conditioned with an increase of n (for $n > 5$ is usually ill-conditioned).

The explanation is quite easy. Let $x \in [0,1],\ x = 0 + jh,\ \ j = 0,1,2,...,N,$ $h = 1/N$. For sufficiently large values of N the following approximation is quite accurate:

$$g_{ik} = (N+1)\frac{1}{N+1}\sum_{j=0}^{N}(x_j)^{i+k} \approx (N+1)\int_0^{1+\frac{1}{N}} x^{i+k}dx$$

$$= (N+1)\frac{(1+\frac{1}{N})^{i+k+1}}{i+k+1} \approx (N+1)\frac{1^{i+k+1}}{i+k+1} = (N+1)\frac{1}{i+k+1}.$$

Applying this approximation, we get

$$\mathbf{G} = (N+1)\begin{bmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n+1} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n+2} \\ \vdots & \vdots & \ddots & \\ \frac{1}{n+1} & \frac{1}{n+2} & & \frac{1}{2n+1} \end{bmatrix} = (N+1)\cdot \mathbf{G}_N.$$

It can be easily seen now that the matrix $\mathbf{G}_N$ is of a Hilbert type, well known to become quickly very ill-conditioned with the increase of n.

As a result, approximations based on the natural polynomial basis (power basis) can be, in practice, applied only for small values of n, usually not exceeding $n \approx 4$. The ill-conditioning can be avoided when the applied basis consists of orthogonal polynomials.

4.1.2. Approximation using an orthogonal function basis

Functions $h(x)$ and $g(x)$ are orthogonal on a set of points $x_0, x_1, ..., x_N$, if the scalar product of these functions, as defined by (4.7), is zero, ie.,

$$\langle h(x), g(x)\rangle = 0 \quad \Leftrightarrow \quad \sum_{j=0}^{N} h(x_j)\, g(x_j) = 0.$$

Thus, the basis functions $\psi_0, ..., \psi_n$ are mutually *orthogonal*, if

$$\sum_{j=0}^{N} \psi_i(x_j)\,\psi_k(x_j) = 0 \text{ for every } i \neq k, \quad i,k = 0,1,...,n.$$

The orthogonal functions are *orthonormal* if, additionally,

$$\sum_{j=0}^{N} \psi_i(x_j)\,\psi_i(x_j) = 1, \quad i = 0,1,...n.$$

The Gram-Schmidt algorithm

Denote:
$\phi_0, ..., \phi_n$ – a non-orthogonal function basis,
$\psi_0, ..., \psi_n$ – an orthogonal function basis.

The (standard) Gram-Schmidt orthogonalization algorithm:

$$\begin{aligned} \psi_0 &= \phi_0, \\ \psi_1 &= \phi_1 - \frac{\langle \psi_0, \phi_1\rangle}{\langle \psi_0, \psi_0\rangle}\psi_0, \end{aligned}$$

$$\psi_i = \phi_i - \sum_{j=0}^{i-1} \frac{\langle \psi_j, \phi_i \rangle}{\langle \psi_j, \psi_j \rangle} \psi_j, \qquad i = 2, ..., n. \tag{4.16}$$

When the basis functions are orthogonal, then the Gram's matrix is a diagonal one, therefore the following very well conditioned set of normal equations occurs:

$$\begin{aligned} a_0 \langle \psi_0, \psi_0 \rangle &= \langle f, \psi_0 \rangle, \\ a_1 \langle \psi_1, \psi_1 \rangle &= \langle f, \psi_1 \rangle, \\ &\vdots \\ a_n \langle \psi_n, \psi_n \rangle &= \langle f, \psi_n \rangle. \end{aligned} \tag{4.17}$$

Example 4.3. We shall solve the problem formulated in the previous example, using an orthogonal function basis, obtained by the Gram-Schmidt orthogonalization of the initial natural basis of polynomials.

a) Applying the Gram-Schmidt algorithm to the function basis $\{1,\ x\}$, we get

$$\begin{aligned} \psi_0(x) &= 1, \\ \psi_1(x) &= x - \frac{\sum_{j=0}^5 x_j}{\sum_{j=0}^5 1} = x - \frac{1}{2}. \end{aligned}$$

The reader can easily check now, that the normal equations are:

$$\begin{aligned} a_0 \cdot 6 &= 3.2, \\ a_1 \cdot 0.7 &= -0.62, \end{aligned}$$

and we obtain the following approximating polynomial

$$w_1(x) = 0.533 - 0.886(x - 0.5) = 0.976 - 0.886x.$$

b) The basis $\{1,\ x,\ x^2\}$ is an augmentation of the basis from the point a), by the addition of the element x^2. Therefore, it is sufficient to add the third orthogonal function only, performing the next step of the Gram-Schmidt algorithm. This results in

$$\begin{aligned} \psi_2(x) &= x^2 - \frac{\sum_{j=0}^5 (x_j)^2}{\sum_{j=0}^5 1} 1 - \frac{\sum_{j=0}^5 (x_j)^2 (x_j - \frac{1}{2})}{\sum_{j=0}^5 (x_j - \frac{1}{2})^2} (x_j - \frac{1}{2}) \\ &= x^2 - x + \frac{2}{15}. \end{aligned}$$

The coefficients a_0 and a_1 remain unchanged, it remains to calculate a_2 from the equation:

$$a_2 \sum\nolimits_{j=0}^5 ({x_j}^2 - x_j + \frac{2}{15})^2 = \sum\nolimits_{j=0}^5 y_j ({x_j}^2 - x_j + \frac{2}{15}),$$

which results in $a_2 \approx 1$, and we get finally the approximating polynomial

$$\begin{aligned} w_2(x) &= 0.976 - 0.886x + (x^2 - x + 0.133) \\ &= 1.11 - 1.886x + x^2. \end{aligned}$$

□

When using he standard Gram-Schimdt algorithm, the accuracy of the orthogonalization may quickly deteriorate for subsequent orthogonal basis functions. Therefore, the *modified Gram-Schimdt algorithm* is recommended in practice, which is numerically superior (see algorithm (3.21) in Section 3.1). The process of orthogonalization of functions from a standard basis, performed by this algorithm, can be written as follows (using the "for" loop in the MATLAB notation):

```
for i=0:n
    $\psi_i$:=$\phi_i$;
    for j=i+1:n
        $\phi_j$:=$\phi_j$ - $\frac{\langle\psi_j,\phi_i\rangle}{\langle\psi_i,\psi_i\rangle}\psi_i$;          (4.18)
    end
end
```

An application of the standard Gram-Schimdt algorithm with a *re-orthogonalization* is an alternative, i.e., the orthogonalization is repeated, either after the whole algorithm is completed, or after evaluation of every basis function ψ_i, see [6].

4.2. Padé approximation

An approximating function is not always defined as the one minimizing a norm of the difference between the original and approximating functions. The Padé approximation, which is important for engineering applications, is an example of a quite different approach. The Padé approximating function is defined as a *rational function*:

$$R_{n,k}(x) = \frac{a_0 + a_1 x + \dots + a_n x^n}{1 + b_1 x + \dots + b_k x^k}, \tag{4.19}$$

where $b_0 = 1$ is assumed, to get a unique formulation.

Assume that the original function $f(x)$ has all derivatives (of any order) and let us represent this function by its Maclaurin series (i.e., the Taylor series at the point $x = 0$ – it will be the *point of approximation*):

$$f(x) = \sum_{i=0}^{\infty} c_i x^i, \tag{4.20}$$

$$c_i = \frac{1}{i!} f^{(i)}(0), \quad i = 0, 1, 2, \dots \tag{4.21}$$

The Padé approximation can be formulated as follows: *select parameters of the function $R_{n,k}$ in a way which results in first coefficients of the Maclaurin representation of $R_{n,k}$ equal to the corresponding coefficients c_i of the Maclaurin representation of the original function $f(x)$ (as many first coefficients as possible).*

There is $n+k+1$ degrees of freedom (the number of parameters of $R_{n,k}$). Therefore, the condition of the Padé approximation can be formulated in the form of the following equality:

$$\sum_{i=0}^{\infty} c_i x^i = \frac{a_0 + a_1 x + ... + a_n x^n}{1 + b_1 x + ... + b_k x^k} + O(x^{n+k+1}), \tag{4.22}$$

where $O(x^{n+k+1})$ can be treated as an algebraic polynomial with αx^{n+k+1} being its term with the lowest power of x (where α is a coefficient). Notice, that it follows directly from (4.22) that the first $n+k+1$ coefficients of the Maclaurin expansion of $R_{n,k}$ must be equal to the coefficients $c_i,\ i = 0, 1, ..., n+k$, thus the original requirement of the Padé approximation is satisfied.

Remark. It follows directly from the presented reasoning, that the condition of the Padé approximation can be also formulated as:

$$f(0) = R_{n,k}(0),$$

$$f^{(j)}(0) = R_{n,k}^{(j)}(0), \qquad j = 1, ..., n+k.$$

□

The equality (4.22) can be equivalently written in the following form:

$$(\sum_{i=0}^{\infty} c_i x^i)(1 + \sum_{i=1}^{k} b_i x^i) = \sum_{i=0}^{n} a_i x^i + O(x^{n+k+1}), \tag{4.23}$$

which can be written more precisely, eliminating those elements from the first sum at the left-hand side which lead (at this side) to the terms x^i with $i \geq n+k+1$:

$$(\sum_{i=0}^{n+k} c_i x^i)(1 + \sum_{i=1}^{k} b_i x^i) = \sum_{i=0}^{n} a_i x^i + O(x^{n+k+1}). \tag{4.24}$$

Define a polynomial with coefficients d_i, as follows:

$$\sum_{i=0}^{n+2k} d_i x^i = (\sum_{i=0}^{n+k} c_i x^i)(1 + \sum_{i=1}^{k} b_i x^i) - \sum_{i=0}^{n} a_i x^i. \tag{4.25}$$

Now the equality (4.24), which is the condition for evaluation of the unknown coefficients a_i, b_i, can be written in the form

$$d_i(a_0, ..., a_n, b_1, ... b_k, c_0, ..., c_{n+k}) = 0, \quad i = 0, 1, 2, ..., n+k, \tag{4.26}$$

where the coefficients c_i are known.

Example 4.4. We shall evaluate the Padé approximation $R_{2,2}$,

$$R_{2,2}(x) = \frac{a_0 + a_1 x + a_2 x^2}{1 + b_1 x + b_2 x^2},$$

of the function $f(x) = e^x$, at $x = 0$.
We have now $n = 2,\ k = 2,\ n + k = 4$. Expanding e^x into the Maclaurin series for $i = 0, ..., n + k$ (i.e., with first 5 terms) gives the first 5 expansion coefficients:

$$c_0 = 1,\ \ c_1 = 1,\ \ c_2 = \frac{1}{2},\ \ c_3 = \frac{1}{6},\ \ c_4 = \frac{1}{24}.$$

The equation (4.25) takes now the following form:

$$\sum_{i=5}^{5} d_i x^i = (1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4)(1 + b_1 x + b_2 x^2) - (a_0 + a_1 x + a_2 x^2)$$

and, after multiplication of the terms in the round brackets at the right-hand side,

$$\sum_{i=5}^{5} d_i x^i = 1 + (b_1 + 1)x + (b_2 + b_1 + \frac{1}{2})x^2 + (b_2 + \frac{1}{2}b_1 + \frac{1}{6})x^3 + \\ + (\frac{1}{2}b_2 + \frac{1}{6}b_1 + \frac{1}{24})x^4 + \cdots - (a_0 + a_1 x + a_2 x^2).$$

This leads, together with (4.26), to the following system of linear equations:

$$\begin{aligned} 1 &= a_0, \\ b_1 + 1 &= a_1, \\ b_2 + b_1 + \frac{1}{2} &= a_2, \\ b_2 + \frac{1}{2}b_1 + \frac{1}{6} &= 0, \\ \frac{1}{2}b_2 + \frac{1}{6}b_1 + \frac{1}{24} &= 0. \end{aligned}$$

The coefficients $b_1 = -\frac{1}{2}$ and $b_2 = \frac{1}{12}$ can first be calculated from two last equalities, next the coefficients a_i of the numerator can be evaluated from the first three equations, by simple substitutions. Thus, the Padé approximation function $R_{2,2}(x)$ is

$$R_{2,2}(x) = \frac{1 + \frac{1}{2}x + \frac{1}{12}x^2}{1 - \frac{1}{2}x + \frac{1}{12}x^2} = \frac{12 + 6x + 2x^2}{12 - 6x + x^2}.$$

Figure 4.4 presents: the original function e^x (solid line), the approximating function $R_{2,2}(x)$ (dashed line) and the function $F_4(x) = \sum_{i=0}^{4} c_i x^i$ (dotted line) consisting of the first 5 terms of the Maclaurin series (as the Padé approximation uses

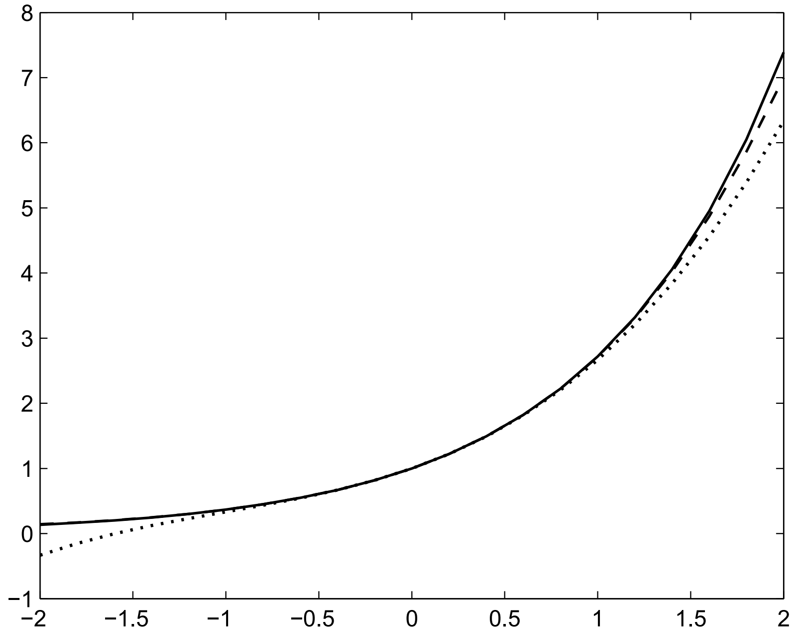


Figure 4.4. The original and the approximating functions in Example 4.4

precisely these terms). It can be easily seen that the Padé approximation, using the same information about the original function $f(x)$ as the truncated Maclaurin function $F_4(x)$ does, is more accurate, in a relatively wide neighbourhood of the approximation point. $\square$

A simple, well organized scheme of calculations leading to the coefficients of the Padé approximating function, applied in the example above, is general. It will be presented below. To this end, write equation (4.25) in the expanded form

$$\sum_{i=0}^{n+2k} d_i x^i = (1+b_1x+\cdots b_kx^k)(c_0+c_1x+\cdots+c_{n+k}x^{n+k})-(a_0+a_1x+\cdots+a_nx^n). \tag{4.27}$$

The formulae for the coefficients $a_i,\ b_i$ can be now easily formulated, according to (4.26):

1. First, the equations (4.26) equivalent to the requirement that the coefficients d_i are equal to zero, for $i = n+1, ..., n+k$, are considered:

$$\begin{gathered} b_kc_{n-k+1} + b_{k-1}c_{n-k+2} + \cdots + b_1c_n + c_{n+1} = 0, \\ b_kc_{n-k+2} + b_{k-1}c_{n-k+3} + \cdots + b_1c_{n+1} + c_{n+2} = 0, \\ \vdots \end{gathered}$$

$$\vdots$$

$$b_k c_n + b_{k-1} c_{n+1} + \cdots + b_1 c_{n+k-1} + c_{n+k} = 0,$$

assuming $c_j = 0$ for $j < 0$, if needed. Thus, a set of k linear equations has been obtained, which can be written in the form:

$$\begin{bmatrix} c_{n-k+1} & c_{n-k+2} & \cdots & c_n \\ c_{n-k+2} & c_{n-k+3} & \cdots & c_{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ c_n & c_{n+1} & \cdots & c_{n+k-1} \end{bmatrix} \begin{bmatrix} b_k \\ b_{k-1} \\ \vdots \\ b_1 \end{bmatrix} = \begin{bmatrix} -c_{n+1} \\ -c_{n+2} \\ \vdots \\ -c_{n+k} \end{bmatrix},$$

with the coefficients b_i, $i = 1, ..., k$, as the solution.

2. Next, the values of a_i are calculated from (4.27), zeroing the coefficients d_i for $i = 0, ..., n$:

$$\begin{aligned} a_0 &= c_0, \\ a_1 &= c_1 + b_1 c_0, \\ a_2 &= c_2 + b_1 c_1 + b_2 c_0, \\ &\vdots \\ a_n &= c_n + \sum_{i=1}^{min\{n,k\}} b_i c_{n-i}. \end{aligned}$$

Therefore, a simple set of k linear equations must be solved, dependent of only k denominator coefficients of the approximating function. Then, the coefficients of the numerator are obtained by simple substitutions.

The orders n and k of the numerator and denominator polynomials of the Padeé approximating function are chosen arbitrarily, according to a required accuracy of the approximation, best results are usually obtained assuming $n = k$.

Problems

1. Given the following collection of function values y_j at points x_j:

x_j	0	0.2	0.4	0.6	0.8	1.0
y_j	0.5	0.5	0.7	1.5	2.2	2.3

find approximating polynomials of the first and second order, using the least-squares method:

a) with the natural polynomial function basis,

b) with the orthogonal basis obtained by applying the Gram-Schmidt algorithm to the basis from a).

Compare the results, evaluate a maximal approximation error.

2. Using a basis consisting of the following two functions

$$\varphi_0(x) = 1, \;\; \varphi_1(x) = x^2,$$

   a) evaluate a least-squares approximation by the function $f(x) = a_0\varphi_0(x) + a_1\varphi_1(x)$, for the data

x_j	0	0.5	1	1.5	2
y_j	0.5	0.8	3	5	8.2

,

   b) orthogonalize the given basis and calculate the least-squares approximation using the orthogonal basis.

3. Evaluate the Padé approximation $R_{1,2}\,(x)$ of the function $f\,(x) = e^x$, at $x = 0$.

4. Evaluate the Padé approximation $R_{2,2}\,(x)$ of the function $f\,(x) = e^{-2x}$, at $x = 0$.

5. Evaluate the Padé approximation $R_{1,2}\,(x)$ of the function $f\,(x) = e^{-x}$, at $x = 2$ (hint: transform the problem to the problem of approximation at $x = 0$).

Chapter 5

Interpolation

The *problem of interpolation* can be formulated in the following way:
Given $n+1$ points $x_0, ..., x_n$ (interpolation nodes, grid points) and corresponding values $y_j = f(x_j)$ of a function $f(x)$, construct an interpolating function $F_n(x)$ (from a given class of functions) satisfying the conditions:

$$f(x_j) = F_n(x_j),\ j = 0, ..., n. \tag{5.1}$$

The interpolation is used in two cases: first, when an analytic form of the function $f(x)$ is not known, but values of this function are given at a finite number of points, and second, when an analytic form of the function $f(x)$ is known, but evaluation of this function is too complex for a class of applications. When the interpolating function $F(x)$ is evaluated, it can be used to calculate (approximately) values of the original function $f(x)$ between the interpolation nodes.

The problem of interpolation is graphically illustrated in Fig. 5.1, for an interpolation of a continuous function $f(x)$ at three interpolation nodes $x_0,\ x_1$ and x_2, using a polynomial of second order (a parabola) as the interpolating function $F(x)$.

An important *difference between interpolation and approximation tasks* should be pointed out. The interpolation results in a function from a chosen class (e.g., algebraic polynomials), matching exactly the values of the original function at all

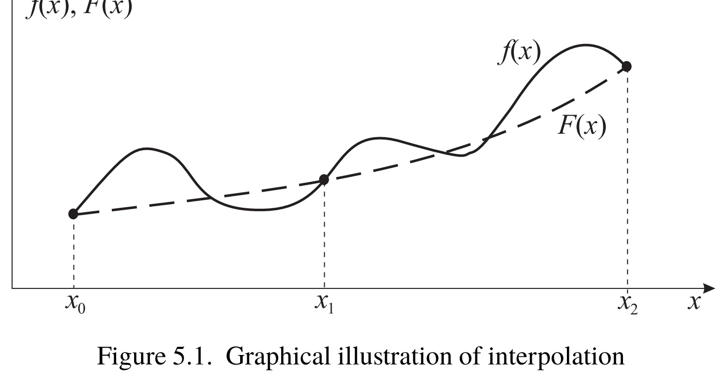


Figure 5.1. Graphical illustration of interpolation

interpolation nodes. Both parameters and coefficients of the interpolating function depend on the number and location of the interpolation nodes (e.g., in the case of polynomials, both the order of a polynomial and values of its coefficients). On the other hand, in the case of the approximation, only coefficients of a more precisely defined approximating function are evaluated (e.g., in the case of polynomials, only coefficients of a polynomial of an assumed order), in a way leading to minimization of a distance between both functions – the approximating function usually does not match the values of the original function at the approximation nodes.

Important classes of interpolating functions include polynomials (in particular algebraic) and spline functions, the considerations will be limited to these cases.

5.1. Algebraic polynomial interpolation

Theorem 5.1. *There is only one interpolating algebraic polynomial of order at most n with the values $y_0 = f(x_0), \ldots, y_n = f(x_n)$ at the points $x_0, ..., x_n$.*
Proof: The required polynomial is of the form

$$W_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0. \tag{5.2}$$

The interpolation conditions (5.1) define a set of $n+1$ linear equations with $n+1$ unknown coefficients $a_0, \ldots, a_n$:

$$a_0 + a_1 x_j + a_2 x_j^2 + \cdots + a_n x_j^n = f(x_j), \quad j = 0, ..., n. \tag{5.3}$$

The matrix $\mathbf{X}$ of this system of linear equations is of the form:

$$\mathbf{X} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}. \tag{5.4}$$

This is a Vandermonde's matrix, with the determinant:

$$\det \mathbf{X} = \prod_{1 \le i < j \le n} (x_j - x_i). \tag{5.5}$$

This determinant is non-zero, if all the points are distinct, i.e., $x_i \neq x_j$ for $i \neq j$, $i, j = 0, ..., n$, which occurs in the considered case of the interpolation. Therefore, the system of equations (5.3) has a unique solution – there is a unique interpolating polynomial of an order not larger than n, which concludes the proof. □

Corollary. In the case of an algebraic polynomial interpolation, all interpolation formulae are equivalent (i.e., for the same data the same polynomial is always generated, only written in a different form).

Theoretically, coefficients of the interpolating algebraic polynomial can be evaluated solving the system of linear equations (5.3). However, the number of computations is then of order n^3, not counting computations needed to calculate elements of the matrix (5.4). The polynomial interpolation formulae presented further in this chapter require much less computations, of order n^2. There are two practically important formulations, developed by Langrange and by Newton.

5.1.1. Lagrange interpolating polynomial

Denote:

$$\begin{aligned} L_j(x) &\overset{\text{df}}{=} \prod_{i=0, i\neq j}^{n} \frac{x-x_i}{x_j-x_i} \\ &= \frac{(x-x_0)(x-x_1)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_n)}{(x_j-x_0)(x_j-x_1)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_n)} \end{aligned} \tag{5.6}$$

and notice that

$$L_j(x_k) = \delta_{jk} = \begin{cases} 1 & \text{for } k=j, \\ 0 & \text{for } k\neq j, \quad j,k=0,1,\ldots,n. \end{cases}$$

Therefore, the polynomial

$$W_n(x) = \sum_{j=0}^{n} y_j L_j(x) \tag{5.7}$$

is the required interpolating polynomial, called the *Lagrange interpolating polynomial.*

It can be proved, provided f is $n+1$ times continuously differentiable on the interval $[a,b]$ containing the interpolation nodes, that the error of interpolation can be expressed in the form

$$r(x) = f(x) - W_n(x) = \frac{f^{(n+1)}(\alpha(x))}{(n+1)!}\omega_n(x), \tag{5.8}$$

where $\alpha(x) \in (a,\ b)$ and

$$\omega_n(x) \overset{\text{df}}{=} (x-x_0)(x-x_1)\ldots(x-x_n). \tag{5.9}$$

Therefore, an upper bound of the interpolation error can be expressed as

$$|f(x) - W_n(x)| \leq \frac{M_{n+1}}{(n+1)!}|\omega_n(x)|, \tag{5.10}$$

where

$$M_{n+1} = \sup_{x\in[a,b]} \left| f^{(n+1)}(x) \right|. \tag{5.11}$$

In many applications the Newton's interpolating polynomial is more convenient.

5.1.2. Newton's interpolating polynomial

Divided differences

The divided differences are defined in the following way:

– of first order:

$$f[x_i, x_{i+1}] \overset{\text{df}}{=} \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i},$$

– of second order:

$$f[x_i, x_{i+1}, x_{i+2}] \overset{\text{df}}{=} \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i},$$

– of n-th order:

$$f[x_i, ..., x_{i+n}] \overset{\text{df}}{=} \frac{f[x_{i+1}, x_{i+2}, ..., x_{i+n}] - f[x_i, x_{i+1}, ..., x_{i+n-1}]}{x_{i+n} - x_i}.$$

It is efficient to calculate the divided differences using a *triangle table scheme:*

$x_i, f(x_i)$	divided difference		
	of order 1	of order 2	of order 3
$x_0, f(x_0)$			
	$f[x_0, x_1] = \frac{f(x_1)-f(x_0)}{x_1-x_0}$		
$x_1, f(x_1)$		$f[x_0, x_1, x_2] = \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$	
	$f[x_1, x_2] = \frac{f(x_2)-f(x_1)}{x_2-x_1}$		$f[x_0, x_1, x_2, x_3]$
$x_2, f(x_2)$		$f[x_1, x_2, x_3] = \frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$	
	$f[x_2, x_3] = \frac{f(x_3)-f(x_2)}{x_3-x_2}$		$\{f[x_1, x_2, x_3, x_4]\}$
$x_3, f(x_3)$		$\left\{f[x_2, x_3, x_4] = \frac{f[x_3,x_4]-f[x_2,x_3]}{x_4-x_2}\right\}$	
	$\left\{f[x_3, x_4] = \frac{f(x_4)-f(x_3)}{x_4-x_3}\right\}$		
$\{x_4, f(x_4)\}$			

Note that the numerator of every next divided difference is a difference of two neighboring, preceding (from the previous column) divided differences of lower order, and its denominator is the difference of two extreme points from denominators of these differences. Note also that after adding a next point, it is enough to calculate divided differences located only along the bottom edge of the "triangle" – the elements calculated after addition of x_4 are shown in curly brackets in the table.

It can be easily shown by induction that:

$$\begin{aligned} f\left[x_i, x_{i+1}, ..., x_{i+k}\right] \\ &= \sum_{j=i}^{j=i+k} \frac{f\left(x_j\right)}{\left(x_j - x_i\right)\left(x_j - x_{i+1}\right)\cdots\left(x_j - x_{j-1}\right)\left(x_j - x_{j+1}\right)\cdots\left(x_j - x_{i+k}\right)} \\ &= \sum_{j=i}^{j=i+k} \frac{f\left(x_j\right)}{\prod\limits_{p=i,p\neq j}^{p=i+k}\left(x_j - x_p\right)}. \end{aligned} \tag{5.12}$$

For example:

$$f\left[x_0, x_1\right] = \frac{f\left(x_1\right) - f\left(x_0\right)}{x_1 - x_0} = \frac{f\left(x_0\right)}{x_0 - x_1} + \frac{f\left(x_1\right)}{x_1 - x_0},$$

$$\begin{aligned} f\left[x_0, x_1, x_2\right] &= \frac{f\left[x_1, x_2\right] - f\left[x_0, x_1\right]}{x_2 - x_0} \\ &= \left(\frac{f\left(x_2\right)}{x_2 - x_1} + \frac{f\left(x_1\right)}{x_1 - x_2} - \frac{f\left(x_1\right)}{x_1 - x_0} - \frac{f\left(x_0\right)}{x_0 - x_1}\right)\frac{1}{x_2 - x_0} \\ &= \frac{f\left(x_0\right)}{(x_0 - x_1)(x_0 - x_2)} + \frac{f\left(x_1\right)}{(x_1 - x_0)(x_1 - x_2)} + \frac{f\left(x_2\right)}{(x_2 - x_0)(x_2 - x_1)}. \end{aligned}$$

Newton's interpolating formula

Consider the interpolating polynomial expressed in the form:

$$\begin{aligned} W_n\left(x\right) = W_0\left(x\right) + \\ + \left[W_1\left(x\right) - W_0\left(x\right)\right] + \left[W_2\left(x\right) - W_1\left(x\right)\right] + \cdots + \left[W_n\left(x\right) - W_{n-1}\left(x\right)\right], \end{aligned}$$

where $W_0 = f\left(x_0\right)$. Notice that

$$\begin{aligned} W_k\left(x\right) - W_{k-1}\left(x\right) &= A_k\left(x - x_0\right)\left(x - x_1\right)...\left(x - x_{k-1}\right) \\ &= A_k\omega_{k-1}\left(x\right), \quad k = 1, ..., n, \end{aligned} \tag{5.13}$$

because $W_k(x)$ and $W_{k-1}(x)$ have the same values at the points $x_{0,}$ $x_{1,}$ ... , x_{k-1}. Therefore

$$W_n\left(x\right) = f\left(x_0\right) + A_1\omega_0\left(x\right) + A_2\omega_1\left(x\right) + ... + A_n\omega_{n-1}\left(x\right).$$

The coefficient by the highest power of x in $\omega_{k-1}\left(x\right)$ is equal to 1, because

$$\omega_{k-1}\left(x\right) \overset{\text{df}}{=} \left(x - x_0\right)\left(x - x_1\right)\cdots\left(x - x_{k-1}\right) = x^k + \cdots$$

and the polynomial $W_{k-1}(x)$ is of order $k-1$. Therefore, (5.13) implies that the coefficient by the highest power (k-th) in the polynomial $W_k(x)$ is A_k, i.e.,

$$W_k\left(x\right)=A_k x^k+\ldots$$

On the other hand, using the Lagrange interpolating polynomial we have

$$\begin{aligned} W_k(x) &= \sum_{j=0}^{k} f(x_j) \prod_{i=0,i\neq j}^{k} \frac{x-x_i}{x_j-x_i} = \sum_{j=0}^{k} f\left(x_j\right) \frac{\prod\limits_{i=0,i\neq j}^{k}\left(x-x_i\right)}{\prod\limits_{i=0,i\neq j}^{k}\left(x_j-x_i\right)} \\ &= \sum_{j=0}^{k} f\left(x_j\right) \frac{x^k+\cdots}{\prod\limits_{i=0,i\neq j}^{k}\left(x_j-x_i\right)} = \sum_{j=0}^{k} \frac{f\left(x_j\right)}{\prod\limits_{i=0,i\neq j}^{k}\left(x_j-x_i\right)}\left(x^k+\cdots\right). \end{aligned}$$

Comparing the last two equations and using (5.12), we obtain

$$A_k=\sum_{j=0}^{k} \frac{f\left(x_j\right)}{\prod\limits_{i=0,i\neq j}^{k}\left(x_j-x_i\right)} = f\left[x_0,x_1,...,x_k\right], \qquad k=1,...,n. \tag{5.14}$$

Thus, we get finally *the Newton's interpolating polynomial*:

$$\begin{aligned} W_n\left(x\right) &= f\left(x_0\right)+f[x_0,x_1]\,\omega_0(x)+ \\ &\quad + f[x_0,x_1,x_2]\,\omega_1\left(x\right)+\cdots+f[x_0,\ldots,x_n]\,\omega_{n-1}\left(x\right) \\ &= f\left(x_0\right)+\sum_{k=1}^{n} f\left[x_0,...,x_k\right]\,\omega_{k-1}(x)\,. \end{aligned} \tag{5.15}$$

In both cases of the interpolating polynomials, the Lagrange one and the Newton's one, no assumption was made concerning relations between values of the interpolation nodes – the nodes may be given in any order. Therefore, when using the Newton's formula, it is easy to add a new, additional node – and it can be located as the last one in the triangle table, regardless of its value.

Example 5.1. Given four points (the interpolation nodes) $a=x_0=0$, $x_1=1$, $x_2=2$, $x_3=b=3$, together with the function values at these points, $y_0=1$, $y_1=2$, $y_2=4$, $y_3=3$, respectively. We shall evaluate the algebraic interpolating polynomial, using both interpolation formulae.

Applying the Lagrange formula we have:

$$\begin{aligned}
W_3(x) &= 1 \cdot \frac{(x-1)(x-2)(x-3)}{-6} + 2 \cdot \frac{x(x-2)(x-3)}{2} + \\
&\quad + 4 \cdot \frac{x(x-1)(x-3)}{-2} + 3 \cdot \frac{x(x-1)(x-2)}{6} \\
&= -\frac{1}{6}(x^3 - 6x^2 + 11x - 6) + (x^3 - 5x^2 + 6x) + \\
&\quad - 2(x^3 - 4x^2 + 3x) + \frac{1}{2}(x^3 - 3x^2 + 2x) \\
&= 1 - \frac{5}{6}x + 2.5x^2 - \frac{2}{3}x^3.
\end{aligned}$$

Applying the Newton's formula we have:

$$\begin{aligned}
W_3(x) &= 1 + 1 \cdot (x-0) + \frac{1}{2}(x-0)(x-1) + \frac{-2}{3}(x-0)(x-1)(x-2) \\
&= 1 + x + \frac{1}{2}(x^2 - x) - \frac{2}{3}(x^3 - 3x^2 + 2x) \\
&= 1 - \frac{5}{6}x + 2.5x^2 - \frac{2}{3}x^3. \qquad (5.16)
\end{aligned}$$

We have obtained, certainly, the same algebraic polynomial – only the algorithms leading to it are different. The polynomial (5.16) is shown in Fig. 5.2. □

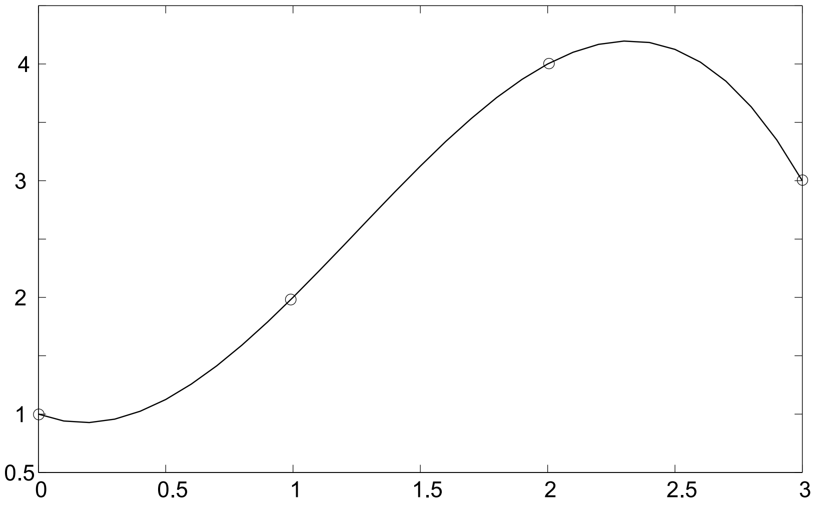


Figure 5.2. The interpolating polynomial (5.16)

The Newton's form of the interpolating polynomial can be used to calculate effectively values of the interpolating polynomial at points located between the interpolation nodes. This leads to the *Horner scheme* which, e.g., for $n = 4$, can be expressed in the form:

$$\begin{aligned} W_n(x) &= f(x_0) + \\ &\quad + A_1(x - x_0) + \\ &\quad + A_2(x - x_0)(x - x_1) + \\ &\quad + A_3(x - x_0)(x - x_1)(x - x_2) + \\ &\quad + A_4(x - x_0)(x - x_1)(x - x_2)(x - x_3) \\ &= f(x_0) + \\ &\quad + (x - x_0)[A_1 + (x - x_1)[A_2 + (x - x_2)[A_3 + (x - x_3)A_4]]]. \end{aligned}$$

The Newton's formula does not depend on the relative location of the interpolation nodes. If we take them in a reversed order, the *backward Newton's interpolating polynomial* is obtained:

$$\begin{aligned} W_n(x) = f(x_n) &+ f[x_n, x_{n-1}](x - x_n) + \\ &+ f[x_n, x_{n-1}, x_{n-2}](x - x_n)(x - x_{n-1}) + \\ &\cdots + f[x_n, x_{n-1}, \ldots, x_0](x - x_n)\cdots(x - x_1). \end{aligned} \tag{5.17}$$

The case of equally-spaced nodes[*]

Assume that the subsequent interpolation nodes are equally-spaced, i.e.,

$$x_i = x_0 + ih, \qquad i = 0, ..., n,$$

and denote

$$y_{i\pm k} = f(x_i \pm kh).$$

The definition of a *forward (progressive) difference*:

– of first order:

$$\Delta f(x_i) = f(x_i + h) - f(x_i), \quad \text{i.e.,}\ \Delta y_i = y_{i+1} - y_i,$$

– of second order:

$$\begin{aligned} \Delta^2 f(x_i) &= \Delta(\Delta f(x_i)) \\ &= \Delta[f(x_i + h) - f(x_i)] \\ &= \Delta f(x_i + h) - \Delta f(x_i) \\ &= f(x_i + 2h) - f(x_i + h) - (f(x_i + h) - f(x_i)) \\ &= f(x_i + 2h) - 2f(x_i + h) + f(x_i), \end{aligned}$$

[*]Optional.

i.e., $\Delta^2 y_i = \Delta(\Delta y_i) = y_{i+2} - 2y_{i+1} + y_i,$

– of order n:

$$\Delta^n y_i = \Delta\left(\Delta^{n-1} y_i\right) = \sum_{j=0}^{n} (-1)^j \tbinom{n}{j} y_{i+n-j}.$$

The difference operator Δ is linear, it has several properties equivalent to those of the differentiation operator $\frac{d}{dt}$, in particular

$$\Delta^m (\Delta^n f) = \Delta^{m+n} f.$$

The definition of a *backward difference*:
– of first order:

$$\nabla f(x_i) = f(x_i) - f(x_i - h), \quad \text{i.e.,} \quad \nabla\, y_i = y_i - y_{i-1},$$

– of second order:

$$\begin{aligned}\nabla^2 y_i &= \nabla(\nabla y_i)\\ &= \nabla(y_i - y_{i-1}) = \nabla y_i - \nabla y_{i-1}\\ &= (y_i - y_{i-1}) - (y_{i-1} - y_{i-2}) = y_i - 2y_{i-1} + y_{i-2},\end{aligned}$$

– of order n:

$$\nabla^n y_i = \nabla\left(\nabla^{n-1} y_i\right).$$

The following relation between the forward and backward differences can be shown to be true:

$$\nabla^k y_i = \Delta^k y_{i-k}, \qquad k = 0, 1, ..., n.$$

It is convenient to calculate the forward and backward differences in the structure of the triangle table, similarly as it was for the divided differences.

Because:

$$\begin{aligned} f(x_0) &= y_0,\\ f[x_0, x_1] &= \frac{y_1 - y_0}{h} = \frac{\Delta y_0}{h},\\ f[x_0, x_1, x_2] &= \frac{\frac{\Delta y_1}{h} - \frac{\Delta y_0}{h}}{2h} = \frac{\Delta^2 y_0}{2h^2},\\ &\vdots\\ f[x_0, x_1, ..., x_k] &= \frac{\Delta^k y_0}{k! h^k}, \quad k = 1, 2, ..., n,\end{aligned}$$

then the *Newton's interpolating polynomial for equally-spaced nodes* can be written in the form:

$$W_n(x) = y_0 + \frac{\Delta y_0}{h}(x - x_0) + \frac{\Delta^2 y_0}{2h^2}(x - x_0)(x - x_1) + \cdots$$
$$\cdots + \frac{\Delta^n y_0}{n!h^n}(x - x_0)\cdots(x - x_{n-1}). \quad (5.18)$$

Similarly, because

$$f[x_n, x_{n-1}, ..., x_{n-k}] = \frac{\nabla^k y_n}{k!h^k},$$

then the *backward Newton's interpolating polynomial for equally-spaced nodes* can be written in the form:

$$W_n(x) = y_n + \frac{\nabla y_n}{h}(x - x_n) + \frac{\nabla^2 y_n}{2h^2}(x - x_n)(x - x_{n-1}) + \cdots$$
$$\cdots + \frac{\nabla^n y_n}{n!h^n}(x - x_n)\cdots(x - x_1). \quad (5.19)$$

Convergence of the interpolation process

Generally, an interpolating polynomial may not converge uniformly to the original function, when the number of the interpolation nodes uniformly increases

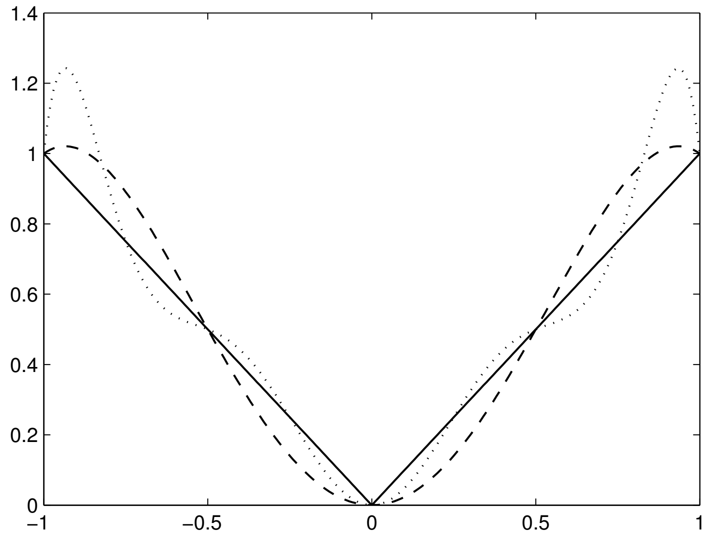


Figure 5.3. Plots of the function $f(x) = |x|$ and its interpolating polynomials of 4-th (dashed line) and 8-th (dotted line) order, interpolation nodes equally spaced over $[-1, 1]$

within the given interval (thus, the order of the polynomial increases). In particular, for a non-smooth function the convergence may fail. Therefore, for such functions the application of the polynomial interpolation is not recommended, especially when using polynomials of higher orders. For such functions, an interpolation with spline functions leads to better results.

Example 5.2. Fig. 5.3 presents plots of algebraic polynomials interpolating the function $f(x) = |x|$ over the interval $[-1, 1]$ at 5 and 9 equally spaced points, respectively (polynomials of 4-th and 8-th order). □

5.2. Spline function interpolation

Consider an interval $[a, b]$ and the interpolation nodes x_i, $i = 0, ..., n$, such that

$$a = x_0 < x_1 < x_2 < ... < x_n = b.$$

Denote this partition of the interval $[a, b]$ by $\triangle_n$.

Definition. The function $s\,(x) = s\,(x, \triangle_n)$ defined on the interval $[a, b]$ is called a *spline function* of order $m \geq 1$, if it satisfies the following conditions:

1. the function $s\,(x)$ is a polynomial of order not higher than m on every subinterval $[x_i, x_{i+1}]$, $i = 0, 1, ...n - 1$,
2. the function $s\,(x)$ is of class $C_{[a,b]}^{(m-1)}$ (has continuous derivative of order $m-1$ on $[a, b]$).

The interpolating spline functions will be considered only, i.e., satisfying

$$s\,(x_i) = y_i, \qquad i = 0, ..., n.$$

The *uniqueness* of the spline function definition:
The first condition means that for every $x \in [x_i, x_{i+1}]$

$$s\,(x) = a_{im}x^m + a_{i(m-1)}x^{m-1} + ... + a_{i1}x + a_{i0}.$$

Thus, for every i there are m+1 unknown coefficients a_{im}, that is together there are $n\,(m + 1)$ *coefficients to be evaluated.* On the other hand, there are the *following conditions (constraints):*

- $s\,(x_i) = y_i$, $i = 0, ..., n$, which results in $n + 1$ conditions,
- continuity of the derivatives of order $0, 1, ..., m - 1$ at the points $x_1, ..., x_{n-1}$, which results in $m\,(n - 1) = mn - m$ conditions.

Therefore, there are together $(m + 1)\,n - m + 1$ conditions, which results in the following number of *degrees of freedom:*

$$n(m + 1) - [(m + 1)\,n - m + 1] = m - 1.$$

There is $m-1$ degrees of freedom, i.e., for linear functions there is no freedom, for the spline functions of second order there is one degree of freedom, etc. Therefore, if a spline function of order $m>1$ is to be calculated uniquely, additional conditions have to be prescribed. Because all function values are uniquely determined, additional conditions must concern the derivatives.

To define uniquely a spline function of order 2 one additional condition has to be assumed, e.g., on boundary values of the derivative:

$$\frac{ds}{dx}\left(b^-\right) = \beta_1, \tag{5.20a}$$

$$\frac{ds}{dx}\left(a^+\right) = \alpha_1, \tag{5.20b}$$

$$\frac{ds}{dx}\left(a^+\right) = \frac{ds}{dx}\left(b^-\right). \tag{5.20c}$$

To define uniquely a spline function of order 3 two additional conditions on the derivatives have to be assumed, e.g., on boundary values:

$$\frac{ds}{dx}\left(a^+\right)=\alpha_1, \quad \frac{ds}{dx}\left(b^-\right)=\beta_1, \tag{5.21a}$$

$$\frac{d^2s}{dx^2}\left(a^+\right)=\alpha_2, \quad \frac{d^2s}{dx^2}\left(b^-\right)=\beta_2, \tag{5.21b}$$

$$\frac{ds}{dx}\left(a^+\right)=\frac{ds}{dx}\left(b^-\right), \quad \frac{d^2s}{dx^2}\left(a^+\right)=\frac{d^2s}{dx^2}\left(b^-\right). \tag{5.21c}$$

Additional conditions on the values of the derivatives can also be defined at intermediate points (*glue points, knobs*). For instance, the default conditions assumed in the ”`spline`” function (which calculates cubic splines) in MATLAB environment are the requirements of continuity of the 3-rd order derivative of the spline function at the points x_1 (first knob) and x_{n-1} (last knob):

$$\frac{d^3s}{dx^3}(x_1^+)=\frac{d^3s}{dx^3}(x_1^-), \quad \frac{d^3s}{dx^3}(x_{n-1}^+)=\frac{d^3s}{dx^3}(x_{n-1}^-). \tag{5.21d}$$

The points x_1 and x_{n-1} are not, therefore, typical intermediate (knob) points, hence the name for these conditions: ”*not-a-knob conditions*” – the spline function has at x_1 and x_{n-1} all (and continuous) derivatives, because the third order derivative of a third order polynomial is a constant function (the higher derivatives are zero).

The *not-a-knob* conditions are equivalent to the requirement to take equal coefficients by the polynomial third order terms, applied to polynomials over the first two subintervals (the knob point x_1) and over the last two subintervals (the knob point x_{n-1}).

Example 5.3. Given four points $a = x_0, x_1, x_2, x_3 = b$, together with the function values at this points y_0, y_1, y_2, y_3, respectively. Define a system of equations uniquely describing a spline function of second order $(m = 2)$, under the condition

$$s'\left(a^+\right) = s_0. \tag{5.22}$$

Over every subinterval, the function $s(x)$ can be written in the form:

$$s(x) = s_i(x) = y_i + b_i(x - x_i) + c_i(x - x_i)^2, \quad x\epsilon[x_i, x_{i+1}], \quad i = 0, 1, 2. \tag{5.23}$$

This reduces the number of unknown coefficients b_i and c_i of the spline function from 9 to 6 (first $n = 3$ conditions satisfied). The remaining conditions are:
– the requirement of function value at the last point:

$$s_2(x_3) = y_2 + b_2(x_3 - x_2) + c_2(x_3 - x_2)^2 = y_3, \tag{5.24}$$

– the requirements of function continuity at two intermediate points:

$$y_0 + b_0(x_1 - x_0) + c_0(x_1 - x_0)^2 = y_1, \tag{5.25}$$

$$y_1 + b_1(x_2 - x_1) + c_1(x_2 - x_1)^2 = y_2, \tag{5.26}$$

– the continuity of the first derivatives at these points:

$$b_0 + 2c_0(x_1 - x_0) = b_1, \tag{5.27}$$

$$b_1 + 2c_1(x_2 - x_1) = b_2, \tag{5.28}$$

– and the boundary condition (5.22):

$$b_0 = s_0. \tag{5.29}$$

The system of equations (5.24) - (5.29) formulates the required set of 6 equations with six coefficients b_i and c_i as the unknowns. Note that it is a set of *linear* algebraic equations.

Assuming the numerical values:

x_i	0	1	2	3
y_i	1	2	4	3

, $s_0 = 0$,

the following system of equations is obtained:

$$\begin{aligned} b_2 + c_2 &= -1, \\ b_0 + c_0 &= 1, \\ b_1 + c_1 &= 2, \\ b_0 + 2c_0 &= b_1, \\ b_1 + 2c_1 &= b_2, \\ b_0 &= 0. \end{aligned}$$

The solution of these equations results in the following spline function:

$$s(x) = 1 + x^2, \quad x \in [0,1],$$
$$s(x) = 2 + 2(x-1), \quad x \in [1,2],$$
$$s(x) = 4 + 2(x-2) - 3(x-2)^2, \quad x \in [2,3].$$

The plot of this function is given in Fig. 5.4.

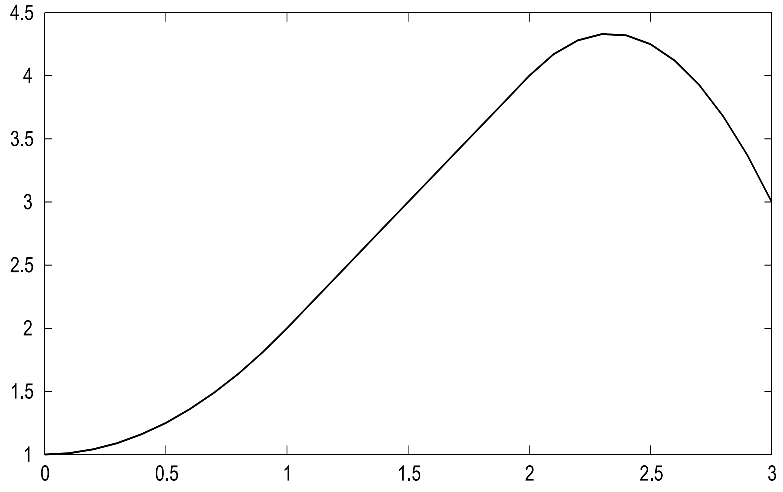


Figure 5.4. Plot of the spline function, Example 5.3

We leave to the reader the problem to derive the spline function for a different value of s_0, to plot the obtained spline function and to compare it with that obtained above. □

Example 5.4. Given three points (interpolation nodes) $a = x_0 = -1$, $x_1 = 0$, $x_2 = 2$, together with the corresponding function values, $y_0 = 0$, $y_1 = 4$, $y_2 = 2$, i.e., we have the data:

x_i	-1	0	2
y_i	0	4	2

For the given data a cubic spline function $s(x)$ (of third order, $(m = 3)$) should be evaluated, under additional (boundary) conditions:

$$\frac{ds}{dx}(-1^+) = 2, \tag{5.30a}$$
$$\frac{ds}{dx}(2^-) = -25. \tag{5.30b}$$

There are two subintervals, $[-1,0]$ and $[0,2]$, therefore the spline function can be written in the form:

$$\begin{aligned} s(x)=s_0(x) &= 0+b_0(x+1)+c_0(x+1)^2+d_0(x+1)^3,\ \ x\in[-1,0],\\ s(x)=s_1(x) &= 4+b_1(x-0)+c_1(x-0)^2+d_1(x-0)^3,\ \ x\in[0,2], \end{aligned}$$

which reduces the number of unknown coefficients b_i, c_i and d_i of the spline function from 8 to 6 (first $n=2$ conditions satisfied). The remaining conditions are on:

– the function value at the end point:

$$s_1(x_2)=y_1+b_1(x_2-x_1)+c_1(x_2-x_1)^2+d_1(x_2-x)^3=y_2,$$

$$\text{i.e.,}\qquad 4+2b_1+4c_1+8d_1=2, \tag{5.31}$$

– the function continuity at the intermediate point:

$$y_0+b_0(x_1-x_0)+c_0(x_1-x_0)^2+d_0(x_1-x_0)^2=y_1,$$

$$\text{i.e.,}\qquad b_0+c_0+d_0=4, \tag{5.32}$$

– the continuity of the first derivatives at this point:

$$b_0+2c_0(x_1-x_0)+3d_0(x_1-x_0)^2=b_1,$$

$$\text{i.e.,}\qquad b_0+2c_0+3d_0=b_1, \tag{5.33}$$

– the continuity of the second derivatives at this point:

$$2c_0+6d_0(x_1-x_0)=2c_1,$$

$$\text{i.e.,}\qquad 2c_0+6d_0=2c_1, \tag{5.34}$$

– and the boundary conditions (5.30a) and (5.30b), respectively:

$$b_0=2, \tag{5.35}$$

$$b_1+2c_1(x_2-x_1)+3d_1(x_2-x_1)^2=-25,$$

$$\text{i.e.,}\qquad b_1+4c_1+12d_1=-25. \tag{5.36}$$

The formulae (5.31) - (5.36) constitute the set of six linear equations defining the six unknown coefficients $b_i, c_i, d_i,\ i=0,1$. From the solution of this set (which is left to the reader) the following spline function is obtained:

$$\begin{aligned} s(x)=s_0(x) &= 2(x+1)+(x+1)^2+(x+1)^3,\ \ x\in[-1,0],\\ s(x)=s_1(x) &= 4+7x+4x^2-4x^3,\ \ x\in[0,2]. \end{aligned}$$

The function is plotted in Fig. 5.5. □

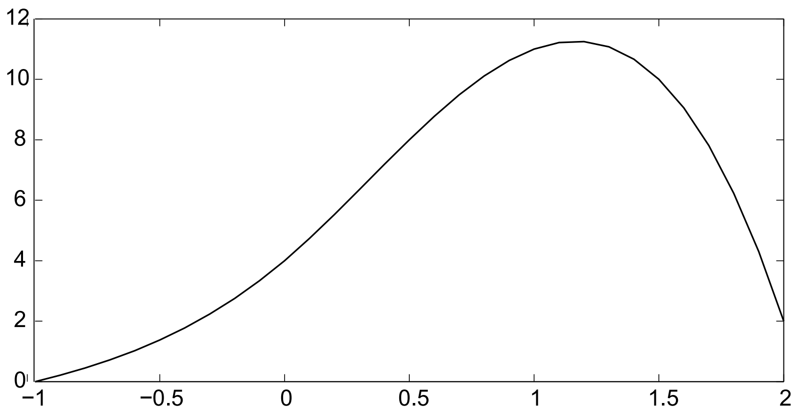


Figure 5.5. The plot of the spline function from Example 5.4

Example 5.5. Let us continue the comparison performed earlier in Example 5.2. Figure 5.6 shows both the interpolating polynomial of order 8 and the cubic spline function with additional conditions of type (5.21a), with first order derivatives at both boundary points equal to zero – both functions are calculated for 9 equally spaced interpolation nodes over the interval $[-1, 1]$, with the values for the function $f(x) = |x|$.

Further, Fig. 5.7 presents a comparison of the plots of the same functions, but with the cubic spline function defined assuming different additional conditions on derivatives, this time of type (5.21d) – *not-a-knob conditions*.

The results presented in both figures clearly indicate much better quality of interpolation by the spline function, as compared to the interpolation by a single polynomial of higher order. □

Example 5.6.[*] We shall derive a system of linear equations defining coefficients of a spline function of third order ($m = 3$, a cubic spline function), for any assumed number n of interpolation nodes $x_0, ..., x_n$ and any given values of the original function at these points, for the case of the boundary conditions (5.21b) – in a form convenient for computer implementations.

For every subinterval $x \in [x_i, x_{i+1}]$, $i = 0, ..., n-1$, the following spline function is defined:

$$s\,(x) = s_i(x) = y_i + b_i\,(x - x_i) + c_i\,(x - x_i)^2 + d_i\,(x - x_i)^3\,.$$

Denote h_i the step-length of the i-th subinterval $h_i \overset{\text{df}}{=} x_{i+1} - x_i$. It is now easier to formulate the conditions on the coefficients of the spline function:

[*]Optional.

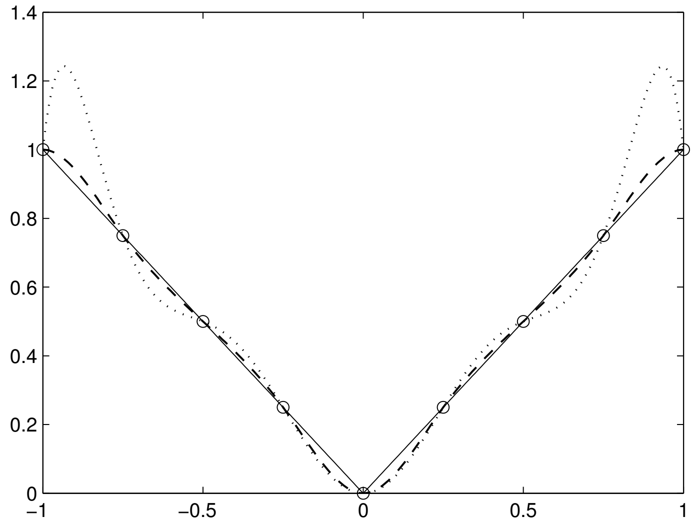


Figure 5.6. A comparison of the interpolating polynomial of 8 order (dotted line) and the cubic spline function (with derivatives at boundary points equal to zero, dashed line), for the function $f(x) = |x|$, using 9 interpolation nodes

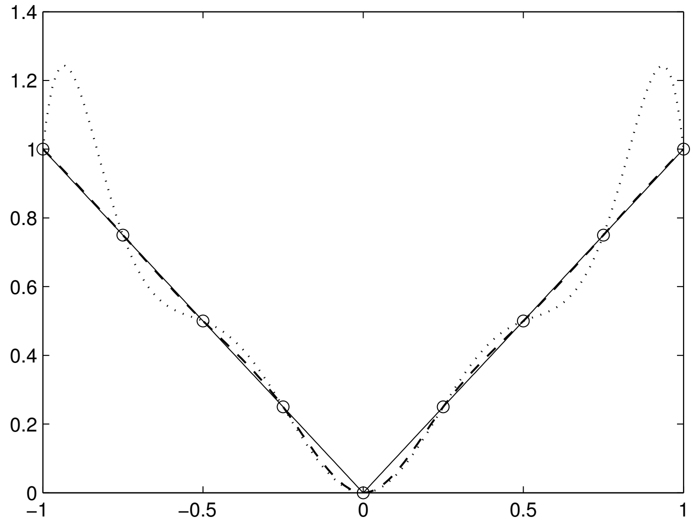


Figure 5.7. A comparison of the interpolating polynomial of 8 order (dotted line) and the cubic spline function (with *not-a-knob* additional conditions, dashed line), for the function $f(x) = |x|$, using 9 interpolation nodes

– From the continuity of $s\,(x)$ at the intermediate nodes and the final value condition $y_n = s_{n-1}(x_n)$, we have:

$$y_{i+1} = y_i + b_i h_i + c_i h_i^2 + d_i h_i^3, \qquad i = 0, ..., n-1. \tag{5.37}$$

– From the continuity of $\frac{d^2 s}{dx^2}\,(x)$ at the intermediate nodes:

$$2c_{i+1} = 2c_i + 6d_i h_i, \quad i = 0, ..., n-1, \tag{5.38}$$

where $2c_n$ is defined as given by the second boundary condition of type b), i.e.:

$$2c_n \overset{\text{df}}{=} \frac{d^2 s}{dx^2}\left(b^-\right) = \beta_2. \tag{5.39}$$

Therefore, we can rewrite (5.38) in then form

$$d_i = \frac{c_{i+1} - c_i}{3h_i}, \quad i = 0, ..., n-1. \tag{5.40}$$

From (5.37) and (5.40), we obtain:

$$y_{i+1} - y_i = b_i h_i + c_i h_i^2 + \frac{1}{3}\left(c_{i+1} - c_i\right) h_i^2,$$

hence

$$b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{1}{3}\left(c_{i+1} + 2c_i\right) h_i, \qquad i = 0, ..., n-1. \tag{5.41}$$

– From the continuity of $\frac{ds}{dx}\,(x)$ at the intermediate nodes:

$$b_{i+1} = b_i + 2c_i h_i + 3d_i h_i^2, \qquad i = 0, ..., n-2. \tag{5.42}$$

Inserting now the equalities (5.41) and (5.40) (which define b_i and d_i) into (5.42), we obtain:

$$\frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{h_{i+1}}{3}(c_{i+2} + 2c_{i+1}) =$$
$$= \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{3}\left(c_{i+1} + 2c_i\right) + 2c_i h_i + (c_{i+1} - c_i) h_i,$$

and further:

$$\frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_i} = (h_i - \frac{2h_i}{3})c_i + (\frac{2h_{i+1}}{3} + \frac{2h_i}{3})c_{i+1} + \frac{h_{i+1}}{3}c_{i+2}.$$

Multiplying both sides by 3, we obtain finally:

$$3(\frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_i}) = h_i c_i + 2(h_{i+1} + h_i)c_{i+1} + h_{i+1}c_{i+2},$$

for $i = 0, 1, ..., n - 2$. Therefore, we have finally $n - 1$ equations for $n + 1$ coefficients c_i (c_n is also present here), and two additional equations stemming from the boundary conditions (5.21b):

$$\frac{d^2s}{dx^2}\left(a^+\right) = 2c_0 = \alpha_2,$$

$$\frac{d^2s}{dx^2}\left(b^-\right) = 2c_n = \beta_2.$$

Define

$$v_i \overset{\text{df}}{=} 3\left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right), \quad i = 1, ..., n - 1,$$

$$u_i \overset{\text{df}}{=} 2(h_i + h_{i-1}), \quad i = 1, ..., n - 1,$$

then we obtain finally the system of linear equations in the following form:

$$\begin{bmatrix} u_1 & h_1 & 0 & \cdots & & 0 \\ h_1 & u_2 & h_2 & & & 0 \\ 0 & h_2 & u_3 & h_3 & & 0 \\ & & & \ddots & & \\ \vdots & & & & u_{n-2} & h_{n-2} \\ 0 & 0 & 0 & \cdots & h_{n-2} & u_{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} v_1 - 0.5h_0\alpha_2 \\ v_2 \\ \\ \vdots \\ v_{n-2} \\ v_{n-1} - 0.5h_{n-1}\beta_2 \end{bmatrix}. \tag{5.43}$$

This allows us to calculate the coefficients c_i, then the coefficients b_i and d_i are obtained by simple substitutions, using (5.41) and (5.40), respectively. □

The spaces of spline functions*

Denote $S_m(\triangle_n)$ the set of all spline functions of an order m, defined over an interval $[a, b]$, for every function with the same division $\triangle_n$ into subintervals. This set can be treated as a *finite dimensional linear space*.

An example of such a space, for $m = 1$, is *the set of all spline functions defined over a fixed division of a given interval* $[a, b]$. We shall constrain our considerations to equally spaced divisions of the interval , i.e., $x_i = x_0 + ih$, $i = 0, ..., n$ ($x_0 = a$, $x_n = b$, $h = \frac{b-a}{n}$). Any piecewise linear spline function can then be presented as a linear combination of the basis functions $\Phi_j(x)$, $j = 0, ..., n$, of the space $S_1(\triangle_n)$:

$$s(x) = \sum_{j=0}^{n} y_j \Phi_j(x),$$

where the shapes of the basis functions are depicted in Fig. 5.8. A sample piecewise linear spline function is shown in Fig. 5.9.

*Optional.

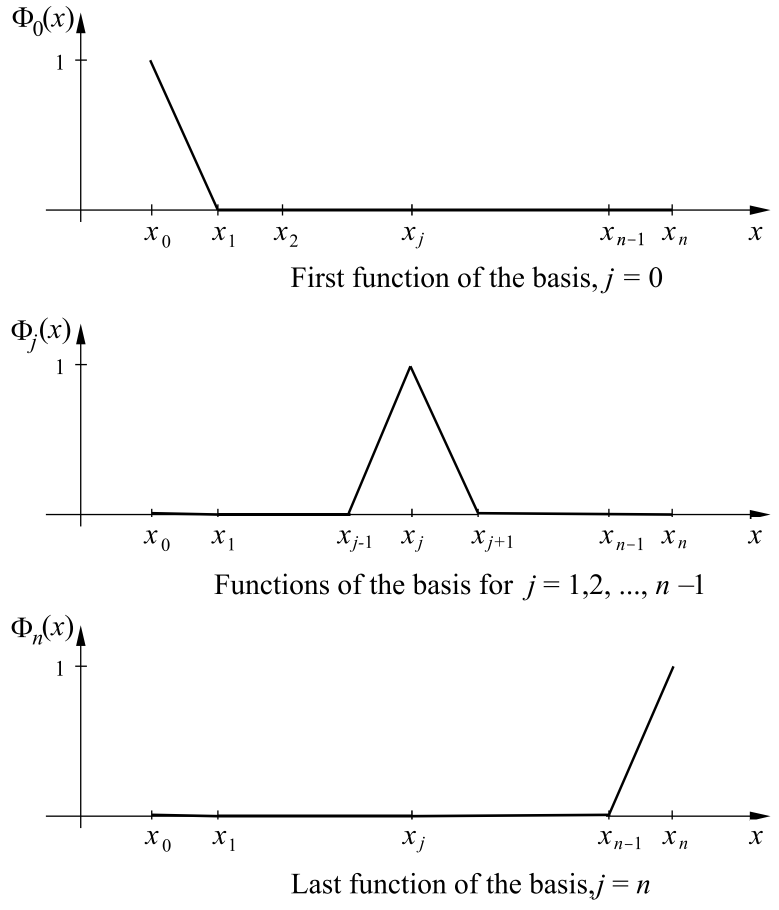

$\Phi_0(x)$
1
x_0 x_1 x_2 x_j x_{n-1} x_n x
First function of the basis, $j = 0$
$\Phi_j(x)$
1
x_0 x_1 x_{j-1} x_j x_{j+1} x_{n-1} x_n x
Functions of the basis for $j = 1,2, ..., n-1$
$\Phi_n(x)$
1
x_0 x_1 x_j x_{n-1} x_n x
Last function of the basis, $j = n$

Figure 5.8. The shapes of the basis functions of the space of spline functions of the first order, over the interval $[x_0,\ x_n]$

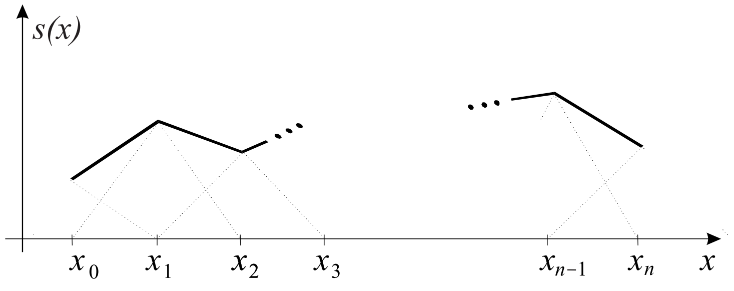


Figure 5.9. A sample piecewise linear spline function

The discussed space $S_1(\triangle_n)$ is $n+1$ dimensional, every its element (a spline function $s(x)$) is fully determined by $n+1$ numbers $y_0, y_1, ..., y_n$. Therefore, if $f(x)$ is a function to be determined over the interval $[a, b]$ (e.g., satisfying certain optimality requirements) and it is constrained to the class of the spline functions of the first order (is piecewise linear) – then an infinite dimensional problem of finding such a function (e.g., in dynamic optimization, etc.) becomes a finite $(n+1)$ dimensional one. The situation is similar for the spline functions of higher orders, only the basis functions are different. The discussed approach is of a significant practical importance. The cubic spline functions (spline functions of third order, $m = 3$) are most important in applications.

Problems

1. Find the interpolating polynomial for the interpolation nodes $x_i = -2, 1, 2, 4$ and the corresponding function values $3, 1, -3, 8$, using:
   a) the Lagrange interpolation formula,
   b) the Newton's interpolation formula.

2. Modify the interpolating polynomial from Problem 1, adding:
   a) the additional interpolation node $x = 6$, with the function value 6,
   b) the additional interpolation node $x = 3$, with the function value -2.

3. Calculate $\sqrt{117}$ interpolating the function $\sqrt{x}$ at the following interpolation nodes:
   a) $81, 100, 121,$
   b) $100, 121, 144,$
   c) $100, 121, 144, 169.$

4. Evaluate the spline function $s(x)$ of second order with the interpolation nodes $x_i = 1, 2, 4, 5$ and the function values $2, 3, 1, 1$, respectively. Additional condition on the derivative: $\frac{ds}{dx}(1^+) = 1$.

5. Evaluate the spline function $s(x)$ of third order on the interval $[0, 6]$ with the partition $x_i = 0, 2, 4, 6$ and the corresponding function values $2, 10, 4, 4$, respectively, and under additional conditions $\frac{ds}{dx}(0^+) = 2$, $\frac{ds}{dx}(6^-) = 17$.

6.* Derive a general system of linear equations defining the coefficients of the cubic spline function, corresponding to the system of equations (5.43), but for the additional conditions (5.21d) the (*not-a-knob* conditions).

7.* Derive a general system of linear equations defining the coefficients of the cubic spline function, similarly as the system of equations (5.43) was derived, but for the additional conditions (5.21a) (the boundary conditions on the first order derivatives).

*Optional.

Chapter 6

Nonlinear equations and roots of polynomials

6.1. Solving a nonlinear equation

Iterative methods for finding a solution of a nonlinear equation $f(x) = 0$ are subject of this section, such solutions are usually called *roots*, or *zeros*, of the nonlinear function $f(x)$. To find a root, an interval wherein it is located (isolated), should be first identified – this phase is called *root bracketing*. If the root bracketing can be performed in an interactive user-computer process, then the simplest way is to apply a graphical program, draw the function $f(x)$ (approximately) and select isolation intervals of all roots of the function. If one wants to implement this process algorithmically (without a user interaction), then checking values of the function on both ends of an interval is a core of the procedure – if a function $f(x)$ is continuous on a closed interval $[a, b]$ and $f(a) \cdot f(b) < 0$, then at least one root of $f(x)$ is located within $[a, b]$.

If we start with a (small) interval $[a = x_1, b = x_2]$, not containing a root ($f(x_1) \cdot f(x_2) > 0$), then a reasonable way to proceed is to expand the interval. The following algorithm (written in MATLAB language) is an example solution to this problem (where $\beta > 1$ is an expansion parameter):

```
for j=1:n
   if f(x$_1$)$\cdot$f(x$_2$) $<$ 0
     a:=x$_1$;
     b:=x$_2$;
   elseif abs(f(x$_1$)) $<$ abs(f(x$_2$))
     x$_1$:=x$_1$+$\beta$(x$_1$-x$_2$);
   else
     x$_2$:=x$_2$+$\beta$(x$_2$-x$_1$);
   end
end
```

Having found an interval containing a root we can initiate an *iterative method* to find it. *Iterative* means that we start with an initial approximation x_0 of the root α (which can be, e.g., the middle point of the interval) and successively improve this approximation, $x_n \underset{n\longrightarrow\infty}{\longrightarrow} \alpha$.

Generally, *order of convergence* of an iterative method is defined as the largest number $p \geq 1$ such that

$$\lim_{n\to\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^p} = k < \infty. \tag{6.1}$$

where the coefficient k is the *convergence factor*. The larger the order of convergence, the better the convergence of the method. If $p = 1$, we say that the method is convergent *linearly* (in this case $k < 1$ is required for convergence), if $p = 1$ and k converges to zero then the method is converging *superlinearly*. If $p = 2$, the convergence is *quadratic*. For $k > 0$, the formula (6.1) can also be written in a practical (but approximate) form:

$$|x_{n+1} - \alpha| \approx k \cdot |x_n - \alpha|^p , \tag{6.2}$$

where the closer x_n to α, the more accurate this approximate equality.

Iterative methods for nonlinear problems are, generally, only *locally convergent*. *A convergence ball* of an iterative method is defined as a neighborhood of the solution (of the root) α of such a radius δ, that for every initial point x_0 satisfying $\|x_0 - \alpha\| \leq \delta$ the method is convergent to α. The convergence ball is, in general, a subset of the *set of attraction* of the method – a set of all initial points, for which the method is convergent to α.

6.1.1. Bisection method

Let $[a, b] = [a_0, b_0]$ be an initial interval containing a root α of the function f. In the *bisection method*, at every (n-th) iteration:

1. The current interval containing the root, $[a_n, b_n]$, is divided into two equal subintervals, with the division point c_n located in the middle of the interval,

$$c_n = \frac{a_n + b_n}{2},$$

   then the value of the function at the new point $f(c_n)$ is evaluated.
2. Products $f(a_n)f(c_n)$ and $f(c_n)f(b_n)$ are evaluated, then the next interval is chosen as the subinterval corresponding to the product with the negative value. The endpoints of this interval are denoted a_{n+1}, b_{n+1}.

The procedure is repeated until $f(x_{n+1}) \leq \delta$, where δ is an assumed solution accuracy. This test may occur imprecise when the function is very ”flat”, i.e., its derivative has small absolute value in a neighbourhood of the root. Therefore, checking the length of the interval, $[b_n - a_n]$, is also recommended, assuming that it should be sufficiently small.

The accuracy of the solution obtained by the bisection method depends only on the number of iterations performed, it does not depend on the accuracy of calculations of the function values. Let ε_i denote the length of the interval at the n-th iteration, $n = 0, 1, 2, \ldots$, $\varepsilon_0 = [a, b]$. Then

$$\varepsilon_{n+1} = \frac{1}{2}\varepsilon_n, \tag{6.3}$$

which shows that the bisection method is linearly convergent ($p = 1$), with the convergence factor $k = \frac{1}{2}$.

6.1.2. Regula falsi method

The *regula falsi* method, called also the *false position* method, is very similar to the bisection method – the difference is that a current interval $[a_n, b_n]$ isolating the root α is also divided into two subintervals, but now by the point of intersection of the secant line connecting, on the plane (f, x), the points $(f(a_n), a_n)$ and $(f(b_n), b_n)$ with the x-axis. Denote c_n the point of intersection, as illustrated in Fig. 6.1.

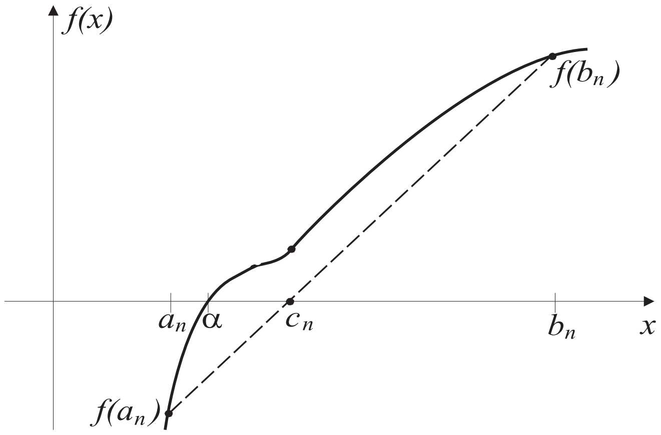


Figure 6.1. A construction of two subintervals by the secant line

It follows directly from the construction shown in Fig. 6.1 that:

$$\frac{f(b_n) - f(a_n)}{b_n - a_n} = \frac{f(b_n) - 0}{b_n - c_n},$$

thus

$$c_n = b_n - \frac{f(b_n)(b_n - a_n)}{f(b_n) - f(a_n)} = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}. \tag{6.4}$$

The choice of the next root isolation interval is performed as in the bisection method. The products of function values at the endpoints of the subintervals are calculated and that subinterval is selected for the next iteration, which corresponds to the negative product value. An example of first iterations of the method is shown in Fig. 6.2, where the first (bracketing) interval is $[a_0, b_0] = [a, b]$, and next points are denoted by the subsequent numbers 1,2,3, etc.

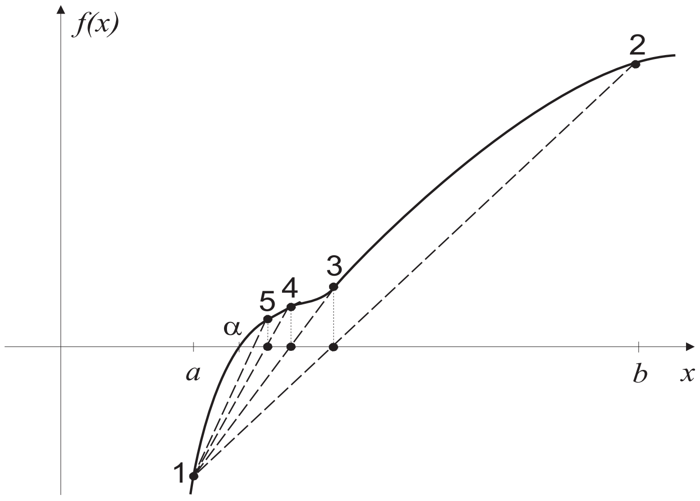


Figure 6.2. An illustration of the regula falsi method

The regula falsi method, as the bisection method, is always convergent, because it always chooses and shortens the interval containing the root. It is linearly convergent (with $p = 1$), provided the function is continuous and differentiable. However, the convergence may become very slow. Fig. 6.2 shows the case, when one endpoint (a) of the subsequent intervals remains unchanged and the iterations do not lead to shortening the intervals to zero – the interval $[a, \alpha]$ is the limit. This usually leads to a slow convergence (even very slow, as in the case of barrier functions, used e.g., in constrained optimization methods).

There is a simple modification of the standard formula (6.4) which avoids the described situation and thus improves the convergence of the method. The remedy is to take a smaller value of the function at the "frozen" end of the subsequent intervals, namely

$$c_n = \frac{a_n \frac{f(b_n)}{2} - b_n f(a_n)}{\frac{f(b_n)}{2} - f(a_n)} \tag{6.5a}$$

for the frozen (at least for two iterations) right end, and

$$c_n = \frac{a_n f(b_n) - b_n \frac{f(a_n)}{2}}{f(b_n) - \frac{f(a_n)}{2}} \tag{6.5b}$$

for the frozen left end (the case shown in Fig. 6.2). The algorithm combining the standard (6.4) and, if needed, the modified (6.5a)-(6.5b) iterations of the regula falsi method is called the *modified regula falsi* algorithm (also *the Illinois algorithm*). It is superlinearly convergent (see (6.1)) while retaining the global convergence. Moreover, the length of the subsequent intervals converges to zero.

6.1.3. Secant method

In the *secant method*, a secant line joins always *the two last obtained points*, as it is shown in Fig. 6.3. Therefore, it may happen that a next interval does not contain the root. If the two last points are denoted x_{n-1} and x_n, then a new point in the secant method is defined by a formula identical with (6.4), i.e.,

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} = \frac{x_{n-1}f(x_n) - x_n f(x_{n-1})}{f(x_n) - f(x_{n-1})}. \tag{6.6}$$

The order of convergence of the secant method is $p = (1+\sqrt{5})/2 \approx 1.618$, thus the method is, in general, better convergent than the bisection and the regula falsi methods. However, it is only *locally convergent* – therefore, a lack of convergence may happen, if the initial isolation interval of the root is not sufficiently small.

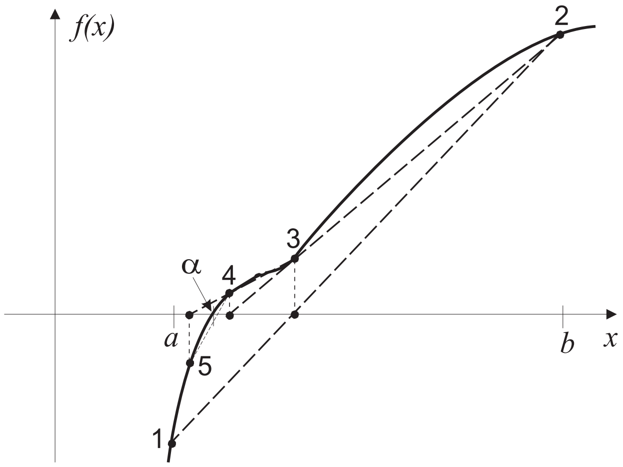


Figure 6.3. An illustration of the secant method

6.1.4. Newton's method

The Newton's method, called also *the tangent method*, operates by approximation of the function $f(x)$ by the first order part of its expansion into a Taylor series at a current point x_n (a current approximation of the root),

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n). \tag{6.7}$$

The next point, x_{n+1}, results as a root of the obtained linear function:

$$f(x_n) + f'(x_n)(x_{n+1} - x_n) = 0,$$

which leads to the iteration formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{6.8}$$

The Newton's method is illustrated in Fig. 6.4.

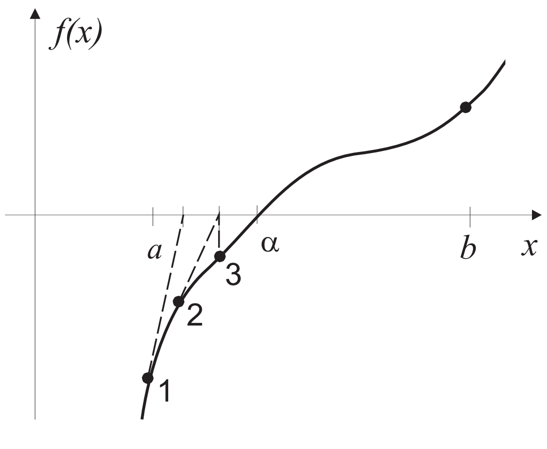


Figure 6.4. An illustration of the Newton's method

The Newton's method is *locally convergent* – if an initial point is too far from the root (outside its set of attraction) than a divergence may occur, as it is illustrated in Fig. 6.5. However, if the Newton's method is convergent, then it is usually very fast, as the convergence is quadratic (i.e., with the order $p = 2$).

The Newton's method is particularly effective when the function derivative at the root is sufficiently far from zero (the slope of the curve of $f(x)$ is steep). On the other hand, it is not recommended when the derivative $f'(\alpha)$ is close to zero, as then it is very sensitive to numerical errors when close to the root.

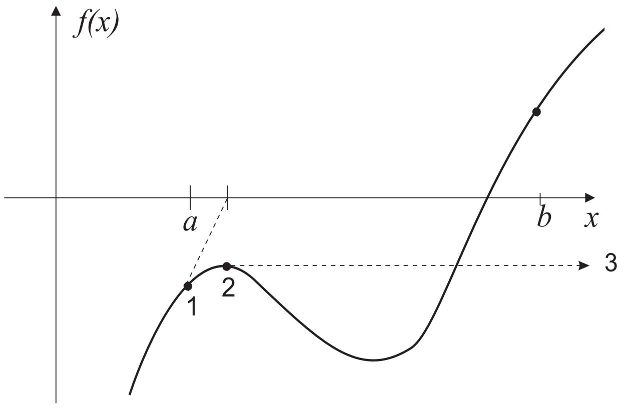


Figure 6.5. The Newton's method – a case of a divergence

Proof[*] of the quadratic convergence of the Newton's method.
Subtract first the root α from both sides of the equality (6.8), then take a truncated Taylor series expansions of the function $f\left(x_n\right)$ and its derivative $f'\left(x_n\right)$ at the point α. Then we have, assuming $f'\left(\alpha\right) \neq 0$:

$$\begin{aligned} x_{n+1} - \alpha &= (x_n - \alpha) - \frac{f\left(x_n\right)}{f'(x_n)} \\ &= \frac{f'(x_n)\,(x_n - \alpha) - f\left(x_n\right)}{f'(x_n)} \\ &= \frac{[f'(\alpha) + f''(\xi_n)\,(x_n - \alpha)](x_n - \alpha)}{f'(x_n)} + \\ &\qquad - \frac{f'(\alpha)\,(x_n - \alpha) + \frac{1}{2} f''(\zeta_n)\,(x_n - \alpha)^2}{f'(x_n)} \\ &= \frac{(f''(\xi_n) - \frac{1}{2} f''(\zeta_n))(x_n - \alpha)^2}{f'(x_n)} \\ &\approx \frac{f''(\zeta_n)}{2f'(x_n)}(x_n - \alpha)^2, \end{aligned}$$

where $\xi_n \in [x_n,\ \alpha],\ \ \zeta_n \in [x_n,\ \alpha]$. Therefore

$$\lim_{n\to\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^2} = \frac{|f''\left(\alpha\right)|}{2|f'\left(\alpha\right)|}.$$

[*]Optional.

Thus, quadratic convergence of the Newton's method has been shown,

$$|x_{n+1} - \alpha| \approx \frac{|f''(\alpha)|}{2|f'(\alpha)|} \cdot |x_n - \alpha|^2 . \tag{6.9}$$

□

6.1.5. An example realization of an effective algorithm

The secant and the Newton's method are only *locally convergent* – but, if convergent, they are more efficient (faster) than the bisection method. On the other hand, the bisection method is *globally convergent* provided it starts from an interval containing al least one root (and the function is continuous). Moreover, its convergence is stable, with the same convergence rate. Therefore, appropriate combination of a fast but locally convergent method with the slower but globally convergent bisection method should result in an efficient globally convergent algorithm. An example of such a combination is given in Fig. 6.6.

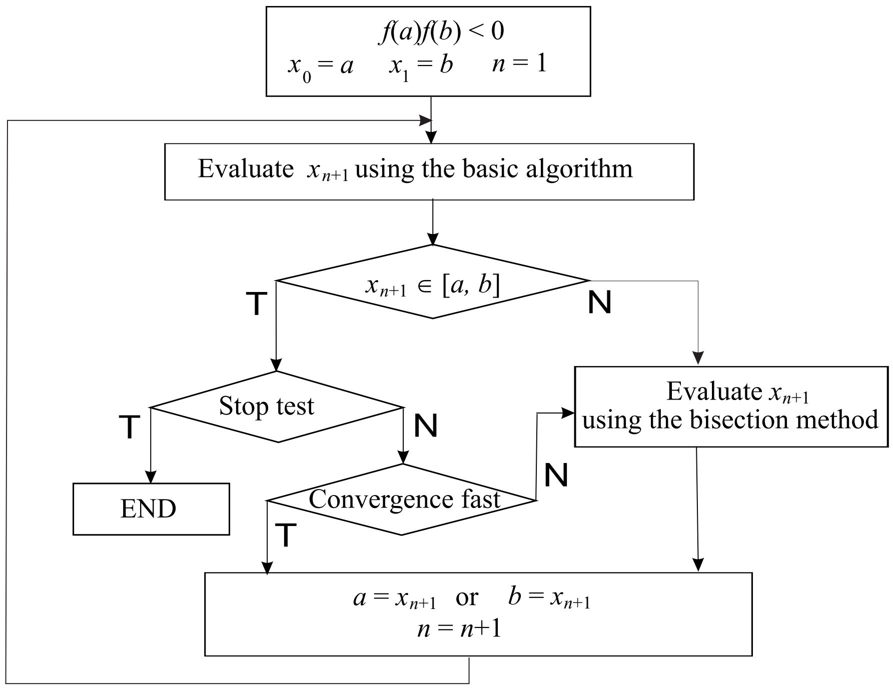

$f(a)f(b) < 0$
$x_0 = a \quad x_1 = b \quad n = 1$
Evaluate x_{n+1} using the basic algorithm
$x_{n+1} \in [a, b]$
T
N
Stop test
T
N
END
Convergence fast
N
T
Evaluate x_{n+1} using the bisection method
$a = x_{n+1}$ or $b = x_{n+1}$
$n = n+1$

Figure 6.6. Structure of an efficient globally convergent root-finding algorithm

An adequate design of the ”convergence fast” test is crucial for the effectiveness of the algorithm presented by the block-diagram shown in Fig. 6.6. A very simple realization could be as follows:

$$|f(x_{n+1})| < \beta\, |f(x_n)|, \qquad \text{where, e.g., } \beta = \frac{1}{2}. \tag{6.10}$$

A solution for the Newton’s method could be the following test (a justification is left to the reader):

$$|f(x_{n+1})| < \frac{1}{2}\left|(b-a)\cdot f'(x_{n+1})\right|. \tag{6.11}$$

If we are interested in a simpler and robust method, then the modified regula falsi method is a sound alternative. It is usually much faster than the bisection method, but also globally convergent.

6.2. Systems of nonlinear equations

Consider first a system of two nonlinear equations, i.e., $\mathbf{x} = [x_1, x_2]^{\mathrm{T}} \in \mathbb{R}^2$,

$$\mathbf{f}(\mathbf{x}) \overset{\mathrm{df}}{=} \begin{bmatrix} f_1(x_1,x_2) \\ f_2(x_1,x_2) \end{bmatrix} = \mathbf{0}.$$

An illustration of possible solutions to such a system of equations is presented in Fig. 6.7, where the curves $f_1(x_1, x_2) = 0$ (thin dotted lines) and $f_2(x_1, x_2) = 0$ (thick solid lines) are shown. In the case depicted in the figure there are six solutions located at intersection points of these curves, denoted by the letters from A to F. Note that if the solid lines representing zeros of the second equation $f_2(\mathbf{x}) = 0$ had been modified as shown by the dashed lines (an almost identical behaviour of the function f_2!), then the system of nonlinear equations would have had no solutions. Note also that the values of both nonlinear functions are close to zero at the point M, although the system of equations has not a solution in a rather large neighborhood of this point.

When looking for all solutions of a set of nonlinear equations, it is recommended to begin with an analysis (e.g., graphical, if the dimension of a problem enables it) of the whole set X of the variables x, where potential solutions can occur. This analysis should result in a determination of subsets containing single solutions. Then a fast but locally convergent method is applied, which finds individual solutions starting from initial points located in the determined subsets, one after another – such methods will be presented further in this Chapter. A simpler, but more numerically demanding solution is to divide, arbitrarily, the set X into smaller subsets, then to apply a fast but locally convergent method which will start from the

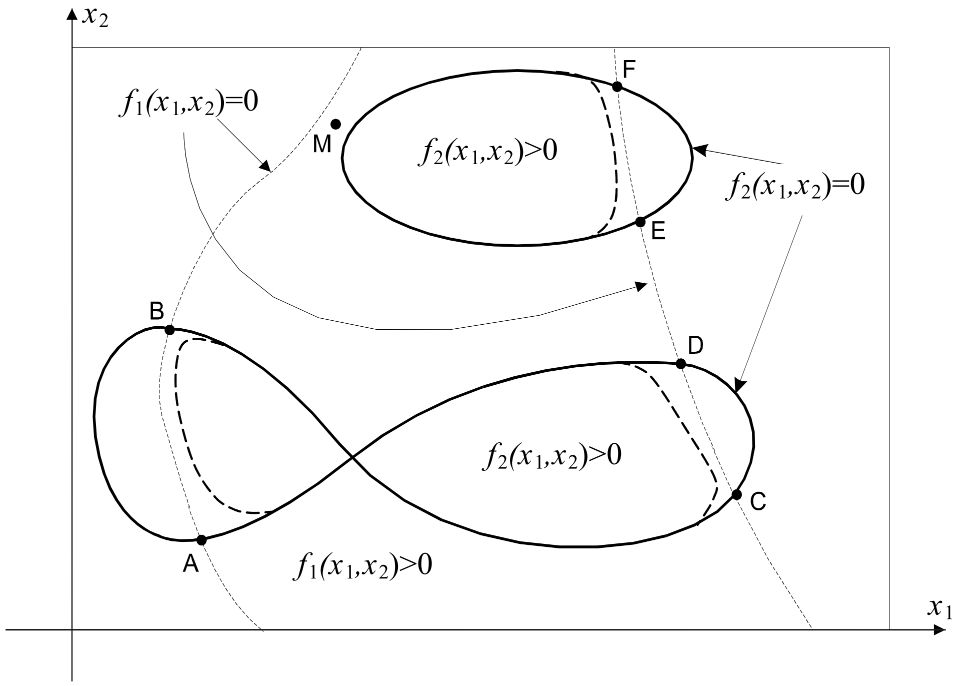


Figure 6.7. Illustration of an example system of two nonlinear equations

points located in all these subsets and will find all individual solutions. Certainly, the more dense the division of X the higher the chance to find all solutions – but also the larger the numerical burden. A stochastic approach can also be applied, when a fast but locally convergent method starts from a number of initial points, randomly chosen within X. Also in this case, the larger the number of points, the higher the chance to find all solutions.

The problem of solving a system of nonlinear equations is closely related to the problem of a minimization of a nonlinear function of many variables, sometimes it is even more difficult. Moreover, formulation of this problem as a minimization problem, e.g., a minimization of the performance function

$$\min_{x_1 \cdots x_n} \sum_{i=1}^{n} f_i\left(x_1, ..., x_n\right)^2 \tag{6.12}$$

does not make finding the solution easier, as the function (6.12) has usually several local minima, at which the function value is greater than zero (e.g., a local minimum point corresponding to the point M from Fig. 6.7).

6.2.1. Newton's method

The mapping $\mathbf{f}:\ \mathbb{R}^n \longrightarrow \mathbb{R}^n$, $f(\mathbf{x}) = [f_1(\mathbf{x}), ..., f_n(\mathbf{x})]$ is expanded into the Taylor series, then all nonlinear terms are skipped. We have then a linear approximation of the function around a current point $\mathbf{x}_k$:

$$\mathbf{f}(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx \mathbf{f}(\mathbf{x}_k) + \mathbf{f}'(\mathbf{x}_k)\,\Delta\mathbf{x}_k,$$
$$\mathbf{f}'(\mathbf{x}_k) = \left[\frac{\partial f_i}{\partial x_j}(\mathbf{x}_k)\right]_{nxn},$$

where $\mathbf{f}'(\mathbf{x}_k)$ is a matrix consisting of partial derivatives of the functions $f_i(\mathbf{x})$, $i = 1, ..., n$, called the Jacobian matrix. Then, analogously as it was in the scalar case, $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$ is determined as a solution to the system of linear equations

$$\mathbf{f}(\mathbf{x}_k) + \mathbf{f}'(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = 0, \qquad (6.13)$$

or, equivalently,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[\mathbf{f}'(\mathbf{x}_k)\right]^{-1}\mathbf{f}(\mathbf{x}_k). \qquad (6.14)$$

In practice, we do not invert the matrix but we find a solution to the system of linear equations, with respect to $\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$:

$$\mathbf{f}'(\mathbf{x}_k)\,\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}_k),$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_\mathbf{k}.$$

The Newton's method is locally convergent, but the order of convergence is two, which can be written as the asymptotic property:

$$\|\mathbf{x}_{k+1} - \alpha\| \cong k\,\|\mathbf{x}_k - \alpha\|^2, \qquad (6.15)$$

where k is a convergence factor (comp. 6.9). The Newton's method does not use other (previous) points (is ”memoryless”), but requires analytic formulae for the derivatives of the functions f_i, $i = 1, ..., n$. There are simplified versions of the Newton's method, as well as certain augmentations to assure a global convergence.

Discrete Newton's method

Each element $\frac{\delta f_i}{\delta x_j}(\mathbf{x}_k)$ of the matrix $\mathbf{f}'(\mathbf{x}_k)$ is apprpximated by a finite difference (see Section 8.1):

$$\Delta_j f_i(\mathbf{x}_k) = \frac{f_i(\mathbf{x}_k + h_j\mathbf{e}_j) - f_i(\mathbf{x}_k)}{h_j}, \qquad (6.16)$$

where $\mathbf{e}_j$ denotes the j-th versor of the Cartesian space and h_j is a length of the step. This requires n additional calculations of the values of the functions from the left hand side of the system of equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, instead of calculations of the partial derivatives.

6.2.2. Broyden's method[*]

The idea of the method is to replace $\mathbf{f}'(\mathbf{x}_k)$ by a certain matrix $\mathbf{B}_k$, which elements can be calculated easier than the exact or approximate partial derivatives.

For a square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, a vector $\mathbf{b} \in \mathbb{R}^n$ and any two points $\mathbf{x}$, $\mathbf{x}' \in \mathbb{R}^n$ let us define

$$\mathbf{p} = \mathbf{x}' - \mathbf{x},$$
$$\mathbf{q} = \mathbf{A}\mathbf{x}' - \mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{p}.$$

Theorem 6.1. *For a square matrix* $\mathbf{B} \in \mathbb{R}^{n\times n}$*, the matrix* $\mathbf{B}'$ *defined by*

$$\mathbf{B}' = \mathbf{B} + \frac{(\mathbf{q} - \mathbf{B}\mathbf{p})\,\mathbf{p}^{\mathrm{T}}}{\mathbf{p}^{\mathrm{T}}\mathbf{p}} \tag{6.17}$$

satisfies

$$\mathbf{B}'\mathbf{p} = \mathbf{q}, \tag{6.18}$$

$$\left\|\mathbf{B}' - \mathbf{A}\right\|_2 \le \left\|\mathbf{B} - \mathbf{A}\right\|_2. \tag{6.19}$$

Treating $\mathbf{A}$ as a linear approximation of the Jacobian matrix $\mathbf{f}'(\mathbf{x})$ of the system of nonlinear equations, the Theorem specifies the way how to construct, iteratively, approximations $\mathbf{B}_k$ of $\mathbf{A}$. In this way we obtain

the Broyden's algorithm:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{d}_k,\ \mathbf{d}_k = \mathbf{B}_k^{-1}\mathbf{f}(\mathbf{x}_k), \tag{6.20}$$

$$\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k,$$
$$\mathbf{q}_k = \mathbf{f}(\mathbf{x}_{k+1}) - \mathbf{f}(\mathbf{x}_k),$$
$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{(\mathbf{q}_k - \mathbf{B}_k\mathbf{p}_k)\,\mathbf{p}_k^{\mathrm{T}}}{\mathbf{p}_k^{\mathrm{T}}\mathbf{p}_k}. \tag{6.21}$$

Results better than for the unit step length (called the Newton's step length), i.e., $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{d}_k$, can be obtained with an approximate optimization of this step length:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k\mathbf{d}_k, \tag{6.22}$$

where, e.g.,

$$\lambda_k = 2^{-j},\ \ j = \min\left\{i \ge 0 : \left\|\mathbf{f}\left(\mathbf{x}_k - 2^{-i}\mathbf{d}_k\right)\right\| < \left\|\mathbf{f}(\mathbf{x}_k)\right\|\right\}. \tag{6.23}$$

[*]Optional.

6.2.3. Fix point method

In many applications an equation or a system of equations of the form

$$\mathbf{x} - \mathbf{f}(\mathbf{x}) = 0 \tag{6.24}$$

can be encountered, or an equation or a system of equations can be relatively easily transformed to this form. Then, the following method of solution can be proposed, called the *fixed point iteration method*, or the *sequential substitution method*:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k). \tag{6.25}$$

The resulting sequence is locally convergent to a solution α, such that $\alpha = \mathbf{f}(\alpha)$ (i.e., α is a fixed point of the mapping $\mathbf{f}$), if and only if

$$\text{sr}\left[\mathbf{f}'(\alpha)\right] < 1, \tag{6.26}$$

where "sr" denotes the spectral radius of the matrix $\mathbf{f}'(\alpha)$ (comp. Theorem 2.2). The convergence can be often improved applying a relaxation scheme:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + s\left(\mathbf{f}(\mathbf{x}_k) - \mathbf{x}_k\right), \qquad s \in (0,1]. \tag{6.27}$$

where s is a relaxation coefficient.

The algorithm of the fix point iteration method is usually slower convergent than the algorithms of Newton's or Broyden's methods, but is much simpler. The Jacobi's and Gauss-Seidel algorithms for iterative solution of systems of linear equations $\mathbf{Ax} = \mathbf{b}$ are examples of linear fix point iteration methods, see Section 2.6. An another well known application: the corrector iterations in multistep predictor-corrector algorithms for systems of differential equations, see Section 7.2.

In general, if an iteration process is defined by the operator:

$$\mathbf{x}_{k+1} = \mathbf{G}(\mathbf{x}_k), \tag{6.28}$$

then it is locally asymptotically convergent to α, where $\alpha = \mathbf{G}(\alpha)$, if and only if

$$\text{sr}\left[\mathbf{G}'(\alpha)\right] < 1. \tag{6.29}$$

Therefore, the algorithm with relaxation is convergent if and only if

$$\text{sr}\left[\mathbf{I}+s\mathbf{f}'(\alpha) - s\mathbf{I}\right] = \text{sr}\left[(1-s)\,\mathbf{I}+s\mathbf{f}'(\alpha)\right] < 1. \tag{6.30}$$

6.3. Roots of polynomials

A polynomial in the following form will be considered:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0. \tag{6.31}$$

It has exactly n roots (zeros), which can be:
– real or complex (for a complex root its conjugate is also a root),
– single or multiple.

Any polynomial is a continuous and differentiable nonlinear function. Therefore, *any of the root-finding methods for nonlinear functions described in the previous section can be applied to find real roots of polynomials*. But there are methods developed specially for finding the roots of polynomials, capable to find complex roots also. The Müller's and the Laguerre's methods are among the most known and recommended.

6.3.1. Müller's method

The idea of the Müller's method is to approximate the polynomial locally in a neighbourhood of a root by a quadratic function. The method can be developed using a quadratic interpolation based on three different points, Therefore, it can be treated as a generalization of the secant method, where a linear interpolation based on two points was applied. However, an efficient realization basing on an information from one point only is also available, i.e., basing on values of the polynomial and its first and second derivative at a current point.

Two mentioned versions of the Müller's method, referenced as **MM1**and **MM2,** will be developed in the text to follow.

MM1. Let us consider three points x_0, x_1, x_2, together with the corresponding polynomial values $f(x_0)$, $f(x_1)$ and $f(x_2)$. A quadratic function (a parabola) is constructed that passes through these points, then the roots of the parabola are found and one of the roots is selected for the next approximation of the solution.

Without loss of generality, assume that x_2 is an actual approximation of the solution (the root of the polynomial). Introduce a new, incremental variable

$$z = x - x_2$$

and use the differences

$$z_0 = x_0 - x_2,$$
$$z_1 = x_1 - x_2.$$

The interpolating parabola defined in the variable z is considered,

$$y(z) = az^2 + bz + c. \tag{6.32}$$

Considering the three given points, we have

$$\begin{aligned} az_0^2 + bz_0 + c = y(z_0) &= f(x_0), \\ az_1^2 + bz_1 + c = y(z_1) &= f(x_1), \\ c = y(0) &= f(x_2). \end{aligned}$$

Therefore, the following system of 2 equations must be solved to find a and b:

$$az_0^2 + bz_0 = f(x_0) - f(x_2), \tag{6.33}$$

$$az_1^2 + bz_1 = f(x_1) - f(x_2). \tag{6.34}$$

The roots of (6.32) are given by

$$z_+ = \frac{-2c}{b + \sqrt{b^2 - 4ac}}, \tag{6.35}$$

$$z_- = \frac{-2c}{b - \sqrt{b^2 - 4ac}}. \tag{6.36}$$

The root that has a smaller absolute value (i.e., nearer to the assumed current best root approximation x_2) is chosen for the next iteration,

$$x_3 = x_2 + z_{\min},$$

where

$$\begin{aligned} z_{min} &= z_+, \quad \text{if } |b + \sqrt{b^2 - 4ac}| \geq |b - \sqrt{b^2 - 4ac}|, \\ z_{min} &= z_-, \quad \text{in the opposite case.} \end{aligned}$$

For the next iteration the new point x_3 is taken, together with those two from points selected from x_0, x_1, x_2 which are closer to x_3.

It should be noted that the algorithm should work properly also in cases when $b^2 - 4ac < 0$ occurs, as this is a standard situation, leading to iterations towards a complex root. Therefore, algorithms of the Müller's method should be implemented in a complex number arithmetic. The formulae given above can be directly used also in this arithmetic.

MM2. Another version of the Müller's method, *using values of a polynomial and of its fist and second order derivatives at one point only* is often recommended. It is slightly more effective numerically, because it is numerically slightly more expensive to calculate values of a polynomial at three different points than to calculate values of a polynomial and of its first and second derivatives at one point only.

It follows directly from the definition of the parabola (6.32), for $z \overset{\text{df}}{=} x - x_k$, that at the point $z = 0$:

$$y(0) = c = f(x_k),$$
$$y'(0) = b = f'(x_k),$$
$$y''(0) = 2a = f''(x_k),$$

which leads to the formula for the roots:

$$z_{+,-} = \frac{-2f(x_k)}{f'(x_k) \pm \sqrt{(f'(x_k))^2 - 2f(x_k)f''(x_k)}}. \tag{6.37}$$

To approximate the solution α, a root with smaller absolute value is chosen:

$$x_{k+1} = x_k + z_{min}, \tag{6.38}$$

where z_{min} is selected from $\{z_+, z_-\}$ in identical way as it was done in the MM1 version.

The MM2 version of the Müller's method should be implemented in a complex number arithmetic, for the same reasons as it was for the MM1 version.

The Müller's method is locally convergent, the order of convergence is 1.84. Therefore, the method is (locally) more effective than the secant method, it is almost as fast as the Newton's method – and capable to find complex roots as well. Certainly, the Müller's method can not only be used to find real or complex roots of polynomials, but also of another nonlinear functions.

6.3.2. Laguerre's method

The method is defined by the following formula:

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) \pm \sqrt{(n-1)[(n-1)(f'(x_k))^2 - nf(x_k)f''(x_k)]}}, \tag{6.39}$$

where n denotes the order of the polynomial, and the sign in the denominator is chosen in a way assuring a larger absolute value of the denominator, as in the Müller's method (see also [3] for a slightly different formulation and an intuitive derivation).

It is easy to observe that the formulae (6.37)-(6.38) and (6.39) are similar. The Laguerre's formula (6.39) is slightly more complex, it takes also into account the order of the polynomial (it can be treated as an augmentation of (6.37)-(6.38)), therefore the Laguerre's methos is better, in general. In the case of polynomials with real roots only, the Laguerre's method is convergent starting from any real initial point, thus it is globally convergent. Despite a lack of formal analysis for a complex roots case, the numerical practice has shown good numerical properties also in this case (although situations of divergence may happen). The Laguerre's method is regarded as one of the best methods for polynomial root finding.

6.3.3. Deflation by a linear term

If we are looking for many or all roots of a polynomial, then after finding a single root α and before looking for a next one, the polynomial should be divided by the linear term $(x-\alpha)$. This is known as a *deflation by a linear term*. The lower order polynomial obtained in this way is simpler and does not contain the just calculated root (unless it is multiple). Accuracy of the deflation algorithm is important from a numerical point of view.

Denote by $Q\left(x\right)$ the polynomial obtained after dividing $f(x)$ by the linear term $(x-\alpha)$. We have

$$\begin{aligned} f\left(x\right) &= \left(x-\alpha\right)\cdot Q\left(x\right), \\ Q\left(x\right) &= q_n x^{n-1} + ... + q_2 x + q_1. \end{aligned}$$

The algorithms for a division of a polynomial $f(x)$ by a linear term $(x-\alpha)$:

The Horner's algorithm (standard):

$$\begin{aligned} q_{n+1} &\overset{\text{df}}{=} 0, \\ q_i &= a_i + q_{i+1}\alpha, \qquad i = n, n-1, ..., 0. \end{aligned} \tag{6.40}$$

The reversed Horner's algorithm (for $\alpha \neq 0$):

$$\begin{aligned} q_0 &\overset{\text{df}}{=} 0, \\ q_{i+1} &= \frac{q_i - a_i}{\alpha}, \qquad i = 0, 1, 2, ..., n-1. \end{aligned} \tag{6.41}$$

The combined Horner's scheme. The standard Horner's algorithm calculates the coefficients in descending order, starting from q_n, while the reversed algorithm calculates the coefficients in a reversed order, starting from q_1. Due to a cumulation of errors, every next coefficient is calculated with descending accuracy, in both schemes. Therefore, especially for polynomials of higher order, *the combined Horner's scheme is more accurate*:

– $q_n, q_{n-1}, ..., q_{k+1}$ are calculated using the standard Horner's scheme,
– $q_1, q_2, ..., q_k$ are calculated using the reversed Horner's scheme,

and the index k denoting the "connection" point can be (roughly) assumed as corresponding to the middle power term of the polynomial. However, the following rule may be more accurate: if

b_i – coefficients from the standard scheme,
c_i – coefficients from the reversed scheme,

then the index k can be defined as that resulting from the minimization

$$\frac{\left|c_k - b_k\right|^r}{\left|a_k\right|^r + \left|c_k\right|^r} = \min_{\left|a_j\right|^r + \left|c_j\right|^r > 0} \frac{\left|c_j - b_j\right|^r}{\left|a_j\right|^r + \left|c_j\right|^r}, \tag{6.42}$$

where $r = 1$ stands for real roots, while $r = 2$ for complex roots.

In should be pointed out that roots of polynomials should be calculated in ascending order of their absolute values (from smallest to largest), to minimize numerical errors.

To perform the deflation of a polynomial (with real coefficients) by a linear term involving a complex root, a complex number arithmetic must be used. There is an alternative: to divide the polynomial by a quadratic term corresponding to a pair of complex conjugate roots. It can be performed using the Bairstow's algorithm, see Section 6.3.5 further below.

6.3.4. Root polishing

The root polishing is a procedure applied to increase precision of the roots calculated from deflated polynomials (of a lower order). It is simple: starting from the root, we apply a standard method (e.g., Newton's, Laguerre's), but to the original (of full order n) polynomial. For pairs of complex roots, if one wants to avoid the complex arithmetic, the Bairstow's algorithm (Section 6.3.5) can be used, certainly with the original polynomial for each pair. The root polishing can be applied "on-line" – directly after every root (pair of roots) has been calculated.

6.3.5. Bairstow's algorithm*

The core of the algorithm is to look simultaneously for two polynomial roots and then the deflation by a quadratic term.

Denote

$$m(x) = m(x, p, r) = x^2 - px - r, \tag{6.43}$$

and the result of the division of $f(x)$ by $m(x)$ by $Q(x; p, r)$, i.e.,

$$f(x) = \left(x^2 - px - r\right) Q(x; p, r) + q_1(p, r)\, x + q_0(p, r). \tag{6.44}$$

We want to choose p and r in such a way that

$$q_1(p, r) = 0, \tag{6.45}$$

$$q_0(p, r) = 0. \tag{6.46}$$

The above system of two nonlinear equations is solved using the Newton's method:

$$\begin{bmatrix} p_{k+1} \\ r_{k+1} \end{bmatrix} = \begin{bmatrix} p_k \\ r_k \end{bmatrix} - \begin{bmatrix} \frac{\partial q_0}{\partial p}(p_k, r_k) & \frac{\partial q_0}{\partial r}(p_k, r_k) \\ \frac{\partial q_1}{\partial p}(p_k, r_k) & \frac{\partial q_1}{\partial r}(p_k, r_k) \end{bmatrix}^{-1} \begin{bmatrix} q_0(p_k, r_k) \\ q_1(p_k, r_k) \end{bmatrix}. \tag{6.47}$$

*Optional.

Since $f(x)$ is a fixed polynomial, it is independent of r, hence

$$\begin{aligned}\frac{\partial}{\partial r}f(x) \equiv 0 \quad = \quad & \left(x^2 - px - r\right)\frac{\partial Q}{\partial r}(x;p,r) - Q(x;p,r) + \\ & + \frac{\partial q_1}{\partial r}(p,r)\,x + \frac{\partial q_0}{\partial r}(p,r)\,, \quad (6.48)\end{aligned}$$

therefore

$$Q(x;p,r) = \left(x^2 - px - r\right)\frac{\partial Q}{\partial r}(x;p,r) + \frac{\partial q_1}{\partial r}(p,r)\,x + \frac{\partial q_0}{\partial r}(p,r)\,. \quad (6.49)$$

Similarly, since $f(x)$ is a fixed polynomial, it is independent of p, hence

$$\begin{aligned}\frac{\partial}{\partial p}f(x) \equiv 0 \quad = \quad & \left(x^2 - px - r\right)\frac{\partial Q}{\partial p}(x;p,r) - xQ(x;p,r) + \\ & + \frac{\partial q_1}{\partial p}(p,r)\,x + \frac{\partial q_0}{\partial p}(p,r)\,, \quad (6.50)\end{aligned}$$

therefore

$$xQ(x;p,r) = \left(x^2 - px - r\right)\frac{\partial Q}{\partial p}(x;p,r) + \frac{\partial q_1}{\partial p}(p,r)\,x + \frac{\partial q_0}{\partial p}(p,r)\,. \quad (6.51)$$

The structure of the k-th step of the Bairstow's algorithm

Given p_k, r_k:

1. $Q(x;p_k,r_k)$, $q_1(p_k,r_k)$ and $q_0(p_k,r_k)$ are calculated dividing $f(x)$ by the polynomial $m_k(x) = x^2 - p_k x - r_k$, for instance applying the formulae:

$$\begin{aligned} q_n &= a_n, \\ q_{n-1} &= pq_n + a_{n-1}, \\ q_i &= pq_{i+1} + rq_{i+2} + a_i, \qquad i = n-2, n-3, ..., 0, \end{aligned}$$

   where

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0 = (x^2 - p_k x - r_k)\cdot Q(x)\,,$$
$$Q(x) = q_n x^{n-2} + ... + q_3 x + q_2.$$

2. $\frac{\partial q_1}{\partial r}(p_k,r_k)$ and $\frac{\partial q_0}{\partial r}(p_k,r_k)$ are calculated, dividing $Q(x;p_k,r_k)$ by $m_k(x)$ according to (6.49),

3. $\frac{\partial q_1}{\partial p}(p_k,r_k)$ and $\frac{\partial q_0}{\partial p}(p_k,r_k)$ are calculated, e.g., dividing $xQ(x;p_k,r_k)$ by $m_k(x)$ according to (6.51) (or simpler, see the Remark below),

4. p_{k+1}, r_{k+1} are calculated according to (6.47), solving the system of two linear equations

$$\begin{bmatrix} \frac{\partial q_0}{\partial p}(p_k, r_k) & \frac{\partial q_0}{\partial r}(p_k, r_k) \\ \frac{\partial q_1}{\partial p}(p_k, r_k) & \frac{\partial q_1}{\partial r}(p_k, r_k) \end{bmatrix} \begin{bmatrix} p_{k+1} - p_k \\ r_{k+1} - r_k \end{bmatrix} = - \begin{bmatrix} q_0(p_k, r_k) \\ q_1(p_k, r_k) \end{bmatrix},$$

etc. until $q_1(p_k, r_k) = q_0(p_k, k_k) = 0$.

Remark: Step 3 can be performed in another way, avoiding a polynomial division. Namely, denote by x_1 and x_2 zeroes of the polynomial $m_k(x) = x^2 - p_k x - r_k$. Hence, from (6.49)

$$Q(x_i; p_k, r_k) = \frac{\partial q_1}{\partial r}(p_k, r_k) x_i + \frac{\partial q_0}{\partial r}(p_k, r_k), \qquad i = 1, 2.$$

Therefore, it follows from (6.50), for $i = 1, 2$,

$$-x_i \left(\frac{\partial q_1}{\partial r}(p_k, r_k) x_i + \frac{\partial q_0}{\partial r}(p_k, r_k) \right) + \frac{\partial q_1}{\partial p}(p_k, r_k) x_i + \frac{\partial q_0}{\partial p}(p_k, r_k) = 0,$$

i.e.,

$$-x_i^2 \frac{\partial q_1}{\partial r}(p_k, r_k) + x_i \left(-\frac{\partial q_0}{\partial r}(p_k, r_k) + \frac{\partial q_1}{\partial p}(p_k, r_k) \right) + \frac{\partial q_0}{\partial p}(p_k, r_k) = 0.$$

But, we have from the definition $x_i^2 = p_k x_i + r_k, \;\; i = 1, 2,$ thus

$$x_i \left(-\frac{\partial q_0}{\partial r}(p_k, r_k) + \frac{\partial q_1}{\partial p}(p_k, r_k) - p_k \frac{\partial q_1}{\partial r}(p_k, r_k) \right) + \\ -r_k \frac{\partial q_1}{\partial r}(p_k, r_k) + \frac{\partial q_0}{\partial p}(p_k, r_k) = 0.$$

This means that (at least for one $x_i \neq 0$)

$$-\frac{\partial q_0}{\partial r}(p_k, r_k) + \frac{\partial q_1}{\partial p}(p_k, r_k) - p_k \frac{\partial q_1}{\partial r}(p_k, r_k) = 0,$$
$$-r_k \frac{\partial q_1}{\partial r}(p_k, r_k) + \frac{\partial q_0}{\partial p}(p_k, r_k) = 0.$$

Therefore, we obtain finally the following simple formulae

$$\frac{\partial q_1}{\partial p}(p_k, r_k) = p_k \frac{\partial q_1}{\partial r}(p_k, r_k) + \frac{\partial q_0}{\partial r}(p_k, r_k), \tag{6.52}$$

$$\frac{\partial q_0}{\partial p}(p_k, r_k) = r_k \frac{\partial q_1}{\partial r}(p_k, r_k). \tag{6.53}$$

□

Problems

1. Implement (e.g., in MATLAB environment) the following methods: bisection, modified regula falsi, secant, Newton's. Apply these methods to calculate, with high precision, roots of the following functions (after preliminary finding the isolation intervals for each root, using a graphical approach, i.e., plotting the functions):
   a) $f(x) = 2x^3 - 2.5x - 5$, in the interval $[1, 2]$,
   b) $f(x) = e^{-x} - \sin(\pi x/2)$, in the interval $[0, 2.5]$.
   Compare and comment the results.

2. Implement (e.g., in MATLAB environment) the Newton's method for solving systems of nonlinear equations. Apply this program to find the solutions of the following system of equations:

$$\begin{aligned} -0.2x_1 + 0.2x_2 - x_3 + 6 &= 0, \\ x_1^2 + x_2^2 + x_3^2 - 36 &= 0, \\ x_1^2 + x_2^2 - 0.65^2(x_3 - 2)^2 &= 0. \end{aligned}$$

3.[*] Implement (e.g., in MATLAB environment) the Broyden's method for solving systems of nonlinear equations. Apply this program to find the solutions of the system of equations from the proceeding problem. Compare the results obtained for the two methods.

4. Implement (e.g., using MATLAB) the following polynomial root-finding methods : Müller's (both variants) and Laguerre's. Apply these methods to find, with high accuracy, all roots of the following polynomials (applying the linear deflation):
   a) $f(x) = x^4 - 5x^3 + 10x^2 - 10x + 4$,
   b) $f(x) = x^4 + x^3 - 5x^2 + 2x + 2$.
   Compare and comment the results

5.[*] Applying the Bairstow's method, find a pair of complex conjugate roots of the polynomial $f(x) = x^5 - 10x^4 + 35x^3 - 50x^2 + 24x$.

[*]Optional.

Chapter 7

Ordinary differential equations

Differential equations are commonly used for mathematical modeling of dynamic systems. Systems of differential equations describing real-life problems are usually nonlinear and analytic solutions are in most cases not possible to obtain – the only way is to calculate the solutions using numerical methods.

Numerical methods for finding a solution to *a system of first order ordinary differential equations (ODEs) with given initial conditions (the initial value problem, the Cauchy problem)* will be considered. The numerical methods for solving this problem are a basic tool used in *program packages for simulation of continuous time dynamic systems*. Moreover, they are also elements of algorithms for more complex systems of differential equations, as two-point boundary value problems, systems of partial differential equations, see, e.g, [3].

The independent variable will be denoted by $x \in \mathbb{R}^1$ – *it represents the time in most cases*, but it can also represent a spatial variable, e.g., if a considered system of ordinary differential equations stems from a system of partial differential equations after a "freezing" of the time variable at some value. The dependent variables (the solutions) will be denoted by $y_i\,(x)$, $i = 1, ..., m$. The considered system of ordinary differential equations will be presented in the form:

$$\frac{dy_i(x)}{dx} = f_i\,(x, y_1(x), ..., y_m(x))\,, \qquad i = 1, ..., m, \tag{7.1}$$

where $x \in [a, b]$, with initial conditions: $y_i\,(a) = y_{ia},\ i = 1, ..., m.$ We shall use a shortened notation, with the argument omitted in the solution functions:

$$\frac{dy_i}{dx} = f_i\,(x, y_1, ..., y_m)\,,\ \ y_i\,(a) = y_{ia},\ \ i = 1, ..., m,\ \ x \in [a, b]. \tag{7.2}$$

Denote $\mathbf{y} = [y_1\ y_2 \cdots y_m]^{\mathrm{T}} \in \mathbb{R}^m$, $\mathbf{f} = [\ f_1\ f_2 \cdots f_m]^{\mathrm{T}}$, $f_i : \mathbb{R}^{1+m} \to \mathbb{R}$, $i = 1, ..., m$, then the system of equations (7.2) can be presented in a more compact vector notation:

$$\mathbf{y}'(x) = \mathbf{f}\,(x, \mathbf{y})\,,\ \ \mathbf{y}\,(a) = \mathbf{y}_a,\ \ x \in [a, b]. \tag{7.3}$$

Theorem 7.1. *If the following conditions are satisfied:*

1. the function $\mathbf{f}$ *is continuous over the set*

$$\mathbf{D} = \{(x, \mathbf{y}) : \quad a \leq x \leq b, \quad \mathbf{y} \in \mathbb{R}^m\}, \tag{7.4}$$

2. the function $\mathbf{f}$ *satisfies the Lipschitz condition with respect to* $\mathbf{y}$*, i.e.,*

$$\exists\, L > 0 \quad \forall x \in [a, b], \quad \forall \mathbf{y}, \overline{\mathbf{y}} \quad \| \mathbf{f}(x, \mathbf{y}) - \mathbf{f}(x, \overline{\mathbf{y}}) \| \leq L \| \mathbf{y} - \overline{\mathbf{y}} \|, \tag{7.5}$$

then, for any initial condition $\mathbf{y}_a$*, there is a unique and continuously differentiable function* $\mathbf{y}\,(x)$*, such that*

$$\mathbf{y}'\,(x) = \mathbf{f}\,(x, \mathbf{y}(x)), \quad \mathbf{y}\,(a) = \mathbf{y}_a, \quad x \in [a, b]. \tag{7.6}$$

Moreover, it can be shown that under the assumptions of the above theorem, the solution $\mathbf{y}\,(x)$ depends in a continuous way on problem perturbations, i.e., on the initial conditions and parametric perturbations of the function $\mathbf{f}$.

The single-step methods and *multistep methods* are widely used classes of numerical methods for finding solutions of systems of nonlinear ordinary differential equations. Numerical algorithms belonging to these two classes will be further presented.

All numerical methods for finding solutions of the systems of differential equations are *difference methods*, i.e., approximate values of the solution functions are calculated at subsequent discrete points x_n only,

$$a = x_0 \leq x_1 \leq \cdots \leq x_n \leq \cdots\ b,$$

where $h_i = x_{i+1} - x_i$ are subsequent steps of a numerical method, which may be constant (equal) or varying.

Definition (convergence of the method). A numerical method for a system of differential equations is *convergent*, if for any system having a unique solution $y(x)$, and for the step length h_n converging to zero for all steps, the following holds:

$$\lim_{h_n \to 0} \mathbf{y}\,(x_n; h_n) \to \mathbf{y}\,(x), \tag{7.7}$$

where $\mathbf{y}(x_n; h_n)$ denotes the approximate solution found by the method.

The Euler's method is the simplest one, it is defined as follows

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h_n \mathbf{f}\,(x_n, \mathbf{y}_n), \qquad n = 0, 1, ..., \quad \mathbf{y}(x_0) = \mathbf{y}_a. \tag{7.8}$$

A graphical interpretation of this method is presented in Fig. 7.1, for the case of a single equation ($m = 1$) and for a fixed step-size $h_n = h = const.$

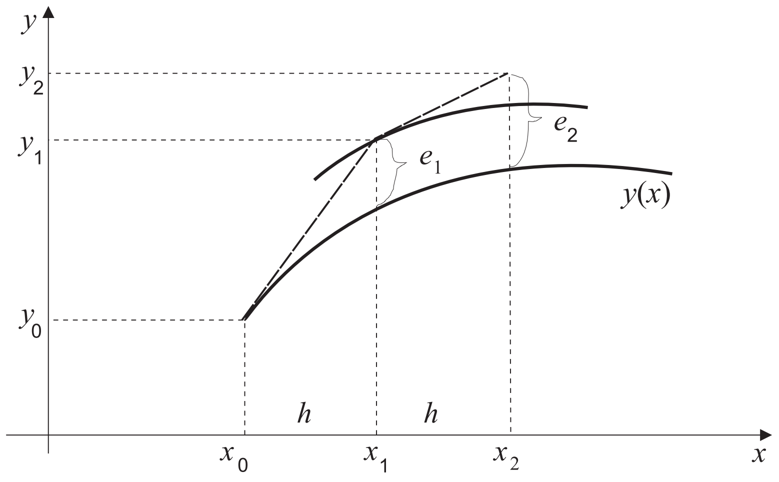


Figure 7.1. A graphical interpretation of the Euler's method

Example 7.1.[*] The convergence of the Euler's method will be proved, for one equation ($m = 1$), to simplify the presentation. Assume $y(x) \in C^2$ on $[a, b]$, and a fixed step-size $h_n = h,\ n = 0, 1, ...$

Expanding the solution into a finite Taylor series, we have

$$y\left(x_n + h\right) = y\left(x_n\right) + y'\left(x_n\right) h + y''\left(\xi_n\right) \frac{h^2}{2}, \qquad \xi_n \in \left[x_n, x_n + h\right]. \quad (7.9)$$

A global error of the method after n steps is defined as

$$e_n \overset{\text{df}}{=} y\left(x_n\right) - y_n.$$

Subtracting (7.8) from (7.9) and using the Lipschitz condition we get

$$\begin{aligned} e_{n+1} &= e_n + h\left[f\left(x_n, y\left(x_n\right)\right) - f\left(x_n, y_n\right)\right] + T_n, \\ \left|e_{n+1}\right| &\leq \left|e_n\right| + hL\left|e_n\right| + \left|T_n\right|, \end{aligned}$$

where L is the Lipschitz constant and

$$T_n = y''\left(\xi_n\right) \frac{h^2}{2}.$$

[*]Optional.

Define
$$M_z = \frac{1}{2} \max_{x\in[a,b]} \left| y''(x) \right| ,$$
then the following error estimation can be performed
$$\begin{aligned}
|e_{n+1}| &\le (1+Lh)\,|e_n| + h^2 M_z \\
&\le (1+hL)^2 |e_{n-1}| + h^2 M_z \left((1+Lh)+1\right) \\
&\le \;\cdots\; \le h^2 M_z \sum_{j=0}^{n} (1+Lh)^j \\
&= h^2 M_z \frac{1-(1+Lh)^{n+1}}{1-(1+Lh)} \\
&= hM_z \frac{(1+Lh)^{n+1}-1}{L}.
\end{aligned}$$
Further, assuming $L \neq 0$, substituting $n+1 = \frac{x_{n+1}-a}{h}$ and using a general identity $1+Lh < e^{Lh}$, we obtain
$$\begin{aligned}
|e_{n+1}| &\le hM_z \frac{(1+Lh)^{\frac{x_{n+1}-a}{h}}-1}{L} \\
&\le hM_z \frac{e^{L(b-a)}-1}{L} = \xi_0 h.
\end{aligned}$$
It proves convergence of the Euler's method, because $|e_{n+1}| \to 0$ for $h \to 0$. □

The Euler's method is the simplest but a least accurate one. The modified Euler's method (the midpoint method) and the Heun's method, presented below, are not much more complicated, but significantly more accurate.

The modified Euler's method (the midpoint method)
$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(x_n + \frac{1}{2}h,\; \mathbf{y}_n + \frac{1}{2}h\mathbf{f}\left(x_n, \mathbf{y}_n\right)). \tag{7.10}$$
A geometric interpretation of a single step of the modified Euler's method is shown in Fig. 7.2, where also the point y^E_{n+1} obtained by the Euler's (not modified) method is given, for a comparison. Errors generated by both methods in the presented single step (so called approximation errors (7.15), see Section 7.1) are also shown in the figure, denoted by r_n and r^E_n, respectively.

The Heun's method
$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{1}{2}h\left[\mathbf{f}\left(x_n, \mathbf{y}_n\right) + \mathbf{f}\left(x_n + h, \mathbf{y}_n + h\mathbf{f}\left(x_n, \mathbf{y}_n\right)\right)\right]. \tag{7.11}$$

It is left to the reader to interpret the Heun's method geometrically, and to compare this method with the Euler's methods, the standard and modified.

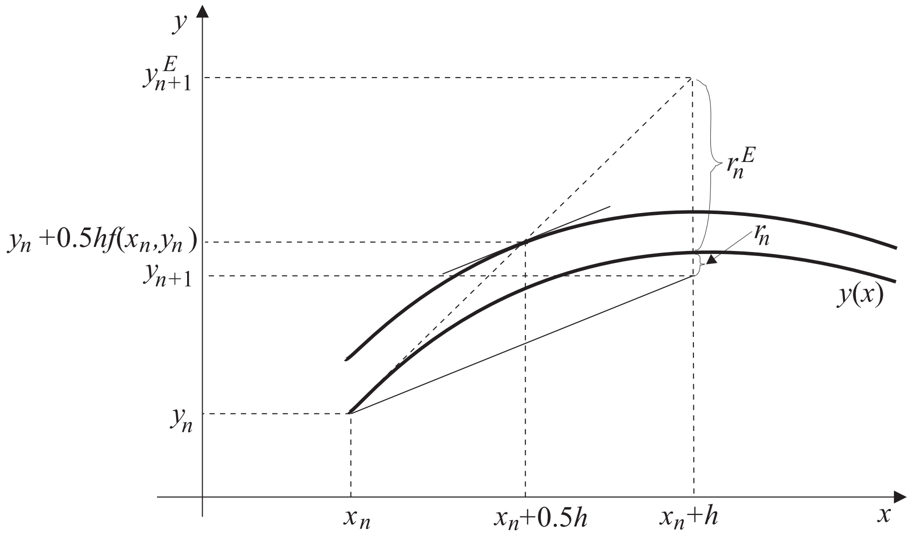


Figure 7.2. A graphical interpretation of one step of the modified Euler's method (results for the Euler's method, shown for a comparison, are denoted by the superscript "E")

Example 7.2. For the following system of two differential equations (an epidemic model):

$$\begin{aligned} {y_1}' &= -ay_1y_2, \\ {y_2}' &= ay_1y_2 - by_2, \end{aligned} \tag{7.12}$$

one (n-th) step of the modified Euler's method (the midpoint method) will be formulated, from the point x_n (the current time), with the solution at this point $y_n = [(y_1)_n, (y_2)_n]^{\mathrm{T}}$ (initial solution point for the n-th step), with the current step-size h_n.

It is convenient (effective) to calculate first the solution using the Euler's method with the step-size $\frac{1}{2}h_n$:

$$\begin{aligned} (y_1)_{n+\frac{1}{2}} &= (y_1)_n + \frac{1}{2}h_n[-a(y_1)_n(y_2)_n], \\ (y_2)_{n+\frac{1}{2}} &= (y_2)_n + \frac{1}{2}h_n[a(y_1)_n(y_2)_n - b(y_2)_n] \end{aligned}$$

and then the required equations of the modified Euler's method:

$$\begin{aligned} (y_1)_{n+1} &= (y_1)_n + h_n\left(-a(y_1)_{n+\frac{1}{2}}(y_2)_{n+\frac{1}{2}}\right), \\ (y_2)_{n+1} &= (y_2)_n + h_n\left(a(y_1)_{n+\frac{1}{2}}(y_2)_{n+\frac{1}{2}} - b(y_2)_{n+\frac{1}{2}}\right). \end{aligned}$$

Figure 7.3 presents results of simulations with parameters $a = 0.0001,\ b = 0.1$, using the modified Euler's method with four different fixed step-sizes h, starting from the initial point $[(y_1)_0, (y_2)_0]^{\mathrm{T}} = [5000, 10]^{\mathrm{T}}$. It can be easily seen that

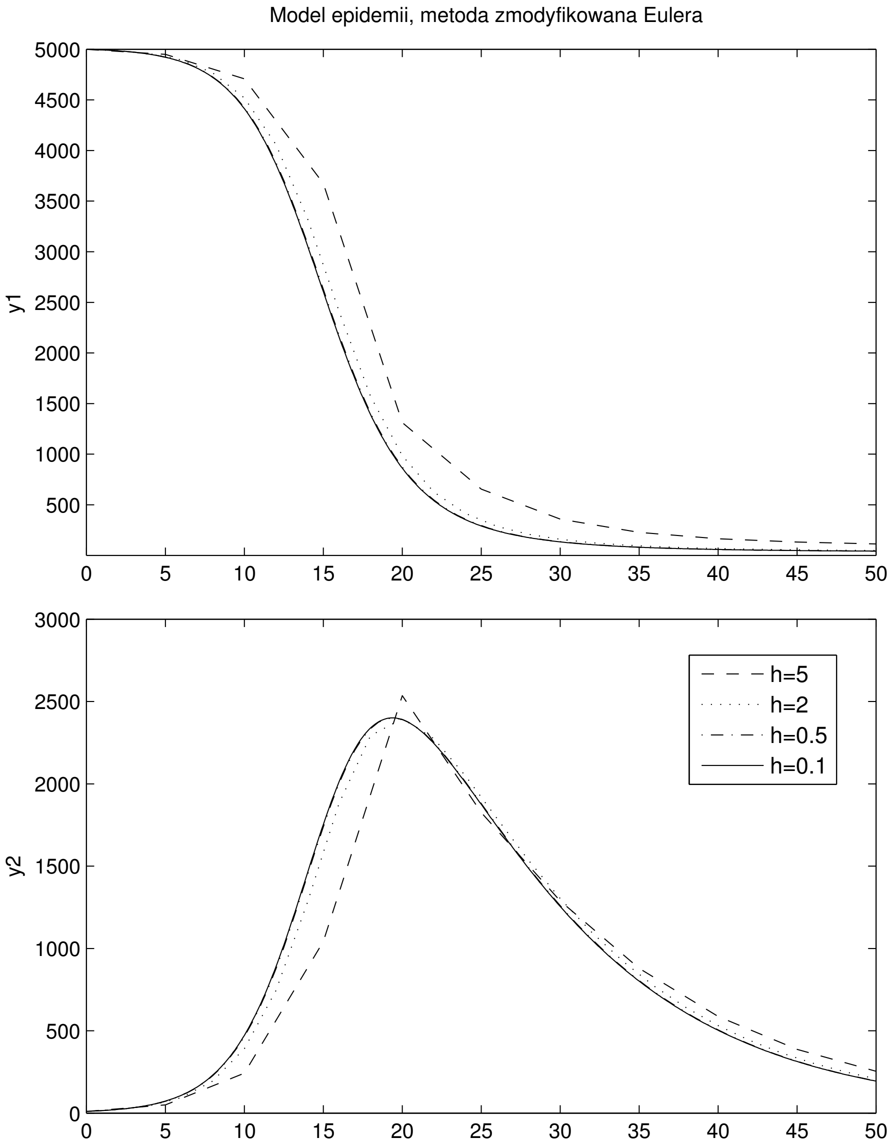


Figure 7.3. Simulations of the epidemic model using the modified Euler's method, with different fixed step-sizes

trajectories for larger h are not accurate, then they stabilize – for $h = 0.5$ and $h = 0.1$ the trajectories are practically indistinguishable. We left to the reader to check, which are the step-sizes leading to comparable results when using the standard Euler's method. □

In the following text in this chapter, *we shall not explicitly use bold typesetting for the vectors*, although all formulae will apply both to a single differential equation ($m = 1$) and to a system of differential equations ($m > 1$).

7.1. Single-step methods

The single-step methods are defined by the following general formula (for a fixed step-size h):

$$y_{n+1} = y_n + h\,\Phi_f\left(x_n, y_n; h\right), \tag{7.13}$$

where

$$x_n = x_0 + nh, \;\; n = 0, 1, \dots \qquad y(x_0) = y_0 = y_a \text{ (given)},$$

i.e., the function $\Phi_f\left(x_n, y_n; h\right)$ defines the method. Assuming $h \neq 0$, we can rewrite the definition formula (7.13) as

$$\Phi_f\left(x_n, y_n; h\right) = \frac{y_{n+1} - y_n}{h}.$$

Define

$$\Delta_f\left(x_n, y_n; h\right) = \frac{y\left(x_n + h\right) - y\left(x_n\right)}{h}.$$

The method is *convergent*, if

$$h \to 0 \qquad \Rightarrow \quad y\left(x_n; h\right) \to y\left(x\right),$$

i.e., when

$$h \to 0 \qquad \Rightarrow \quad y_n \to y(x_n), \;\; y_{n+1} \to y(x_n + h),$$

which, in turn, implies

$$\Phi_f\left(x_n, y_n; h\right) \underset{h\to 0}{\longrightarrow} \Delta_f\left(x_n, y_n; h\right).$$

On the other hand, we have

$$\Delta_f\left(x_n, y_n; h\right) \underset{h\to 0}{\longrightarrow} y'(x_n) = f\left(x_n, y_n\right).$$

Therefore, the condition

$$\Phi_f\left(x, y; 0\right) = f\left(x, y\right) \tag{7.14}$$

is called the *approximation condition*. If the assumptions of Theorem 7.1 are fulfilled and $\Phi_f\left(x, \cdot; h\right)$ satisfies the Lipschitz condition, then (7.14) is the *necessary and sufficient condition for the convergence of a single-step method.*

The approximation error (*the local, single-step error*) is defined as the error generated in one step of the method, i.e., under the assumption that the method starts this step with an accurate solution value, $y_n = y(x_n)$:

$$r_n(h) \overset{\mathrm{df}}{=} y(x_n + h) - [y(x_n) + h\Phi_f(x_n, y_n; h)], \tag{7.15}$$

where $y(x_n + h)$ is an accurate solution for $x = x_n + h$, to the problem

$$\begin{aligned} y'(x) &= f(x, y(x)), \\ y(x_n) &= y_n, \qquad x \in [x_n, b]. \end{aligned} \tag{7.16}$$

Assuming $r_n(h)$ is appropriately differentiable and taking its Taylor series expansion at $x = x_n$ (i.e., at $h = 0$), we have

$$r_n(h) = r_n(0) + r_n'(0)\,h + \frac{1}{2}r_n''(0)\,h^2 + \ldots$$

The method is said *to be of order p*, if

$$r_n(0) = 0,\ \ r_n'(0) = 0,\ \ \ldots,\ \ r_n^{(p)}(0) = 0,\ \ r_n^{(p+1)}(0) \neq 0. \tag{7.17}$$

Then

$$r_n(h) = \frac{r_n^{(p+1)}(0)}{(p+1)!}h^{p+1} + O\left(h^{p+2}\right), \tag{7.18}$$

where the first term is called *the main part of the approximation error*, and the second one denotes a function of order not lower than order of h^{p+2} (i.e., the function $\frac{O(h^{p+2})}{h^{p+2}}$ is constrained in a neighbourhood of zero).

Example 7.3. For the Euler's method:

$$r_n(h) = y(x_n + h) - y_n - hf(x_n, y_n).$$

Expanding $y(x_n + h)$ into the Taylor series we have

$$y(x_n + h) = y(x_n) + y'(x_n)\,h + \frac{1}{2}y''(x_n)\,h^2 + \ldots$$

From the definitions: $y_n = y(x_n)$, $y'(x_n) = f(x_n, y_n)$, thus

$$y(x_n + h) = \underbrace{y_n + hf(x_n, y_n)}_{y_{n+1}} + y''(x_n)\,h^2 + \ldots$$

A direct comparison yields the result

$$r_n(h) = y_n''(x_n)\,h^2 + O\left(h^3\right),$$

indicating that the Euler's method is of order one. □

7.1.1. Runge-Kutta (RK) methods

A family of Runge-Kutta methods can be defined by the following formula:

$$y_{n+1} = y_n + h \cdot \sum_{i=1}^{m} w_i k_i, \tag{7.19a}$$

where

$$\begin{aligned} k_1 &= f\left(x_n, y_n\right), & \text{(7.19b)} \\ k_i &= f(x_n + c_i h, y_n + h \cdot \sum_{j=1}^{i-1} a_{ij} k_j), \quad i = 2, 3, ..., m, & \text{(7.19c)} \end{aligned}$$

and also

$$\sum_{j=1}^{i-1} a_{ij} = c_i, \quad i = 2, 3, ..., m.$$

To perform a single step of the method the values of the right-hand sides of the equations must be calculated precisely *m times* – therefore, the method can be described as an *m-stage one*. The parameters $w_i,\ a_{ij},\ c_i$ are not unique, several methods with the same value m can be defined. Usually, the coefficients are chosen in a way assuring a high order of the method, for a given m. If $p(m)$ denotes a maximal possible order of the RK method, then it can be shown that

$$\begin{aligned} p\left(m\right) &= m & &\text{for } m = 1, 2, 3, 4, \\ p\left(m\right) &= m - 1 & &\text{for } m = 5, 6, 7, \\ p\left(m\right) &\leq m - 2 & &\text{for } m \geq 8. \end{aligned}$$

The methods with $m = 4$ and of order $p = 4$ are most important in practice – they constitute a good tradeoff between the approximation accuracy (given by the order of a method) and the number of arithmetic operations performed at one iteration, defining both a numerical burden and an influence of numerical errors.

The RK method of order 4 (RK4, ”classical”):

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6} h \left(k_1 + 2k_2 + 2k_3 + k_4\right), & \text{(7.20a)} \\ k_1 &= f\left(x_n, y_n\right), & \text{(7.20b)} \\ k_2 &= f(x_n + \frac{1}{2} h, y_n + \frac{1}{2} h k_1), & \text{(7.20c)} \\ k_3 &= f(x_n + \frac{1}{2} h, y_n + \frac{1}{2} h k_2), & \text{(7.20d)} \\ k_4 &= f\left(x_n + h, y_n + h k_3\right). & \text{(7.20e)} \end{aligned}$$

A graphical interpretation of one iteration of the RK4 method is presented in Fig. 7.4, for the case of one equation ($m = 1$). The coefficient k_1 represents the derivative at (x_n, y_n) (illustrated by a line tangent to a solution passing through

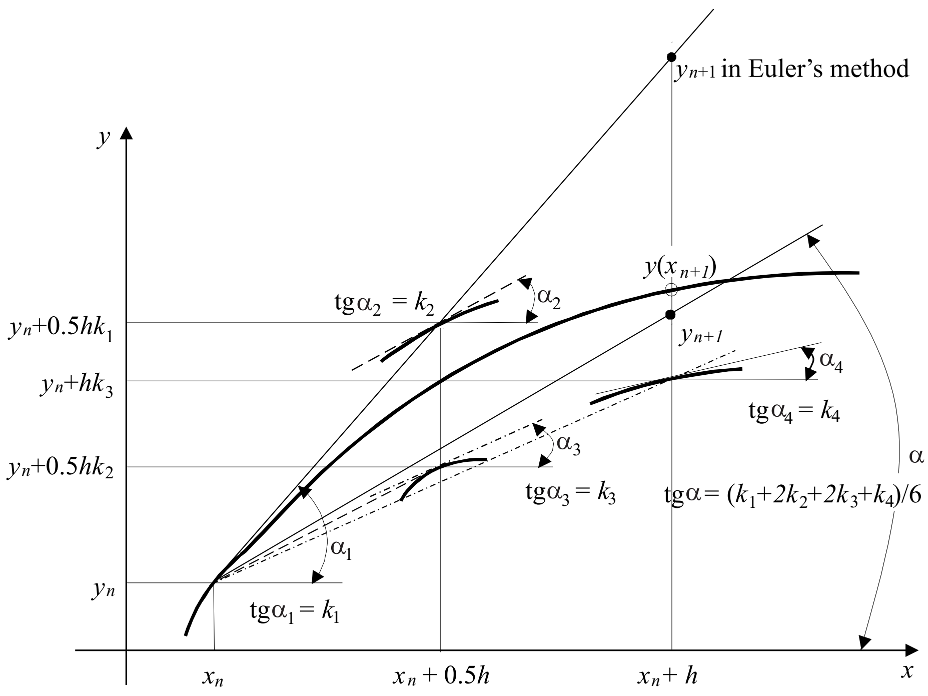


Figure 7.4. A graphical interpretation of one step of the RK4 algorithm

this point). The value k_2 is calculated as in the modified Euler's method – as a derivative of the solution calculated by the standard Euler's method at the midpoint $(x_n + 0.5h, y_n + 0.5hk_1)$, the dashed tangent line corresponds to this derivative. Next, the value k_3 is calculated similarly as it was for k_2, but this time at the point $(x_n + 0.5h, y_n + 0.5hk_2)$ – i.e., with a tangent line corresponding to k_2 (the dashed line). The derivative corresponding to k_3 is depicted in the figure by a dash-dotted line. Finally, we start with this line from the initial point until the endpoint $x_n + h$, i.e., the derivative k_4 of a solution at the point $(x_n + h, y_n + hk_3)$ is calculated. Therefore, we have four approximate values of the solution derivative over the one step interval: one at the initial point, two at the midpoint and one at the endpoint. The final approximation of the solution derivative for the final full step of the method is calculated as a weighted mean value of these derivatives, with the weight 1 for the initial and end points and the weight 2 for the midpoint – that is the explanation of the formula (7.20a).

Example 7.4. For the following set of two differential equations (7.12):

$$\begin{aligned} y_1{}' &= -ay_1y_2, \\ y_2{}' &= ay_1y_2 - by_2, \end{aligned}$$

the equations of one step of the RK4 method will be formulated, from the point x_n, with the solution at this point $y_n = [(y_1)_n, (y_2)_n]^{\mathrm{T}}$ (initial solution point for n-th step), with the current step-size h_n:

$$\begin{aligned}
k_{1,1} &= -a(y_1)_n \cdot (y_2)_n, \\
k_{1,2} &= a(y_1)_n \cdot (y_2)_n - b(y_2)_n, \\
k_{2,1} &= -a[(y_1)_n + \frac{1}{2}h_n k_{1,1}] \cdot [(y_2)_n + \frac{1}{2}h_n k_{1,2}], \\
k_{2,2} &= a[(y_1)_n + \frac{1}{2}h_n k_{1,1}] \cdot [(y_2)_n + \frac{1}{2}h_n k_{1,2}] - b[(y_2)_n + \frac{1}{2}h_n k_{1,2}], \\
k_{3,1} &= -a[(y_1)_n + \frac{1}{2}h_n k_{2,1}] \cdot [(y_2)_n + \frac{1}{2}h_n k_{2,2}], \\
k_{3,2} &= a[(y_1)_n + \frac{1}{2}h_n k_{2,1}] \cdot [(y_2)_n + \frac{1}{2}h_n k_{2,2}] - b[(y_2)_n + \frac{1}{2}h_n k_{2,2}], \\
k_{4,1} &= -a[(y_1)_n + h_n k_{3,1}] \cdot [(y_2)_n + h_n k_{3,2}], \\
k_{4,2} &= a[(y_1)_n + h_n k_{3,1}] \cdot [(y_2)_n + h_n k_{3,2}] - b[(y_2)_n + h_n k_{3,2}], \\
(y_1)_{n+1} &= (y_1)_n + \frac{1}{6}h_n(k_{1,1} + 2k_{2,1} + 2k_{3,1} + k_{4,1}), \\
(y_2)_{n+1} &= (y_2)_n + \frac{1}{6}h_n(k_{1,2} + 2k_{2,2} + 2k_{3,2} + k_{4,2}).
\end{aligned}$$

□

Step-size selection

A fundamental problem for a practical implementation of numerical methods for solving differential equations is an appropriate procedure for a selection / correction of the step-size h_n. There are two counteracting phenomena here:

– if the step h_n becomes smaller, then the approximation error of the method becomes smaller, but
– if the step h_n becomes smaller, then also the number of steps needed to find a solution on a given interval $[a, b]$ increases – and, therefore, the number of arithmetic calculations needed to find a solution also increases, together with numerical (roundoff) errors.

Therefore, calculations with too small steps are not recommended – the step should be sufficiently small to perform calculations with a desired accuracy, but not much smaller than necessary.

According to the above reasoning, a fixed step-size could be adequate only for problems with solutions varying at similar rate of change over the whole solution interval $[a, b]$. And even in this case, there is a problem how to find (or guess) an optimal step-size. Such a fixed step-size can be iteratively chosen by the user, but then the problem must be solved many times with different step-sizes, to find the best one by a comparison. Certainly, it would be better when an initial step-size given by the user (or set to a default value) was adjusted automatically by the method itself. But, for problems having solutions with a rate of change differing significantly over $[a, b]$, the step-size should be variable and adjusted automatically by the method itself, possibly at each step.

A fundamental information necessary to perform an automatic step-size adjustment is an estimate of the approximation error (local error of the method) occurring at each step of the method.

Error estimation (the step-doubling approach)

To estimate the approximation error, for every step of the size h two additional steps of the size $\frac{h}{2}$ are additionally performed in parallel (i.e. the first starting also from the point x_n). Denote:

$y_n^{(1)}$ – a new point obtained using the step-size h,
$y_n^{(2)}$ – a new point obtained using two consecutive steps of the size $\frac{h}{2}$.

Denoting by $r^{(1)}$ the approximation error after the single step h, and by $r^{(2)}$ the summed approximation errors after the two smaller steps (of length $0.5h$ each), the principle of step-doubling can be illustrated as shown in Fig. 7.5.

Assuming the same approximation error for each of the smaller steps of the size $\frac{h}{2}$ (a simplifying assumption), we have

$$y(x_n + h) = y_n^{(1)} + \underbrace{\frac{r_n^{(p+1)}(0)}{(p+1)!} \cdot h^{p+1}}_{\text{main part of the error}} + O(h^{p+2}) \ - \ \text{after a single step,}$$

$$y(x_n + h) \simeq y_n^{(2)} + \underbrace{2 \cdot \frac{r_n^{(p+1)}(0)}{(p+1)!} \left(\frac{h}{2}\right)^{p+1}}_{\text{main part of the error}} + O\left(h^{p+2}\right) \ - \ \text{after a double step.}$$

Evaluating the unknown coefficient $\gamma = \frac{r_n^{(p+1)}(0)}{(p+1)!}$ from the first equation and inserting it into the second one, we obtain

$$y\left(x_n + h\right) = y_n^{(2)} + \frac{h^{p+1}}{2^p} \frac{y\left(x_n + h\right) - y_n^{(1)}}{h^{p+1}} + O\left(h^{p+2}\right)$$

and after further manipulations

$$\begin{aligned} y\left(x_n + h\right)(1 - \frac{1}{2^p}) &= y_n^{(2)} - \frac{y_n^{(1)}}{2^p} + O\left(h^{p+2}\right) \\ &= y_n^{(2)}(1 - \frac{1}{2^p}) + \frac{y_n^{(2)}}{2^p} - \frac{y_n^{(1)}}{2^p} + O\left(h^{p+2}\right). \end{aligned}$$

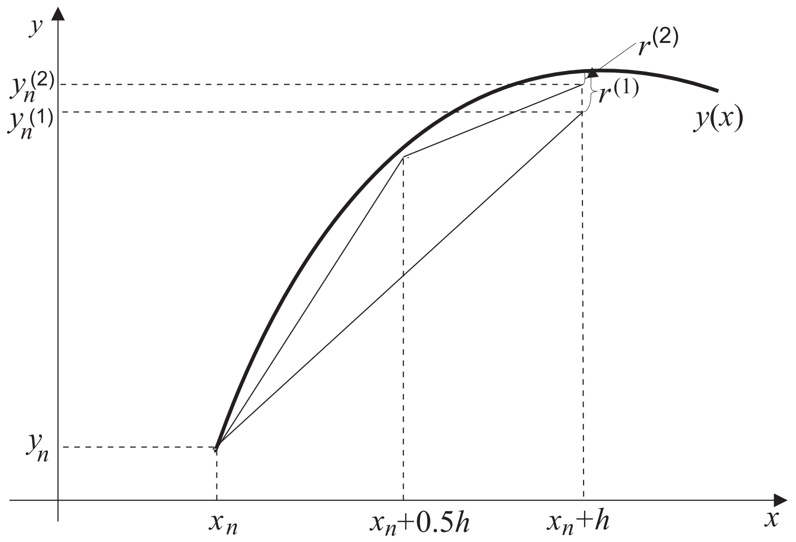


Figure 7.5. A graphical interpretation of the step-doubling approach

Multiplying both sides by $\frac{2^p}{2^p-1}$, we get

$$y\left(x_n+h\right)=y_n^{(2)}\ +\ \frac{y_n^{(2)}-y_n^{(1)}}{2^p-1}\ +\ O\left(h^{p+2}\right). \tag{7.21}$$

Making similar algebraic manipulations, we obtain also

$$y\left(x_n+h\right)=y_n^{(1)}\ +\ 2^p\frac{y_n^{(2)}-y_n^{(1)}}{2^p-1}\ +\ O\left(h^{p+2}\right). \tag{7.22}$$

The error estimate for a single step with the step-size h follows from (7.22), assuming the main part of the error is taken as the error estimate:

$$\delta_n(h)=\frac{2^p}{2^p-1}(y_n^{(2)}-y_n^{(1)}). \tag{7.23}$$

But, it follows from (7.21) that the expression

$$\delta_n\left(2\times\frac{h}{2}\right)=\frac{y_n^{(2)}-y_n^{(1)}}{2^p-1} \tag{7.24}$$

can be treated as an estimate of *the error of two consecutive steps of the size* $h/2$, the estimate being 2^p times smaller than $\delta_n(h)$. Therefore, taking $y_{n+1}=y_n^{(1)}$ with the error estimate $\delta_n\left(h\right)$ ($\approx y_n^{(2)}-y_n^{(1)}$) is correct but leads to a loss of accuracy. More practical is to use $y_{n+1}=y_n^{(2)}$ with the error estimate $\delta_n\left(2\times\frac{h}{2}\right)$ given by (7.24) – and, perhaps with a slightly less safety coefficient, see Section 7.1.3.

7.1.2. Runge-Kutta-Fehlberg (RKF) methods

Consider two RK methods, a m-stage one of the order p and a $(m+1)$-stage one of the order $p+1$:

– the RK method of the order p:

$$\begin{aligned} & y_{n+1} = y_n + h\cdot\sum_{i=1}^{m} w_i^* k_i, \\ & k_1 = f\left(x_n, y_n\right), \\ & k_i = f(x_n + c_i h,\ y_n + h\cdot\sum_{j=1}^{i-1} a_{ij}k_j), \qquad i = 2,3,...,m, \end{aligned} \tag{7.25}$$

– the RK method of order $p+1$:

$$\begin{aligned} & y_{n+1} = y_n + h\cdot\sum_{i=1}^{m+1} w_i k_i, \\ & k_1 = f\left(x_n, y_n\right), \\ & k_i = f(x_n + c_i h,\ y_n + h\cdot\sum_{j=1}^{i-1} a_{ij}k_j), \qquad i = 2,3,...,m+1. \end{aligned} \tag{7.26}$$

In both methods, the coefficients w_i^* and w_i are different, but the coefficients c_i and a_{ij} are the same for $j = 1,...,i-1,\ i = 2,...,m$, i.e. the coefficients k_i are the same for $i = 1,...,m$. If we perform calculations using both methods in parallel, then we get:

– for the method of the order p:

$$y\left(x_n + h\right) = \underbrace{y_n + h\cdot\sum_{i=1}^{m} w_i^* k_i\left(h\right)}_{y_{n+1}^{(0)}} + \underbrace{\frac{r_n^{(p+1)}(0)}{(p+1)!}h^{p+1}}_{\text{main part of the error}} + O\left(h^{p+2}\right), \tag{7.27}$$

– for the method of the order $p+1$:

$$y\left(x_n + h\right) = \underbrace{y_n + h\cdot\sum_{i=1}^{m+1} w_i k_i\left(h\right)}_{y_{n+1}^{(1)}} + \underbrace{\frac{r_n^{(p+2)}(0)}{(p+2)!}h^{p+2} + O\left(h^{p+3}\right)}_{O(h^{p+2})}. \tag{7.28}$$

Subtracting one equation from the other and removing the terms $O\left(h^{p+2}\right)$, the following error estimate (for the method of the order p) is obtained:

$$\delta_n\left(h\right) = \frac{r_n^{(p+1)}(0)}{(p+1)!}h^{p+1} = h\cdot\sum_{i=1}^{m}\left(w_i - w_i^*\right)\cdot k_i\left(h\right) + hw_{m+1}k_{m+1}\left(h\right). \tag{7.29}$$

The considered pair of the RK methods is called *a pair of embedded methods*. A convenient presentation of coefficients of a m-stage RK method is in a compact table form, as shown in Table 7.1. Using this format, the coefficients of a pair of embedded methods can be conveniently presented in one table only, as shown in Table 7.2 for a m-stage method and a $(m+1)$-stage one.

Table 7.1. A presentation of parameters of a m-stage RK method

c_i	a_{ij}				w_i^*
0					w_1^*
c_2	a_{21}				w_2^*
c_3	a_{31}	a_{32}			w_3^*
$\vdots$	$\vdots$	$\vdots$			$\vdots$
c_m	a_{m1}	a_{m2}	$\cdots$	$a_{m,m-1}$	w_m^*

Table 7.2. A presentation of a pair of embedded methods, a m- and a $(m+1)$-stage one

c_i	a_{ij}				w_i^*	w_i
0					w_1^*	w_1
c_2	a_{21}				w_2^*	w_2
$\vdots$	$\vdots$				$\vdots$	$\vdots$
c_m	a_{m1}		$a_{m,m-1}$		w_m^*	w_m
c_{m+1}	$a_{m+1,1}$	$\cdots$	$a_{m+1,m-1}$	$a_{m+1,m}$		w_{m+1}

The Runge-Kutta-Fehlberg (RKF) methods are the popular, efficient embedded RK methods. Parameters of the embedded RKF methods of orders 1 and 2 (RKF12), orders 2 and 3 (RKF23) and of orders 4 and 5 (RKF45) are given in Tables 7.3, 7.4 and 7.5, respectively.

To complete a single step of the RKF45 method, values of the functions of the right-hand side of the differential equations must be calculated six times (together with the error estimation). On the other hand, to complete a single step of the RK4 method using the step-doubling rule and with the error estimation performed according to (7.21), values of the functions of the right-hand side of the differential equations must be calculated 11 times (4+3+4) – but this is in fact a method

Table 7.3. Parameters of the RKF12 embedded methods (of order 1 and 2)

c_i	a_{ij}		w_i^*	w_i
0			$\frac{1}{256}$	$\frac{1}{512}$
$\frac{1}{2}$	$\frac{1}{2}$		$\frac{255}{256}$	$\frac{255}{256}$
1	$\frac{1}{256}$	$\frac{255}{256}$		$\frac{1}{512}$

Table 7.4. Parameters of the RKF23 embedded methods (of order 2 and 3)

c_i	a_{ij}			w_i^*	w_i
0				$\frac{214}{891}$	$\frac{533}{2106}$
$\frac{1}{4}$	$\frac{1}{4}$			$\frac{1}{33}$	0
$\frac{27}{40}$	$-\frac{189}{800}$	$\frac{729}{800}$		$\frac{650}{891}$	$\frac{800}{1053}$
1	$\frac{214}{891}$	$\frac{1}{33}$	$\frac{650}{891}$		$-\frac{1}{78}$

Table 7.5. Parameters of the RKF45 embedded methods (of order 4 and 5)

c_i	a_{ij}					w_i^*	w_i
0						$\frac{25}{216}$	$\frac{16}{135}$
$\frac{1}{4}$	$\frac{1}{4}$					0	0
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				$\frac{1408}{2565}$	$\frac{6656}{12825}$
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			$\frac{2197}{4104}$	$\frac{28561}{56430}$
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		$-\frac{1}{5}$	$-\frac{9}{50}$
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$		$\frac{2}{55}$

operating with the step-size $h/2$. Therefore, when comparing these methods, the computations performed for two steps of each method should be taken into account. Thus, for the successful steps, i.e., satisfying the accuracy test ($s\alpha \geq 1$), the amount of computations is comparable, even slightly smaller for the RK4 method (11 versus 12). On the other hand, for unsuccessful steps, when the step-size must be reduced and the computations repeated, the RKF methods are superior, needing significantly less computations than the RK methods of the same order (in the RKF method only one unsuccessful step is repeated, but in the RK method with error estimation based on the step-doubling rule, two steps are in fact repeated).

7.1.3. Correction of the step-size

A general formula for the main part of the approximation error, after a step of a length h, is

$$\delta_n(h) = \gamma \cdot h^{p+1}, \quad \text{where} \quad \gamma = \frac{r_n^{(p+1)}(0)}{(p+1)!} \tag{7.30}$$

is the first non-zero coefficient in a Taylor series expansion of the approximation error. If the step-size is changed to αh, then

$$\delta_n(\alpha h) = \gamma \cdot (\alpha h)^{p+1},$$

therefore

$$\delta_n(\alpha h) = \alpha^{p+1} \cdot \delta_n(h). \tag{7.31}$$

Assuming a tolerance ε:

$$|\delta_n(\alpha h)| = \varepsilon, \tag{7.32}$$

we get

$$\alpha^{p+1}|\delta_n(h)| = \varepsilon.$$

Therefore, the coefficient α for the step-size correction can be evaluated as

$$\alpha = (\frac{\varepsilon}{|\delta_n(h)|})^{\frac{1}{p+1}}. \tag{7.33}$$

This formula is obviously correct for any method with an error estimate $\delta_n(h)$. It is also true for the RK4 method with error estimation according to (7.24), i.e., when $\delta_n(h) = \delta_n\left(2 \times \frac{h}{2}\right)$. This can be easily proved in a formal way:

$$\delta_n\left(2 \times \frac{h}{2}\right) \approx 2\gamma(\frac{h}{2})^{p+1},$$
$$\delta_n\left(2 \times \frac{\alpha h}{2}\right) \approx 2\gamma(\frac{\alpha h}{2})^{p+1} = \alpha^{p+1} \cdot 2\gamma(\frac{h}{2})^{p+1} = \alpha^{p+1} \cdot \delta_n\left(2 \times \frac{h}{2}\right).$$

i.e., the obtained formula is analogous to (7.31).

In practice, a lack of accuracy in the error estimation should also be taken into account, this is achieved by by the use of a safety factor s, as follows:

$$h_{n+1} = s \cdot \alpha \cdot h_n, \quad \text{where } s < 1 \quad (\text{for RK4, RKF45: } s \approx 0.9). \tag{7.34}$$

The accuracy parameters (usually user defined values) should preferably be defined in the following way:

$$\varepsilon = |y_n| \cdot \varepsilon_r + \varepsilon_a, \tag{7.35}$$

where

ε_r – a relative tolerance,

ε_a – an absolute tolerance.

For a set of m differential equations, a worst case approach must be used, i.e. the equation which requires the highest accuracy (thus the smallest step-size) determines the step-size, e.g., for the RK methods with error estimation based on the step-doubling approach we have

$$\delta_n(h)_i = \frac{(y_i)_n^{(2)} - (y_i)_n^{(1)}}{2^p - 1}, \qquad i = 1, 2, ..., m, \tag{7.36}$$

$$\varepsilon_i = \left|(y_i)_n^{(2)}\right| \cdot \varepsilon_r + \varepsilon_a, \tag{7.37}$$

$$\alpha = \min_{1\le i\le m} \left(\frac{\varepsilon_i}{\left|\delta_n(h)_i\right|}\right)^{\frac{1}{p+1}}. \tag{7.38}$$

A block diagram of a general implementation structure of RK (or RKF) algorithms is presented in Fig. 7.6.

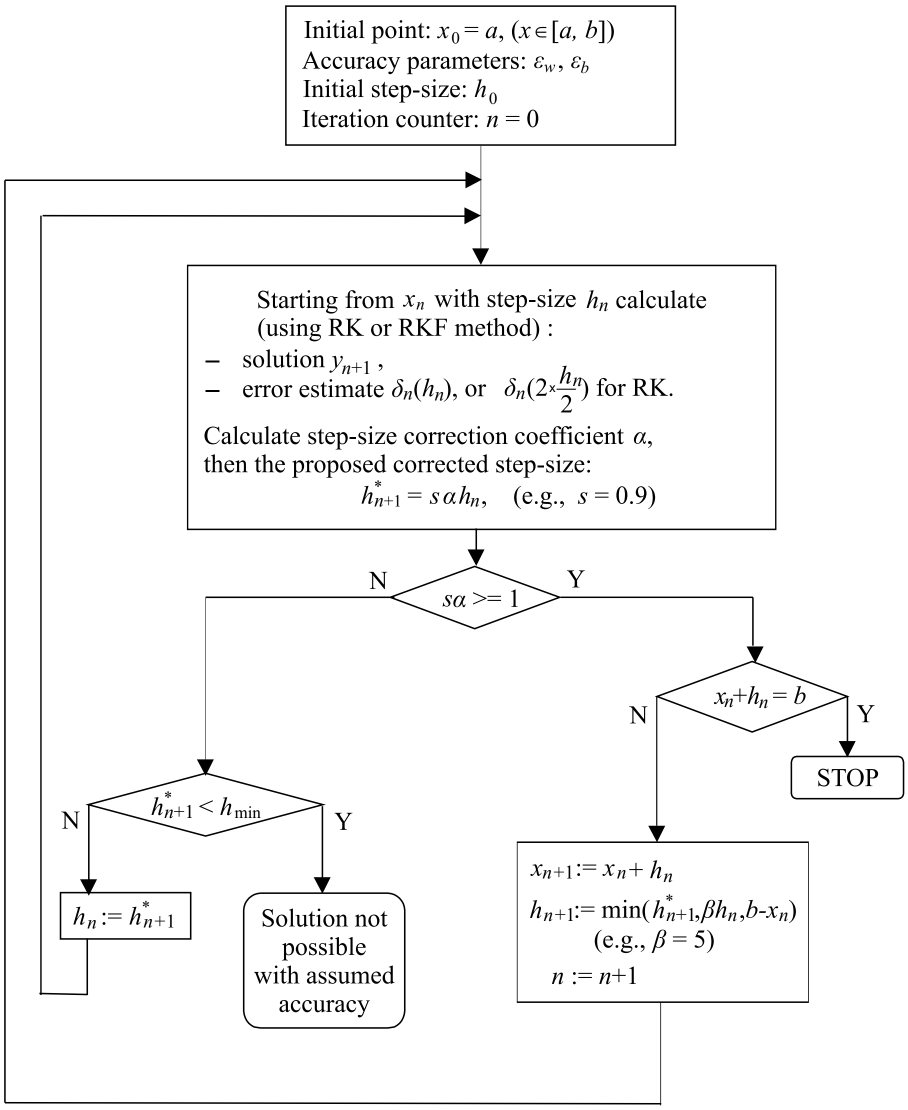

Initial point: $x_0 = a$, ($x \in [a, b]$)
Accuracy parameters: ε_w, ε_b
Initial step-size: h_0
Iteration counter: $n = 0$
Starting from x_n with step-size h_n calculate (using RK or RKF method) :
– solution y_{n+1} ,
– error estimate $\delta_n(h_n)$, or $\delta_n(2\times\frac{h_n}{2})$ for RK.
Calculate step-size correction coefficient α, then the proposed corrected step-size:
$h^*_{n+1} = s\alpha h_n$, (e.g., $s = 0.9$)
N
$s\alpha >= 1$
Y
$x_n+h_n = b$
N
Y
STOP
$h^*_{n+1} < h_{\min}$
N
Y
$h_n := h^*_{n+1}$
Solution not possible with assumed accuracy
$x_{n+1} := x_n + h_n$
$h_{n+1} := \min(h^*_{n+1}, \beta h_n, b-x_n)$
(e.g., $\beta = 5$)
$n := n+1$

Figure 7.6. Flow diagram of an implementation structure of the RK/RKF methods

7.2. Multistep methods

A single step (single iteration) of a *linear multistep (k-step) method* with a constant step-size h can be defined by the following formula:

$$y_n = \sum_{j=1}^{k} \alpha_j \cdot y_{n-j} + h \sum_{j=0}^{k} \beta_j \cdot f\left(x_{n-j}, y_{n-j}\right),$$
$$y_0 = y(x_0) = y_a,\ x_n = x_0 + nh,\ x \in [a = x_0, b]. \tag{7.39}$$

A multistep method is *explicit* if $\beta_0 = 0$. In this case, the value y_n depends *explicitly* on values of the solution y and and its derivative $f(x, y)$ taken at previously calculated points only, i.e., y_n depends only on $y_{n-1},\ y_{n-2},\ \dots,\ y_{n-k},$ and $f_{n-1} = f\left(x_{n-1}, y_{n-1}\right), f_{n-2} = f\left(x_{n-2}, y_{n-2}\right), \dots,\ f_{n-k} = f\left(x_{n-k}, y_{n-k}\right).$

A multistep method is *implicit* if $\beta_0 \neq 0$. In this case, the value y_n depends not only on previously calculated values $y_{n-1}, \dots,\ y_{n-k}$ and $f_{n-1}, \dots,\ f_{n-k},$ where $f_j = f\left(x_j, y_j\right)$, but also on $f_n = f(x_n, y_n)$. Therefore, in order to calculate y_n, it is necessary to solve an algebraic (and nonlinear, if f is nonlinear) equation $\varphi\left(y_n\right) = 0$, where

$$\varphi\left(y_n\right) \overset{\text{df}}{=} -y_n + \sum_{j=1}^{k} \alpha_j y_{n-j} + h \cdot \sum_{j=1}^{k} \beta_j f\left(x_{n-j}, y_{n-j}\right) + h\beta_0 f\left(x_n, y_n\right). \tag{7.40}$$

It can be shown that, under the assumptions of Theorem 7.1, this equation has a unique solution for sufficiently small h, precisely for

$$h < \left(L\beta_0\right)^{-1},$$

where L is a Lipschitz constant of f(with respect to y, see Theorem 7.1).

The equation (7.39), describing the algorithm of a multistep method, is a *difference equation*. For a first use of this equation on the interval $[a, b]$, first k values of the solution, $y_0, \dots, y_{k-1}$, are necessary – which cannot be evaluated using a k-step method. Therefore, a special *starting procedure* is needed to evaluate these values. Application of an appropriate (having the same accuracy) RK algorithm can be a solution.

7.2.1. Adams methods

An initial value problem

$$y'(x) = f\left(x, y\left(x\right)\right),$$
$$y\left(a\right) = y_a, \quad x \in [a, b],$$

can be equivalently formulated in the form of an integral equation

$$y(x) = y(a) + \int_a^x f(t, y(t))\, dt.$$

Considering this integral on the interval $[x_{n-1}, x_n]$,

$$y(x_n) = y(x_{n-1}) + \int_{x_{n-1}}^{x_n} f(t, y(t))\, dt, \tag{7.41}$$

leads to Adams methods.

Explicit Adams methods (Adams-Bashforth methods)

The function $f(x, y(x))$ in (7.41) is replaced by an interpolation polynomial $W(x)$ of order $k-1$, calculated at the points $x_{n-1}, ..., x_{n-k}$ with the corresponding function values $y(x_{n-j}) \approx y_{n-j}$. Using the Lagrange interpolation formula (see Section 5.1.1) we have

$$f(x, y(x)) \approx W(x) = \sum_{j=1}^{k} f(x_{n-j}, y_{n-j}) \cdot L_j(x), \tag{7.42}$$

$$y_n = y_{n-1} + \sum_{j=1}^{k} f(x_{n-j}, y_{n-j}) \cdot \int_{x_{n-1}}^{x_n} L_j(t)\, dt,$$

where $L_j(x)$ are the Lagrange polynomials,

$$L_j(x) = \prod_{m=1, m\neq j}^{k} \frac{x - x_{n-m}}{x_{n-j} - x_{n-m}}.$$

Assuming that the points are equally spaced, $x_{n-j} = x_n - jh$, $j = 1, 2, ..., k$, the integration process yields

$$y_n = y_{n-1} + h \sum_{j=1}^{k} \beta_j f(x_{n-j}, y_{n-j}), \tag{7.43}$$

where values of the coefficients β_j are given in Table 7.6, for $k =$ 1,...,7.

Table 7.6. Parameters of the explicit Adams methods (Adams-Bashforth methods)

k	β_1	β_2	β_3	β_4	β_5	β_6	β_7
1	1						
2	$\frac{3}{2}$	$-\frac{1}{2}$					
3	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$				
4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$			
5	$\frac{1901}{720}$	$-\frac{2774}{720}$	$\frac{2616}{720}$	$-\frac{1274}{720}$	$\frac{251}{720}$		
6	$\frac{4277}{1440}$	$-\frac{7923}{1440}$	$\frac{9982}{1440}$	$-\frac{7298}{1440}$	$\frac{2877}{1440}$	$-\frac{475}{1440}$	
7	$\frac{198721}{60480}$	$-\frac{447288}{60480}$	$\frac{705549}{60480}$	$-\frac{688256}{60480}$	$\frac{407139}{60480}$	$-\frac{134472}{60480}$	$\frac{19087}{60480}$

Implicit Adams methods (Adams-Moulton methods)

The function $f\left(x, y\left(x\right)\right)$ in (7.41) is now replaced by an interpolation polynomial $W^*\left(x\right)$ of order k calculated at the points $x_n, x_{n-1}, ..., x_{n-k}$ with the corresponding solution values $y\left(x_{n-j}\right) \approx y_{n-j}$. Reasoning in the same way as it was done in the case of the explicit methods, we finally get

$$\begin{aligned} y_n &= y_{n-1} + h\sum_{j=0}^{k} \beta_j^* \cdot f\left(x_{n-j}, y_{n-j}\right) \\ &= y_{n-1} + h \cdot \beta_0^* \cdot f\left(x_n, y_n\right) + h\sum_{j=1}^{k} \beta_j^* \cdot f\left(x_{n-j}, y_{n-j}\right), \qquad (7.44) \end{aligned}$$

where values of the parameters β_j^*, for k = 1, ..., 7, are given in Table 7.7, with $k = 1^+$ denoting the backward Euler's method (see Example 7.7 in Section 7.3).

7.2.2. The approximation error

A *(local) approximation error* at a point x_n is defined as

$$r_n\left(h\right) \overset{\text{df}}{=} [\sum_{j=1}^{k} \alpha_j y\left(x_{n-j}\right) + h\sum_{j=0}^{k} \beta_j f\left(x_{n-j}, y(x_{n-j})\right)] - y\left(x_n\right), \qquad (7.45)$$

i.e., $r_n\left(h\right)$ is the error generated in a single (n-th) step only, i.e., when the method starts from an accurate solution at the point x_{n-1} (in the square brackets, all solution points are accurate, $y_{n-j} = y(x_{n-j}),\ j = 1, \ldots, k$) and generates a new

Table 7.7. Parameters of the implicit Adams methods (Adams-Moulton methods)

k	β_0^*	β_1^*	β_2^*	β_3^*	β_4^*	β_5^*	β_6^*	β_7^*
1^+	1							
1	$\frac{1}{2}$	$\frac{1}{2}$						
2	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$					
3	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$				
4	$\frac{251}{720}$	$\frac{646}{720}$	$-\frac{264}{720}$	$\frac{106}{720}$	$-\frac{19}{720}$			
5	$\frac{475}{1440}$	$\frac{1427}{1440}$	$-\frac{798}{1440}$	$\frac{482}{1440}$	$-\frac{173}{1440}$	$\frac{27}{1440}$		
6	$\frac{19087}{60480}$	$\frac{65112}{60480}$	$-\frac{46461}{60480}$	$\frac{37504}{60480}$	$-\frac{20211}{60480}$	$\frac{6312}{60480}$	$-\frac{863}{60480}$	
7	$\frac{36799}{120960}$	$\frac{139849}{120960}$	$-\frac{121797}{120960}$	$\frac{123133}{120960}$	$-\frac{88547}{120960}$	$\frac{41499}{120960}$	$-\frac{11351}{120960}$	$\frac{1375}{120960}$

approximate solution point y_n at $x_n = x_{n-1} + h$. Note that this definition is the same as in the case of the single-step methods, see (7.15) in Section 7.1.

Remark*. The equations of the multistep method and of the approximation error yield together:

$$y_n - y(x_n) = h\beta_0[f(x_n, y_n) - f(x_n, y(x_n))] + r_n(h).$$

Applying now the mean-value theorem we obtain

$$r_n(h) = (1 - h\beta_0 \frac{\partial f}{\partial y}(x_n, \zeta_n))(y_n - y(x_n)), \tag{7.46}$$

where $\zeta_n \in (y_n, y(x_n))$. Thus, for the explicit methods the approximation error is equal to the difference $y_n - y(x_n)$, and for the implicit methods it is proportional to this difference, whereas the smaller the step-size h the closer the proportionality coefficient to 1.

□

Assuming $x_{n-j} = x_n - jh, \quad j = 0, 1, ..., k$ (a constant step-size), we have

$$r_n(h) = \sum_{j=0}^{k} \alpha_j y(x_n - jh) + h \sum_{j=0}^{k} \beta_j y'(x_n - jh), \tag{7.47}$$

where $\alpha_0 \overset{\text{df}}{=} -1$.

*Optional.

Expanding now $y\,(x_n - jh)$ and $y'(x_n - jh)$ in a Taylor's series at the point x_n (assuming $y\,(x)$ is $p+1$ times differentiable, $p \geq 1$) and rearranging the terms appropriately, we get the result

$$r_n\,(h) = c_0 y\,(x_n) + \sum_{m=1}^{p+1} h^m c_m\, y^{(m)}\,(x_n) + O\left(h^{p+2}\right), \qquad (7.48)$$

where

$$c_0 = \sum_{j=0}^{k} \alpha_j = -1 + \sum_{j=1}^{k} \alpha_j,$$

$$c_1 = -\sum_{j=1}^{k} j\alpha_j + \sum_{j=0}^{k} \beta_j,$$

$$c_m = \frac{1}{m!} \sum_{j=1}^{k} (-j)^m\, \alpha_j + \frac{1}{(m-1)!} \sum_{j=1}^{k} (-j)^{(m-1)}\, \beta_j, \qquad m \geq 2. \qquad (7.49)$$

Definition. *A multistep method is of order p* if the coefficients of the expansion formula (7.49) satisfy (structurally, for any system of differential equations):

$$c_0 = 0,\ c_1 = 0,\ \ldots,\ c_p = 0,\ c_{p+1} \neq 0. \qquad (7.50)$$

Therefore, for a method of the order p, the approximation error is expressed by the formula

$$r_n(h) = c_{p+1} h^{p+1} y^{(p+1)}\,(x_n) + O(h^{p+2}), \qquad (7.51)$$

where c_{p+1} is called the *error constant* and $c_{p+1} h^{p+1} y^{(p+1)}\,(x_n)$ is called the *main part of the approximation error*. The orders and error constants of the explicit and implicit Adams methods are given in Tables 7.8 and 7.9, respectively.

Table 7.8. Orders and error constants of the explicit Adams methods

k	1	2	3	4	5	6	7
p	1	2	3	4	5	6	7
c_{p+1}	$-\frac{1}{2}$	$-\frac{5}{12}$	$-\frac{3}{8}$	$-\frac{251}{720}$	$-\frac{95}{288}$	$-\frac{19087}{60480}$	$-\frac{36799}{120960}$

Table 7.9. Orders and error constants of the implicit Adams methods

k	1^+	1	2	3	4	5	6	7
p		2	3	4	5	6	7	8
c^*_{p+1}	$\frac{1}{2}$	$\frac{1}{12}$	$\frac{1}{24}$	$\frac{19}{720}$	$\frac{3}{360}$	$\frac{863}{60480}$	$\frac{275}{24192}$	$\frac{339533}{3628800}$

7.2.3. Stability and convergence

A multistep method is called *stable*, if its difference equation taken for $h = 0$:

$$-y_n + \sum_{j=1}^{k} \alpha_j y_{n-j} = -y_n + \alpha_1 y_{n-1} + \cdots + \alpha_{k-1} y_{n-k-1} + \alpha_k y_{n-k} = 0, \quad (7.52)$$

is stable. The stability condition for the equation (7.52) is defined by the roots of its *characteristic polynomial*:

$$\rho(z) = -z^k + \alpha_1 z^{k-1} + \ldots + \alpha_{k-1} z + \alpha_k. \quad (7.53)$$

The equation (7.52) is stable, if the polynomial (7.53) has all roots within or on the unit circle (the roots lying on the unit circle can be only single).

Theorem 7.2. *A multistep method is convergent, if and only if it is stable and at least of order* 1 *(i.e.,* $c_0 = c_1 = 0$ *– the consistency condition).*

Define also a polynomial corresponding to the coefficients β_j (i.e., in the equation of the method, the coefficients of the terms dependent on the derivatives of the solution – the values $f(x_{n-j}, y_{n-j})$):

$$\sigma(z) = \beta_0 z^k + \ldots + \beta_{k-1} z + \beta_k. \quad (7.54)$$

Choosing appropriately the coefficients α_j and β_j (for an assumed value of k), one can design a multistep method which satisfies not only the convergence conditions (which is necessary for any potentially useful method), but also maximizing its order or/and minimizing its error constant.

Example 7.5. Consider a general explicit 2-step method. We have

$$y_n = \alpha_1 y_{n-1} + \alpha_2 y_{n-2} + h\beta_1 f(x_{n-1}, y_{n-1}) + h\beta_2 f(x_{n-2}, y_{n-2}).$$

Therefore

$$\rho(z) = -z^2 + \alpha_1 z + \alpha_2,$$
$$\sigma(z) = \beta_1 z + \beta_2.$$

We have four coefficients, thus we can design a multistep method of order at least 2. The conditions of the second order (including the consistency conditions) are

$$\begin{aligned} c_0 = 0 &\Rightarrow\ -1 + \alpha_1 + \alpha_2 = 0, \\ c_1 = 0 &\Rightarrow\ -\alpha_1 - 2\alpha_2 + \beta_1 + \beta_2 = 0, \\ c_2 = 0 &\Rightarrow\ \frac{1}{2}(\alpha_1 + 4\alpha_2) - \beta_1 - 2\beta_2 = 0. \end{aligned}$$

These are three equations with four unknowns. Treating β_2 as a parameter we have three equations with three unknowns, yielding a well defined solution

$$\alpha_1 = -2\beta_2, \ \ \alpha_2 = 1 + 2\beta_2, \ \ \beta_1 = \beta_2 + 2.$$

This leads to the following form of the characteristic polynomial:

$$\rho(z) = -z^2 - 2\beta_2 z + 1 + 2\beta_2.$$

Note that $z_1 = 1$ is a root of $\rho(z)$ (for a convergent method, the value 1 is always the root of the characteristic polynomial, because: $\rho(1) = -1 + \sum_{i=1}^{n} \alpha_i = c_0 = 0$). This can be written as

$$\rho(z) = -(z-1)(z-z_2),$$

where $z_2 = -(1 - 2\beta_2)$. The stability conditions require

$$-1 \le z_2 < 1, \quad \text{which implies} \quad -1 < \beta_2 \le 0.$$

Any value of β_2 from this interval leads to the method of the order at least 2. Thus, we can use β_2 to a choice of the error constant, or to a choice of the root z_2, which has an influence on the stability properties of the method. It is easy to calculate that

$$c_3 = \frac{1}{6}\beta_2 - \frac{1}{3}.$$

For $\beta_2 = 0$ we get the method with the least value of c_3, $|c_3| = \frac{1}{3}$, (2-step Nystroem's method) – then $z_2 = -1$.
For $\beta_2 = -\frac{1}{2}$ we get $c_3 = -\frac{5}{12}$ and $z_2 = 0$, this is precisely the 2-step explicit Adams (Adams-Bashforth) method. □

For practical applications, stability of the difference equation of the method *with finite values of the step-size h* must be considered. However, the difference equation with $h \neq 0$ depends also on the problem considered, on the function $f(x, y)$ of the right-hand side of the differential equations describing the problem ($f(x, y)$ describes derivatives of the solution). The stability for a non-zero step-size can be analyzed for linear problems (i.e., with linear functions $f(x, y)$) having asymptotically stable solutions. For a nonlinear problem, this corresponds to the analysis of its linear approximation (linearization).

Consider a problem with a one-dimensional (scalar) linear differential equation

$$\begin{aligned} y'(x) &= \lambda \cdot y(x), \\ y(0) &= 1, \quad x \in [0, b]\,, \ b \gg 0, \end{aligned} \tag{7.55}$$

where $\lambda \in \mathbb{C}$ (a complex value is needed for a further generalization to a multidimensional case) and $\mathrm{Re}\lambda < 0$ due to the assumed asymptotic stability. Notice that the solution to this problem converges to zero as x increases from 0 to b. Applying a multistep method to this equation yields the following difference equation

$$y_n = \sum_{j=1}^{k} \alpha_j y_{n-j} + h \cdot \sum_{j=0}^{k} \beta_j \lambda y_{n-j}, \tag{7.56}$$

or, equivalently,

$$\sum_{j=0}^{k} (\alpha_j + h\lambda\beta_j)\, y_{n-j} = 0, \qquad \text{where } \alpha_0 = -1. \tag{7.57}$$

The equation (7.57) is asymptotically stable (has an asymptotically stable solution) if and only if all roots of its characteristic polynomial are within the unit circle, where the characteristic polynomial is

$$\widetilde{\rho}\,(z; h\lambda) = \rho\,(z) + h\lambda\sigma\,(z)\,, \tag{7.58}$$

where the polynomials $\rho\,(z)$ and $\sigma\,(z)$ are given by (7.53) and (7.54).

A multistep method with a given constant value of the step-size $h > 0$ is said to be *absolutely stable* for the problem (7.55) if the difference equation (7.57) is asymptotically stable, i.e., if it describes an asymptotically stable discrete dynamical system (a solution sequence $y_0 = 1, y_1, y_2,\ \ldots,\ y_n, y_{n+1}, \ldots$ converges to zero as $n \to \infty$). Note that not the individual values h and λ, but the product $h\lambda$ is a parameter of the characteristic polynomial (7.58). Therefore, we have the following definition:

Definition. A set of values $h\lambda$, where $\lambda \in \mathbb{C}$, $\mathrm{Re}\lambda < 0$, $h \in \mathbb{R}$, $h > 0$ (i.e., $h\lambda \in \mathbb{C}$, $\mathrm{Re}(h\lambda) < 0$), such that the roots of the polynomial (7.58) are located within the unit circle, is called the *set of the absolute stability* of the multistep method (defined by polynomials $\rho\,(z)$ and $\sigma\,(z)$), and the method is called *absolute stable* in this set.

Note that the roots of the characteristic equations may be complex, therefore the sets of the absolute stability are subsets of the complex plane. Moreover, we have $\mathrm{Re}\,(h\lambda) < 0$, because $\mathrm{Re}\,(\lambda) < 0$ and $h > 0$, therefore the sets of the absolute stability are located within the left half-plane. It can be shown that, if all roots of $\rho(z)$ except one (which is always equal to 1 for a convergent method, as shown in the

Example 7.5) have absolute values less than one, then for sufficiently small values $h\lambda$ the method is absolutely stable (the set of the absolute stability has non-empty interior).

The shape of sets of the absolute stability of the Adams methods is depicted in Fig. 7.7, where $[q, 0]$ is an *interval of the absolute stability*. The values of its left endpoint q for the Adams methods are given in the next section, in Table 7.10.

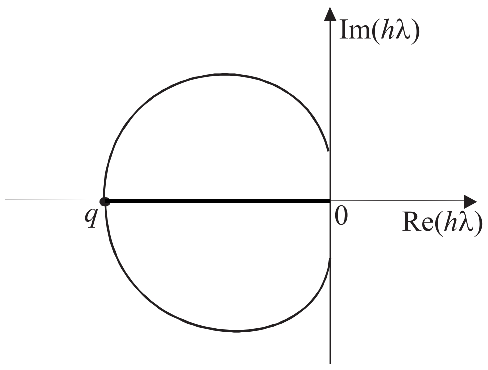


Figure 7.7. The shape of the absolute stability sets of the Adams methods

The definition of the absolute stability can easily be extended to the multidimensional linear case

$$\begin{aligned} y'(x) &= \mathbf{A}\, y(x)\,, \\ y\,(0) &= 1, \quad x \in [0, b]\,,\ b \gg 0, \end{aligned} \tag{7.59}$$

where all m eigenvalues λ_i of the matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$ have negative real parts, $\mathrm{Re}\lambda_i < 0$, because the system (7.59) is assumed to be asymptotically stable. For the absolute stability of a multistep method with a given value of the step-size h, applied to the system (7.59), *it is necessary and sufficient that for every eigenvalue λ_i, the product $h\lambda_i$ belongs to the set of the absolute stability of the method.*

7.2.4. Predictor-corrector methods

A multistep method with the following properties would be most useful:

1. a high order and a small error constant,
2. a large set of the absolute stability,
3. a small number of arithmetic operations performed during one iteration.

The explicit methods do fulfill the last property, but do not the first two. On the other hand, the implicit methods fulfill the first two properties better, but are much worse in fulfilling the last one, because at each iteration the nonlinear equation

$$-y_n + \sum_{j=1}^{k} \left(\alpha_j^* y_{n-j} + h\beta_j^* f_{n-j}\right) + h\beta_0^* f\left(x_n, y_n\right) = 0. \tag{7.60}$$

must be solved, with respect to y_n.

Practical implementations of the multistep methods are in a *predictor-corrector* (PC) structure, which combine the explicit and an the implicit methods into one algorithm. For the k-step methods, *the predictor-corrector method* P$_k$EC$_k$E, with one iteration of the corrector algorithm only, which solves the nonlinear equation (7.60) by the fix point iterative method, takes the following form:

$$\begin{array}{lll}
\text{P:} & y_n^{[0]} = \sum\limits_{j=1}^{k} \alpha_i y_{n-j} + h \sum\limits_{j=1}^{k} \beta_j f_{n-j}, & \text{(P – prediction)}\\
\text{E:} & f_n^{[0]} = f(x_n, y_n^{[0]}), & \text{(E – evaluation)}\\
\text{C:} & y_n = \sum\limits_{j=1}^{k} \alpha_j^* y_{n-j} + h \sum\limits_{j=1}^{k} \beta_j^* f_{n-j} + h\beta_0^* f_n^{[0]}, & \text{(C – correction)}\\
\text{E:} & f_n = f\left(x_n, y_n\right). & \text{(E – evaluation)}
\end{array}$$

Interpretation: The predictor iteration yields a good initial point $y_n^{[0]}$ for the corrector iterations solving the equation (7.60) by the fix point method (see Section 6.2.3). The predictor should provide a very good initial point – the smaller h and the higher the order of the method the better $y_n^{[0]}$. In the P$_k$EC$_k$E algorithm given above only one corrector iteration is performed. Let us emphasize: *the PC method is in fact an efficient way of an implementation of the implicit (corrector) method, the algorithm of the predictor (the explicit method) plays here only an auxiliary role to submit a good initial point.*

More precise results can be obtained if not one but $m > 1$ iterations of the fixed point corrector routine are performed – it is a P$_k$(EC$_k$)^{m}E structure:

$$\begin{array}{ll}
\text{P:} & y_n^{[0]} = \sum\limits_{j=1}^{k} \alpha_i y_{n-j} + h \sum\limits_{j=1}^{k} \beta_j f_{n-j}, \quad s = 0,\\
\text{E}_1\text{:} & f_n^{[s]} = f(x_n, y_n^{[s]}),\\
\text{C:} & y_n^{[s]} = \sum\limits_{j=1}^{k} \alpha_j^* y_{n-j} + h \sum\limits_{j=1}^{k} \beta_j^* f_{n-j} + h\beta_0^* f_n^{[s]},\\
 & s = s+1,\\
 & \text{if } s < m-1, \text{ go to E}_1, \text{ else}\\
\text{E:} & f_n = f(x_n, y_n^{[m]}).
\end{array}$$

For the *Adams methods* the P$_k$EC$_k$E algorithm has the following form:

$$\begin{aligned}
&\text{P:} && y_n^{[0]} = y_{n-1} + h \sum_{j=1}^{k} \beta_j f_{n-j},\\
&\text{E:} && f_n^{[0]} = f(x_n, y_n^{[0]}),\\
&\text{C:} && y_n = y_{n-1} + h \sum_{j=1}^{k} \beta_j^* f_{n-j} + h\beta_0^* f_n^{[0]},\\
&\text{E:} && f_n = f\,(x_n, y_n).
\end{aligned}$$

Left endpoint values of the absolute stability intervals for the Adams methods: explicit, implicit and P$_k$EC$_k$E, are presented in Table 7.10.

Table 7.10. Left endpoint values of the absolute stability intervals of the Adams methods

	the Adams method		
k	explicit	implicit	P$_k$EC$_k$E
1	-2	$-\infty$	-2
2	-1	-6	-2.4
3	-0.55	-3	-2
4	-0.3	-1.83	-1.4
5	-0.18	-1.18	-1.05
6	-0.12	-0.78	-0.76

The following theorem is of major importance for practical applications of the predictor-corrector methods.

Theorem 7.3. *If the order of the predictor p_p is not less than the order of the corrector p, $p_p \geq p$, then for any number of corrector iterations ($m = 1, 2, ...$) the following holds:*

$$y_n^{[m]} - y\,(x_n) = c_{p+1}^* h^{p+1} y^{(p+1)}\,(x_n) + O\left(h^{p+2}\right), \tag{7.61}$$

where c_{p+1}^* is the error constant of the corrector method.
If the corrector order p is larger than the predictor order p_p, $p = p_p + r$, $r \geq 1$, then the formula (7.61) is true for $m > r$.

The above theorem states: if the predictor is sufficiently accurate, i.e., with the order not less than the corrector's order, then for sufficiently small step-sizes h

(i.e., when the main part of the approximation error is dominating) the order of the corrector method is achieved in the PC algorithm, even when one corrector iteration (one step of the fix point iterative method solving the corrector's nonlinear equation (7.60)) is performed only. If the predictor is less accurate, then more iterations of the corrector is needed to achieve the corrector's order, which is the maximal possible order in the PC structure.

7.2.5. Predictor-corrector methods with a variable step-size[*]

Estimation of the approximation error in the PC methods

According to the Theorem 7.3, if only the order of the predictor p_p is not less than the order of the corrector p, then the order of the PC method is equal to the corrector's order p. If we take the main part of the approximation error as its estimate,

$$\delta_n\left(h\right) = c^*_{p+1}\, y^{(p+1)}(x_n)\, h^{p+1}, \tag{7.62}$$

then an estimation of the value of the derivative $y^{(p+1)}(x_n)$ is needed only. This can be done using, e.g., an appropriate backward difference formula (constant step-size is assumed),

$$y^{(p+1)}\left(x_n\right) \cong \frac{\nabla^{p+1} y\left(x_n\right)}{h^{p+1}},$$

which implies

$$h^{p+1} y^{(p+1)}(x_n) \cong \nabla^{p+1} y\left(x_n\right) \approx \nabla^{p+1} y_n, \tag{7.63}$$

where the backward difference is calculated using the points $y_n, y_{n-1}, \cdots, y_{n-p-1}$, i.e.,

$$\begin{aligned} \nabla y_n &= y_n - y_{n-1}, \\ \nabla^2 y_n &= \nabla y_n - \nabla y_{n-1}, \\ \nabla^3 y_n &= \nabla^2 y_n - \nabla^2 y_{n-1}, \\ &\text{etc.} \end{aligned}$$

As a result, the following error estimate is obtained:

$$|\delta_n\left(h\right)| \approx |c^*_{p+1} \nabla^{p+1} y_n|, \tag{7.64}$$

where c^*_{p+1} is the corrector's error constant. The estimate (7.64) can be obtained calculating the value of the backward difference $\nabla^{p+1} y_n$, based on the points $y_n, y_{n-1}, ..., y_{n-p-1}$. However, for the PC methods with equal orders of the predictor and the corrector there is a more efficient way of an approximation of the error.

[*]Optional.

Consider a *PC method with an equal order p of the predictor and of the corrector.* It follows from the Theorem 7.3 that both the predictor and and the PC method are of the order p. Denote by $y_n^{[0]}$ the result of the predictor's and by y_n the result of the corrector's iterations (y_n is here, in general, a result of m corrector's iterations, $y_n = y_n^{[m]}$, $m = 1$ or $m > 1$). We can write

$$\begin{aligned} y_n^{[0]} - y(x_n) &= c_{p+1}\, y^{(p+1)}(x_n)h^{p+1} + O(h^{p+2}), \\ y_n - y(x_n) &= c^*_{p+1}\, y^{(p+1)}(x_n)h^{p+1} + O(h^{p+2}). \end{aligned}$$

Neglecting the terms $O(h^{p+2})$, we can eliminate the unknown derivative $y^{(p+1)}(x_n)$ from these equations. This leads to

$$y_n - y(x_n) = \frac{c^*_{p+1}}{c_{p+1}}(y_n^{[0]} - y(x_n)).$$

Adding $-\frac{c^*_{p+1}}{c_{p+1}}(y_n - y(x_n))$ to both sides of this equality results in

$$\frac{c_{p+1} - c^*_{p+1}}{c_{p+1}}(y_n - y(x_n)) = \frac{c^*_{p+1}}{c_{p+1}}(y_n^{[0]} - y_n),$$

which can be rewritten in the form

$$y_n - y(x_n) = \frac{c^*_{p+1}}{c_{p+1} - c^*_{p+1}}(y_n^{[0]} - y_n). \tag{7.65}$$

The above equality represents a local approximation error estimate (defined with an accuracy corresponding to the omitted terms $O(h^{p+2})$):

$$\delta_n\,(h_{n-1}) = \frac{c^*_{p+1}}{c_{p+1} - c^*_{p+1}}(y_n^{[0]} - y_n), \tag{7.66}$$

which is very easy to calculate using only the predicted and the final (corrected) values $y_n^{[0]}$ and y_n of the PC method, respectively. For example, for the Adams PC method $\mathrm{P_4EC_3E}$ with the 4-stage Adams-Bashforth predictor and 3-stage Adams-Moulton corrector we have (see Tables 7.8 and 7.9 for the error constants, both methods are of order 4)

$$y_n - y(x_n) = -\frac{19}{270}(y_n^{[0]} - y_n).$$

For the $\mathrm{P}_k\mathrm{EC}_k\mathrm{E}$ Adams methods, where the order of the predictor is less by one than the order of the corrector, we have two possibilities:

1. Instead of the $\mathrm{P_kEC_kE}$ method, we apply the $\mathrm{P_k(EC_k)^2E}$ method, i.e., two subsequent iterations of the corrector algorithm are performed ($m = 2$, see Theorem 7.3). This allows to retain the order of the PC algorithm equal to the corrector's order – but the efficient formula (7.66) for error estimation cannot be applied. A less effective approach based on the general formula (7.64) can be used only.
2. We start as in the $\mathrm{P_kEC_{k\text{-}1}E}$ method, i.e., using the predictor based on k and the corrector based on $k-1$ points – then the predictor's and corrector's orders are equal. But, after acceptance of the approximation error (a successful step), the final corrector's step can be made applying the k-step corrector, i.e., the algorithm which is more accurate than the $(k-1)$-step one. This can only increase a reliability of the error estimation, as a step of the Adams-Moulton method of order k+1 should be more accurate than a step of this method of (lower) order k. However, this procedure will be effective if a transition from a solution value calculated using the $(k-1)$-step corrector to the solution value of the k-step corrector is simple, requiring a small number of calculations. This is precisely the case if we apply the Adams methods in a *difference form.*

Adams methods in the difference form

Applying, instead of the Lagrange interpolation formula, the Newton's interpolation formula (see the reasoning in Section 5.1), one obtains a formula defining the Adams-Bashforth methods which is equivalent to the standard one, but is based not on the function values, but on backward differences of this values:

$$y_n = y_{n-1} + h \cdot \sum_{j=0}^{k-1} \gamma_j \cdot \nabla^j f_{n-1}, \tag{7.67}$$

where $f_{n-1} = f\left(x_{n-1}, y_{n-1}\right)$, $f_{n-2} = f\left(x_{n-2}, y_{n-2}\right), \dots$, and the backward differences are based on the points $f_{n-1}, f_{n-2}, \dots, f_{n-k}$, i.e.,

$$\begin{aligned} \nabla f_{n-1} &= f_{n-1} - f_{n-2}, \\ \nabla^2 f_{n-1} &= \nabla f_{n-1} - \nabla f_{n-2}, \\ \nabla^3 f_{n-1} &= \nabla^2 f_{n-1} - \nabla^2 f_{n-2}, \\ &\text{etc.} \end{aligned}$$

An similar formula for the Adams-Moulton methods, based on the backward differences, obtained in an analogous way, is as follows:

$$y_n = y_{n-1} + h \cdot \sum_{j=0}^{k} \gamma_j^* \cdot \nabla^j f_n, \tag{7.68}$$

where the differences are calculated using the points $f_n = f_n(x_n, y_n)$, f_{n-1}, $f_{n-2}, \dots, f_{n-k}$.

The values γ_j and γ_j^* for the above defined Adams methods are given in Table 7.11. Error constants are defined in this case uniquely by this parameters. It is interesting, that much more simple formulae now hold : $c_{k+1} = -\gamma_k$, $c_{k+1}^* = -\gamma_k^*$ (comp. (7.49)).

Table 7.11. Parameters of the explicit and implicit Adams methods in the difference form

j	0	1	2	3	4	5	6	7
γ_j	1	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$	$\frac{19087}{60480}$	$\frac{36799}{120960}$
γ_j^*	1	$-\frac{1}{2}$	$-\frac{1}{12}$	$-\frac{1}{24}$	$-\frac{19}{720}$	$-\frac{3}{160}$	$-\frac{863}{60480}$	$-\frac{1375}{120960}$

Taking the explicit and implicit Adams methods in the difference form, the Adams $\mathrm{P}_k\mathrm{EC}_{k-1}\mathrm{E}$ predictor-corrector algorithm can be written in the following *equivalent difference form*:

$$\mathrm{P}: \quad y_n^{[0]} = y_{n-1} + h\sum_{j=0}^{k-1} \gamma_j \, \nabla^j f_{n-1}, \tag{7.69a}$$

$$\mathrm{E}: \quad f_n^{[0]} = f(x_n, y_n^{[0]}), \tag{7.69b}$$

$$\mathrm{C}: \quad y_n = y_{n-1} + h\sum_{j=0}^{k-1} \gamma_j^* \nabla^j f_n^{[0]}, \tag{7.69c}$$

$$\mathrm{E}: \quad f_n = f\left(x_n, y_n\right), \tag{7.69d}$$

where $\nabla^j f_{n-1}$, $j = 0, 1, \ldots, k-1$, are the backward differences defined using the values $f_{n-1}, \ldots, f_{n-k}$, and $\nabla^j f_n^{[0]}$, $j = 0, 1, ..., k-1$, are the backward differences defined using the values $f_n^{[0]}, f_{n-1}, \ldots,\ f_{n-k+1}$. Because the predictor and the corrector are in the considered case of the same order, then the formula (7.66) can be applied for the error estimation, which takes now the form

$$\delta_n\left(h_{n-1}\right) = \frac{\gamma_k^*}{\gamma_k - \gamma_k^*}(y_n^{[0]} - y_n). \tag{7.70}$$

Taking into account the equality (see, e.g., Table 9)

$$\gamma_j^* = \gamma_j - \gamma_{j-1},$$

the presented above basic difference form of the $\mathrm{P}_k\mathrm{EC}_{k-1}\mathrm{E}$ Adams method can be transformed to an equivalent but simpler form, with the following corrector:

$$\mathrm{C}: \quad y_n = y_n^{[0]} + h\gamma_{k-1}\nabla^k f_n^{[0]}. \tag{7.71}$$

The formula (7.70) for the error estimation takes now the form

$$\delta_n\left(h_{n-1}\right) = -h\gamma_k^*\, \nabla^k f_n^{[0]}, \tag{7.72}$$

where $\nabla^k f_n^{[0]}$ is a backward difference defined using the values $f_n^{[0]}$, $f_{n-1}, \ldots, f_{n-k}$. Note that this difference can be calculated directly after the calculation of the predictor's solution value – avoiding the calculation of the corrector's iteration, if the obtained accuracy is not satisfactory. However, this is in fact not so significant, because having calculated $y_n^{[0]}$ and $\nabla^k f_n^{[0]}$, the corrector's iteration in the form (7.71) is very simple.

The formula (7.72) can be very easily transformed to the form

$$\delta_n\left(h_{n-1}\right) = y_n - y_n^{(+1)}, \tag{7.73}$$

where y_n – the result of a step performed by the P$_k$EC$_{k-1}$E Adams algorithm (7.69a)-(7.69d), and $y_n^{(+1)}$– the result of a step of the Adams P$_k$EC$_k$E algorithm:

$$\mathrm{P}: \quad y_n^{[0]} = y_{n-1} + h\sum_{j=0}^{k-1} \gamma_j\, \nabla^j\, f_{n-1}, \tag{7.74a}$$

$$\mathrm{E}: \quad f_n^{[0]} = f(x_n, y_n^{[0]}), \tag{7.74b}$$

$$\mathrm{C}: \quad y_n^{(+1)} = y_{n-1} + h\sum_{j=0}^{k} \gamma_j^* \nabla^j f_n^{[0]}, \tag{7.74c}$$

$$\mathrm{E}: \quad f_n = f\left(x_n, y_n\right). \tag{7.74d}$$

The equality (7.73) can be obtained subtracting the corrector's equations (7.69c) and (7.74c) and comparing the result with (7.72).

An application of the error estimation valid for the P$_k$EC$_{k-1}$E algorithm to the P$_k$EC$_k$E one, i.e., to a more accurate one, can only increase the reliability of the estimation. Therefore, applying the algorithm P$_k$EC$_k$E with the error estimation for P$_k$EC$_{k-1}$E, we can very effectively (without any additional calculations) calculate this estimate from (7.73). This is because a solution value of the $(k-1)$-step corrector is a by-product of the calculation of a solution value of the k-step corrector – due to the application of the Adams methods in the difference form.

Variable step-size and order

The formula for a proposed step-size correction, discussed in Section 7.1.3 for the single-step methods, is general and applies also for the multistep methods,

$$h_n = s \cdot \alpha \cdot h_{n-1}. \tag{7.75}$$

In the above equality, the parameter α is given by the formula

$$\alpha = \left[\frac{\varepsilon}{|\delta_n\left(h_{n-1}\right)|}\right]^{\frac{1}{p+1}}, \tag{7.76}$$

where $|\delta_n\left(h_{n-1}\right)|$ is an estimate of the absolute value of the main part of the approximation error (at an iteration starting from x_{n-1} with a step-size h_{n-1}), s is a safety coefficient, and ε is an assumed tolerance (a required accuracy),

$$\varepsilon = |y_n| \cdot \varepsilon_r + \varepsilon_a, \tag{7.77}$$

where ε_r and ε_a are parameters of a relative and absolute tolerance.

In multistep methods based on equally distant points (as in all the methods presented in this Section), changing the length of the step h is more difficult than is the single-step methods.

When $s\alpha > 1$ in the formula (7.75), then *the step-size can be increased.* To calculate the next solution point with the new step-size $h_n = s\alpha h_{n-1}$, we must provide solution points and corresponding values of the right-hand side functions (i.e., values of y and $f(x, y)$) at the points $x = x_n - j(s\alpha h_{n-1}), j = 1, \ldots, k+1$. These are, in general, *new $k-1$ points* $x_{n-1}, ..., x_{n-k+1}$, because they must be distant from the last point x_n by intervals defined by the new step-size h_n, and not the old step-size h_{n-1}. In general, at x_n the solution and derivative values are known only, it is the last newly calculated solution point – the initial point for the next step of the algorithm.

A general rule in this situation can be as follows (for an increase of the step-size): *Evaluate a function (a polynomial) interpolating the obtained solution points on the interval $[x_{n-l}, x_n]$ containing all new points needed, $x_{n-j} = x_{n-1} - jh_n$, $j = 1, ..., k+1$. Then calculate y_{n-j}, $j = 1, ..., k+1$ as the values of this interpolation function at these new points, then finally the values $f(x_{n-j}, y_{n-j})$ of the right-hand sides of the differential equations.*

Moreover, because the described step-size correction rule is not very simple, then we can design an algorithm of the method which performs the step-size increase less frequently, e.g., only if the proposed correction is significant, indicated by a value of $s\alpha$ sufficiently larger than 1.

In the case of a *step-size decrease* (i.e., after a failed step, not satisfying the accuracy requirements) the procedure can be similar, only we must skip the new point x_n as not acceptable and the previous point x_{n-1} (not updated) remains as the current initial point for the next iteration, with the decreased step-size. The interpolation can be performed using the last k solution points.

For the Adams methods, only previous values of the right-hand side functions are needed, $f_{n-j} = f(x_{n-j}, y_{n-j})$. Therefore, it is more effective to interpolate directly these values. The Newton's interpolating polynomial (5.17), see Section 5.1.2, can be used here.

In the multistep predictor-corrector methods not only the step-size is changed, also *changes of the order* can be applied to control the solution accuracy. For the Adams methods, changes of the order are especially convenient when the difference version is used. An increase of the order should lead to decrease of the error, thus to an increase of the accuracy. However, it should be pointed out that, at least theoretically, an increase of the order does not necessarily lead to a decrease of the error. In the general formula for the error estimation, there is a product of the error constant and the backward difference, $c^*_{p+1} \cdot \bigtriangledown^{p+1} y_n$, where the backward difference is an approximation of the product $h^{p+1} \cdot y^{(p+1)}\left(x_n\right)$. Theoretically, there are functions, e.g., $\exp(\beta x),\ \beta > 1$, when the value of the derivative increases with the increase of the derivative order. If this increase is faster than the decrease of the term $c^*_{p+1} h^{p+1}$, then the situation when the error increases while the order also increases is theoretically possible.

The methods with a constant step-size may be convenient, e.g., in simulations of dynamic systems. But *effective, professional implementations of the Adams methods utilize more algorithmically complex versions with not equally spaced points, with frequent changes of both step-size and order. They are usually self-starting, changing of the step-size and order is in these implementations relatively simple. These methods are currently regarded as outstanding in their class.*

7.3. Stiff systems of differential equations

Consider the following example:

$$y'(x) = \mathbf{A}\, y(x),$$

where

$$\mathbf{A} = \begin{bmatrix} -667 & 333 \\ 666 & -334 \end{bmatrix}, \quad y_0 = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \quad x \in [0, 10]\,.$$

It can be easily calculated that

$$\mathrm{sp}\,(\mathbf{A}) = \{\lambda_1, \lambda_2\} = \{-1, -1000\}\,,$$

and the solution is

$$y_1\,(x) = e^{-x} - e^{-1000x},$$
$$y_2\,(x) = 2e^{-x} + e^{-1000x}.$$

The solution varies very fast in the first initial phase where the term e^{-1000x} dominates, then it is slow varying. An effective numerical algorithm should calculate an approximate solution with a small step-size in the first phase, then with a much larger step-size in the remaining phase. Applying, e.g., the P_3EC_3E predictor corrector method, we have from the absolute stability requirement (the absolute stability interval is $[-2, 0]$):

$$|\lambda_1 h| < 2 \;\Rightarrow\; h < \frac{2}{|\lambda_1|} = 2,$$

$$|\lambda_2 h| < 2 \;\Rightarrow\; h < \frac{2}{|\lambda_2|} = 2 \cdot 10^{-3}.$$

The PC procedure calculates first the subsequent solution points with a sufficiently small step-size, less than $2 \cdot 10^{-3}$. Then the procedure should calculate solution points with a much larger step-size – but it cannot be done, as the step-size cannot be made greater than $2 \cdot 10^{-3}$, due to the absolute stability requirements. For a larger step-size, the discrete dynamic equation of the procedure becomes unstable and starts to generate large and fast increasing errors. This results in an increase of errors and forces a decrease of the step-size – as it cannot be outside the absolute stability set.

We say that a system of linear differential equations $y'(x) = \mathbf{A}y(x)$ is *stiff* (or *ill-conditioned*), if

$$\frac{|\lambda_{\max}|}{|\lambda_{\min}|} \ggg 1,$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and the smallest (in absolute values) eigenvalues of the matrix $\mathbf{A}$. If the system of equations is nonlinear, than its linear approximation (i.e., the matrix of the first derivatives of the right-hand side functions, the Jacobian matrix) at a current point or region of interest should be considered.

For the stiff systems of equations numerical methods with large sets of absolute stability should be applied. The predictor-corrector Adams methods or explicit Runge-Kutta methods described in Section 7.1.1 do not have this property.

Example 7.6. For the *Euler's method* (an explicit method) we have:

$$y_n = y_{n-1} + h \cdot f\left(x_{n-1}, y_{n-1}\right).$$

For the differential equation

$$y' = \lambda y, \tag{7.78}$$

where $\lambda < 0$, we get the discrete equation of the method

$$y_n = y_{n-1} + h\lambda y_{n-1} = (1 + h\lambda)\; y_{n-1},$$

i.e.,

$$y_n = (1 + h\lambda)^n \, y_0.$$

The method is absolutely stable, i.e., $(1 + h\lambda)^n \to 0$, if $|1 + h\lambda| < 1$, which implies

$$h\,|\lambda| < 2, \quad \text{i.e.,} \quad h < \frac{2}{|\lambda|}.$$

□

Example 7.7. For the *implicit Euler's method* (an implicit method) we have:

$$y_n = y_{n-1} + h \cdot f\left(x_n, y_n\right)$$

and with the differential equation (7.78) we have

$$y_n = y_{n-1} + h\lambda y_n.$$

This results in the discrete equation of the method

$$y_n = \frac{1}{1 - \lambda h} y_{n-1}.$$

Because

$$\forall \lambda < 0 \qquad \frac{1}{1 - \lambda h} < 1,$$

the implicit Euler's method is absolutely stable for any positive step-size h (the interval of absolute stability is $(-\infty,\ 0)$. □

A family of BDF (backward differentiation formula) multistep methods, implemented in the predictor-corrector structure, is suitable for the stiff systems of diffeential equations. BDF methods have very large sets of absolute stability, with absolute stability intervals equal to $(-\infty,\ 0)$.

BDF methods

The explicit BDF (Backward Differentiation Formula) methods:

$$y_n = \sum_{j=1}^{k} \alpha_j y_{n-j} + h \cdot \beta_1 f\left(x_{n-1}., y_{n-1}\right). \tag{7.79}$$

The implicit BDF methods:

$$y_n = \sum_{j=1}^{k} \alpha_j^* y_{n-j} + h \cdot \beta_0^* f\left(x_n, y_n\right). \tag{7.80}$$

Parameters of the explicit and implicit BDF methods are given in Tables 7.12 and 7.13, respectively.

Table 7.12. Parameters of the explicit BDF methods

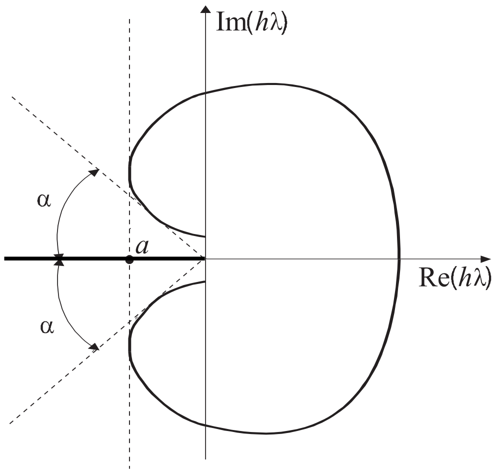
c_{p+1}	p	k	β_1	α_1	α_2	α_3	α_4	α_5	α_6
$-\frac{1}{2}$	1	1	1	1					
$-\frac{1}{3}$	2	2	2	0	1				
$-\frac{1}{4}$	3	3	3	$-\frac{3}{2}$	3	$-\frac{1}{2}$			
$-\frac{1}{5}$	4	4	4	$-\frac{10}{3}$	6	-2	$\frac{1}{3}$		
$-\frac{1}{6}$	5	5	5	$-\frac{65}{12}$	10	-5	$\frac{5}{3}$	$-\frac{1}{4}$	
$-\frac{1}{7}$	6	6	6	$-\frac{77}{10}$	15	-10	5	$-\frac{3}{2}$	$\frac{1}{5}$

Table 7.13. Parameters of the implicit BDF methods

c_{p+1}	p	k	β_0^*	α_1^*	α_2^*	α_3^*	α_4^*	α_5^*	α_6^*
$\frac{1}{2}$	1	1	1	1					
$\frac{2}{9}$	2	2	$\frac{2}{3}$	$\frac{4}{3}$	$-\frac{1}{3}$				
$\frac{3}{22}$	3	3	$\frac{6}{11}$	$\frac{18}{11}$	$-\frac{9}{11}$	$\frac{2}{11}$			
$\frac{12}{125}$	4	4	$\frac{12}{25}$	$\frac{48}{25}$	$-\frac{36}{25}$	$\frac{16}{25}$	$-\frac{3}{25}$		
$\frac{10}{137}$	5	5	$\frac{60}{137}$	$\frac{300}{137}$	$-\frac{300}{137}$	$\frac{200}{137}$	$-\frac{75}{137}$	$\frac{12}{137}$	
$\frac{20}{343}$	6	6	$\frac{60}{147}$	$\frac{360}{147}$	$-\frac{450}{147}$	$\frac{400}{147}$	$-\frac{225}{147}$	$\frac{72}{147}$	$-\frac{10}{147}$

Because only the implicit methods have infinite sets of the absolute stability, only the predictor-corrector (PC) algorithms are used. Moreover, in order to maintain the sets of the absolute stability infinite *the corrector algorithms must be precisely performed*, i.e., until solutions of nonlinear corrector equations are found with a prescribed accuracy. This means that the corrector iterations, performed using the fix-point method or the Newton's algorithm, should be performed until a certain assumed solution accuracy is satisfied. Shapes of the absolute stability sets for the implicit BDF methods is shown in Fig. 7.8.

The definition of the absolute stability applies to all difference numerical methods for solving systems of differential equations, also to the single-step methods presented in Section 7.1. The RK and RKF methods presented there are explicit methods, having bounded sets of the absolute stability. Thus, they are not effective for stiff systems of differential equations. For such equations, also *implicit single-step methods* (of Runge-Kutta type) have been derived, having large sets of the absolute stability. The simplest example can be here the implicit Euler's method considered earlier in this section.



k	a	α
1	0.0	90°
2	0.0	90°
3	0.1	88°
4	0.7	73°
5	2.4	51°
6	6.1	18°

Figure 7.8. Regions of the absolute stability of the implicit BDF methods

Problems

1. Check if the multistep method
$$y_n = \frac{9y_{n-1} - y_{n-3}}{8} + 3h\frac{f_n + 2f_{n-1} - f_{n-2}}{8},$$
where $f_i = f\left(x_i, y_i\right)$, is convergent. If yes, determine the order and the error constant of this method.
2. The polynomial $\rho(z) = -z^2 + 1$ is defined. Choose a polynomial $\sigma(z)$ in such a way, that a stable explicit two-step method ($k = 2$) of maximal order is obtained.
3. What can be said about the absolute stability of a two-step ($k = 2$) explicit methods having maximal order ? (see the Example in Section 7.2.3)
4. Check the stability of the explicit BDF methods, for $k = 1, 2, 3$.
5. Check, for which values of the step-size h an application of the two-step explicit Adams method to the differential equation

$$y' = -5y$$

will be stable.

6. The following system of differential equations:

$$y_1'(x) = a\,y_1(x) - b\,y_1(x)\,y_2(x),$$
$$y_2'(x) = c\,y_1(x)\,y_2(x) - d\,y_2(x),$$

describes a "predator-prey model", where $y_1(x)$ describes a population of preys and $y_2(x)$ a population of predators.
Assuming: $a = 10$, $b = 0.2$, $c = 0.02$, $d = 5$ and initial conditions: $y_1(0) = 200$, $y_2(0) = 10$, calculate trajectories $y_1(x)$ and $y_2(x)$ for $x \in [0,\ 4]$, applying:
   a) the RK4 method with a fixed step-size, choosing the step-size value by trial-and-error method, i.e., repeating calculations with a successively decreasing step-size, until its sufficiently small value is found – a further decrease of the step-size does not practically influence the solution (e.g., trajectories visually the same on the screen);
   b) the RK4 method with a variable step-size, automatically selected by the method using the step-doubling approach.

   The applied algorithms should be implemented in MATLAB environment, plotting the trajectories of the solution. Check correctness of the results, using one of the MATLAB procedures for systems of ordinary differential equations.

7. Calculate the proceeding problem, but using the RKF45 method, instead of the RK4; in point b) estimate the error using the algorithm suitable for the RKF methods.

8. Calculate the Problem 6a, but using the four-step PC Adams method P_4EK_4E, instead of the RK4 method.

9.* Calculate the Problem 6b, but using the four-step PC Adams method P_4EK_4E, instead of the RK4. Implement the error estimation procedure defined as for the P_4EK_3E method, based on the difference between solution values calculated by the predictor and the corrector.

*Optional.

Chapter 8

Numerical differentiation and integration

Formulae for a numerical approximation of derivatives of functions of one or several variables are essential in many applications of numerical algorithms. They can be used for numerical calculation of a velocity and an acceleration of known position trajectories (e.g., in robotics), for numerical calculation of the derivatives in optimization algorithms and in algorithms solving systems of nonlinear equations, in the formulation of discrete implementations of continuous-time control algorithms, in the sensitivity analysis (of models, control laws, optimal solutions, etc.), in numerical algorithms for solving systems of ordinary or partial differential equations, and many others.

8.1. Numerical approximation of derivatives

A simple approximation of a first order derivative can be obtained using a divided difference; we have different possibilities of the choice of points defining the difference, leading to the approximation by:

1. the backward difference formula:
$$f^{'}(x) \approx \frac{f(x) - f(x-h)}{h} = D_{f-}(x, h), \tag{8.1}$$
2. the forward difference formula:
$$f^{'}(x) \approx \frac{f(x+h) - f(x)}{h} = D_{f+}(x, h), \tag{8.2}$$
3. the centered difference formula:
$$f^{'}(x) \approx \frac{f(x+h) - f(x-h)}{2h} = D_{fc}(x, h), \tag{8.3}$$

where h is the approximation step (interval), i.e., the distance of the point (points) used to calculate the difference from the point at which the derivative is approximated.

As one could expect, the centered difference leads to a better approximation than the backward or the forward differences (i.e., with a smaller approximation error), but it cannot be used in all applications. Formulae for approximation errors can be derived using the Taylor's series expansions. For the backward difference, we have:

$$f(x-h) = f(x) - f^{'}(x)h + \frac{f^{(2)}(\alpha)}{2!}h^2, \tag{8.4}$$

where $\alpha \in [x-h, x]$. Therefore

$$f^{'}(x) = \frac{f(x) - f(x-h)}{h} + \frac{f^{(2)}(\alpha)}{2!}h = D_{f-}(x,h) - O(h), \tag{8.5}$$

where $O(h) = \frac{f^{(2)}(\alpha)}{2!}h$ is the approximation error (the error of the approximation method, the truncation error). An analogous result holds for the forward difference. But for the centered difference we have (assuming $f \in C^3$):

$$\begin{aligned} f(x+h) &= f(x) + f^{'}(x)h + \frac{f^{(2)}(x)}{2!}h^2 + \frac{f^{(3)}(\alpha_+)}{3!}h^3, \\ f(x-h) &= f(x) - f^{'}(x)h + \frac{f^{(2)}(x)}{2!}h^2 - \frac{f^{(3)}(\alpha_-)}{3!}h^3, \end{aligned}$$

where $\alpha_+ \in (x, x+h),\ \alpha_- \in (x-h, x)$. Subtracting the last equality from the previous one, we get:

$$f(x+h) - f(x-h) = 2f^{'}(x)h + \frac{f^{(3)}(\alpha_+) + f^{(3)}(\alpha_-)}{3!}h^3, \tag{8.6}$$

which results in

$$D_{fc}(x,h) = f^{'}(x) + \frac{f^{(3)}(\alpha_+) + f^{(3)}(\alpha_-)}{12}h^2 = f^{'}(x) + O(h^2). \tag{8.7}$$

This proves that a derivative approximation based on the centered difference is one order of magnitude better (more accurate) than the approximations based on the backward or forward differences (the reason is that it is equivalent to a three-point approximation, based on points $x-h, x, x+h$, see Problem 3 at the end of the Chapter). Therefore, the use of the centered difference is recommended, if possible, e.g. for a numerical calculation of the derivative of a known function. However, in many practical applications, e.g., in automatic control, there is the task of a current (on-line) approximation of the derivative of a measured signal (or estimated, based on current measurements) – therefore, future values of the signal are not known. In such cases the backward difference formula must be applied.

Example 8.1. We shall calculate numerically, in MATLAB environment (thus using the double precision), the derivative of the function cos(x) at $x = \pi/4$, with the step $h = 0.01$, applying the backward and centered difference formulae.

For the backward difference formula we have ($\tilde{D}$ denotes a numerically calculated value of a difference approximation of a derivative D):

$$\begin{aligned}\tilde{D}_{f-}\Big(\frac{\pi}{4}, 0.01\Big) &= \frac{0.707106781186548 - 0.714142376103440}{0.01}\\ &= -0.703559491689199.\end{aligned}$$

For the centered difference formula we have:

$$\begin{aligned}\tilde{D}_{fc}\Big(\frac{\pi}{4}, 0.01\Big) &= \frac{0.700000476180791 - 0.714142376103440}{0.02}\\ &= -0.707094996132451.\end{aligned}$$

Thus, the absolute errors are ($\cos'(\pi/4) = -\sin(\pi/4) = -0.707106781186548$):

$$\tilde{D}_{f-}\Big(\frac{\pi}{4}, 0.01\Big) - \cos'\Big(\frac{\pi}{4}\Big) = 0.003547289497349,$$

$$\tilde{D}_{fc}\Big(\frac{\pi}{4}, 0.01\Big) - \cos'\Big(\frac{\pi}{4}\Big) = 0.000011785054096\,15838.$$

These calculations confirm a significantly better accuracy of the approximation based on the centered difference formula. $\square$

In the analysis presented above, *the errors of the approximation method* were considered only, but there are also numerical errors of a computer representation of numbers and of floating-point arithmetic operations. Assume the step length h is a power of 2 (therefore, its machine representation is accurate and divisions by h do not introduce errors) and the absolute values of the representation errors of the numbers $f(x)$ and $f(x-h)$ are not larger than Eps, i.e., $fl(f(x)) = f(x)(1+\varepsilon_1)$, $fl(f(x-h)) = f(x-h)(1+\varepsilon_2)$, $|\varepsilon_i| \le Eps,\ i = 1,2,\ Eps \ge eps.$ Then we have, for the backward difference formula:

$$\begin{aligned}\tilde{D}_{f-}(x,h) &= \frac{[f(x)(1+\varepsilon_1) - f(x-h)(1+\epsilon_2)](1+\varepsilon_3)}{h}\\ &\underset{1}{=} \frac{f(x) - f(x-h) + f(x)\varepsilon_1 - f(x-h)\varepsilon_2}{h}(1+\varepsilon_3)\\ &= \frac{f(x)-f(x-h)}{h}\left[1 + \frac{f(x)\varepsilon_1 - f(x-h)\varepsilon_2}{f(x)-f(x-h)}\right](1+\varepsilon_3)\\ &\underset{1}{=} \frac{f(x)-f(x-h)}{h}\left[1 + \frac{f(x)\varepsilon_1 - f(x-h)\varepsilon_2}{f(x)-f(x-h)} + \varepsilon_3\right]\\ &= \frac{f(x)-f(x-h)}{h}\left[1 + \delta(f,x,h)\right],\end{aligned}$$

where $\epsilon_3 \leq eps$ and $\tilde{D}_{f+}(x,h)$ denotes an approximation of the derivative using the backward difference formula, obtained numerically (with machine number representation and floating point arithmetic errors). Therefore, the estimate $\delta(f,x,h)$ of the upper bound of the relative error is

$$\begin{aligned}
|\delta(f,x,h)| &= \left| \frac{f(x)\varepsilon_1 - f(x-h)\varepsilon_2}{f(x)-f(x-h)} + \varepsilon_3 \right| \\
&\leq \frac{|f(x)| + |f(x-h)|}{|f(x)-f(x-h)|} Eps + eps \\
&\approx \frac{2|f(x)|}{|f(x)-f(x-h)|} Eps + eps. \qquad (8.8)
\end{aligned}$$

Because small steps should be applied for the derivative approximation (the smaller the step, the smaller the error of the method), then the first term in the above formula is usually significantly larger than the second term (eps) – and may be very large, due to a possibly very small number in the denominator.

The above analysis indicates that there is a counteracting influence of the step-size h on the error of the approximation method and on the final error of the numerical (floating point) operations – diminishing the step-size h results in a smaller value of the first error and in an increase of the second one (due to a decrease of $|f(x+h) - f(x)|$). Therefore, a trade-off should exist resulting in an optimal step-size which minimizes the overall error. Before passing to this analysis, let us present the numerical error in the form

$$|\delta(f,x,h)| \approx \frac{2|f(x)|}{|f'(x)|h} Eps + 2eps. \qquad (8.9)$$

Therefore, estimate of the upper bound of the overall absolute error is

$$|\tilde{D}_{f-}(x,h) - f'(x)| \leq \frac{|f^{(2)}(\alpha)|}{2} h + \frac{2}{h}|f(x)|Eps + 2|f'(x)|eps. \qquad (8.10)$$

This function has a clear minimum, a necessary condition for this minimum is

$$\frac{|f^{(2)}(\alpha)|}{2} - \frac{2}{h^2}|f(x)|Eps = 0,$$

which yields the optimal step-size describing an optimal trade-off between the approximation error and the numerical errors:

$$\hat{h}_- = 2\sqrt{\frac{|f(x)|}{|f^{(2)}(\alpha)|} Eps}. \qquad (8.11)$$

A direct application of this formula to the step-size selection requires a numerical approximation of the second derivative (see further considerations). However, even if we do not evaluate the coefficient by which eps is multiplied in the above formula, we can apply the following "engineering rule": *the optimal step-size* h *should be of the order of* $\sqrt{Eps}$.

For a derivative approximation based on the progressive and centered difference formulae, the error analysis can be performed in a quite analogous way. The optimal step-size for the progressive difference is the same as for the backward difference, for the centered difference it depends on the derivative of the third order. These calculations are left to the reader (see problems at the end of this chapter).

Derivatives of second order can be evaluated using a recurrent definition of the derivative of the first order, e.g., for approximations based on the backward and centered differences we have:

1. for the backward difference:

$$\begin{aligned} f^{(2)}(x) &\approx \frac{D_{f-}(x,h) - D_{f-}(x-h,h)}{h} \\ &= \frac{\frac{f(x)-f(x-h)}{h} - \frac{f(x-h)-f(x-2h)}{h}}{h} \\ &= \frac{f(x) - 2f(x-h) + f(x-2h)}{h^2} = D^2_{f-}(x,h), \end{aligned}$$

2. for the centered difference:

$$\begin{aligned} f^{(2)}(x) &\approx \frac{D_{fc}(x+\frac{h}{2},\frac{h}{2}) - D_{fc}(x-\frac{h}{2},\frac{h}{2})}{h} \\ &= \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} \\ &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} = D^2_{fc}(x,h). \end{aligned}$$

Approximations of the derivatives of higher orders can be evaluated in the same way. However, we should remember that the difference formulae for a numerical calculation of the first order derivative are numerically ill-conditioned, thus for the derivatives of higher orders the situation can only be worse.

The resented formulae for a numerical calculation of the derivatives of the first and second order are based on minimal numbers of points (for the first order derivatives – on 2 points, for the second order derivatives – on 3 points). Formulae using more points can also be formulated. The way of proceeding is to evaluate the required derivative as a derivative of an interpolation polynomial based on the assumed number of points. In fact, all formulae given above can also be deduced as derivatives of interpolation polynomials based on appropriately located points. An example given below illustrates this procedure.

Example 8.2. We shall formulate the Lagrange interpolating polynomial based on 3 equally spaced nodes $x_0,\ x_0-h,\ x_0-2h$. Next, we shall evaluate a numerical approximation of the first order derivative of the original function at x_0 as a derivative of the interpolating polynomial.
The Lagrange interpolating polynomial:

$$\begin{aligned} W_2(x) &= f(x_0)\frac{(x-(x_0-h))(x-(x_0-2h))}{(x_0-(x_0-h))(x_0-(x_0-2h))}+ \\ &+ f(x_0-h)\frac{(x-x_0)(x-(x_0-2h))}{(x_0-h-(x_0))(x_0-h-(x_0-2h))}+ \\ &+ f(x_0-2h)\frac{(x-x_0)(x-(x_0-h))}{(x_0-2h-(x_0))(x_0-2h-(x_0-h))} \\ &= f(x_0)\frac{(x-x_0+h)(x-x_0+2h)}{2h^2}+ \\ &+f(x_0-h)\frac{(x-x_0)(x-x_0+2h)}{-h^2}+ \\ &+ f(x_0-2h)\frac{(x-x_0)(x-x_0+h)}{2h^2}. \end{aligned}$$

The derivative of this polynomial is the following function

$$\begin{aligned} W_2'(x) &= f(x_0)\frac{(x-x_0+h)+(x-x_0+2h)}{2h^2}+ \\ &+f(x_0-h)\frac{(x-x_0)+(x-x_0+2h)}{-h^2}+ \\ &+ f(x_0-2h)\frac{(x-x_0)+(x-x_0+h)}{2h^2}. \end{aligned}$$

The value of this derivative at $x=x_0$ is

$$\begin{aligned} f'(x_0) &\approx W_2'(x_0)=f(x_0)\frac{(h+2h)}{2h^2}+f(x_0-h)\frac{2h}{-h^2}+\ f(x_0-2h)\frac{h}{2h^2} \\ &= \frac{3f(x)-4f(x-h)+f(x-2h)}{2h}=D_{f3p-}(x,h). \qquad (8.12) \end{aligned}$$

□

The accuracy of the approximation given by the formula (8.12) is of the same order as for the formula basing on a centered difference derived earlier. This can be easily shown using two Taylor's series expansions of the function $f(x)$ (assuming $f \in C^3$, with the remainder of the Taylor's series in the Lagrange form):

$$\begin{aligned} f(x-h) &= f(x) - f^{'}(x)h + \frac{f^{(2)}(x)}{2!}h^2 - \frac{f^{(3)}(\alpha_1)}{3!}h^3, \\ f(x-2h) &= f(x) - f^{'}(x)2h + \frac{f^{(2)}(x)}{2!}(2h)^2 - \frac{f^{(3)}(\alpha_2)}{3!}(2h)^3, \end{aligned}$$

where $\alpha_1 \in (x-h, x),\ \alpha_2 \in (x-2h, x)$. Multiplying the first equation by 4 and subtracting from the second, we obtain

$$f^{'}(x)2h = 3f(x) - 4f(x-h) + f(x-2h) + \frac{f^{(3)}(\alpha_2)}{3!}(2h)^3 - 4\frac{f^{(3)}(\alpha_1)}{3!}h^3,$$

which yields directly

$$f^{'}(x) = \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h} + O(h^2).$$

Note that the presented reasoning is an alternative way of derivation of the formula (8.12).

Example 8.1. (*ctd*) We shall calculate numerically, using MATLAB, the derivative of the function cos(x) at the point $x = \pi/4$, using the step-size $h = 0.01$ and applying the formula (8.12). We have

$$\tilde{D}_{f3p-}\left(\frac{\pi}{4}, 0.01\right) = -0.707130173813869.$$

Therefore, the absolute error ($\cos^{'}(\pi/4) = -\sin(\pi/4) = -0.707106781186548$) is

$$\tilde{D}_{f3p-}\left(\frac{\pi}{4}, 0.01\right) - \cos^{'}\left(\frac{\pi}{4}\right) = -0.0000233926273216\,5$$

and is comparable with the error of the two-point approximation using the centered difference formula, and is significantly smaller than the error of the two-point approximation using the backward difference formula. □

One should be extremely cautious when the derivatives are numerically calculated using function values from *current real-time measurements, which are usually corrupted with a measurement noise*, or disturbances of other type. Usually, the measurement errors are much larger than the numerical errors, and a difference approximation of a derivative is an ill-conditioned problem. Therefore, if a measurement signal contains a disturbing random component of significantly faster variability than the original measured variable, this component should be preliminary filtered out from the measurements, before using the signal for the difference derivative approximation.

8.2. Numerical integration

A numerical evaluation of an integral of a nonlinear function defined on a certain interval, or a set (a multiple integral), is a typical task in numerous applications. The simplest (and basic) problem is to calculate an integral of a continuous function $f(x)$ over a closed interval $[a, b]$:

$$I(f; a, b) = \int_a^b f(x)dx. \tag{8.13}$$

Assume a fixed division of the interval $[a, b]$ is given, defined by $n+1$ points $a = x_0 < x_1 < \cdots < x_n = b$, and there are values $f(x_i)$ of the function f given at these points. Using this data, we shall calculate an approximate value of the integral (8.13). The approximation will be of the form

$$Q(f; a, b) = \sum_{i=0}^{n} A_i f(x_i), \tag{8.14}$$

which is called a *quadrature rule* (or simply *a quadrature*), the values x_i are called *nodes of the quadrature*. The difference

$$R(f; a, b) = I(f; a, b) - Q(f; a, b) \tag{8.15}$$

is an *error term* (a truncation error term, a residuum) of the quadrature.

The quadrature rule is said to be *of order p*, if it is exact (i.e., its error term is equal to zero) for all polynomials of order less than p, and there exists at least one polynomial of order p which yields a non-zero quadrature error term. The number p is also called *a degree of precision* or *a degree of exactness* of the quadrature rule.

The quadrature rules depend on the way the coefficients A_i are evaluated. A basic, natural way to derive a quadrature rule is to construct a a polynomial interpolating the values $f(x_i)$ at the quadrature nodes and then derive the quadrature rule as an integral of the interpolating polynomial.

Newton-Cotes quadrature rules

If polynomials used for construction of the quadrature rules are the Lagrange interpolating polynomials defined on equally spaced nodes, then the resulting quadrature rules are called the *Newton-Cotes quadrature rules*. Assume $x_i = a + ih$, $i = 0, \ldots, n$, $a + nh = b$ (i.e., $h = \frac{b-a}{n}$). Applying a change of variables $x = a + th$, $t \in [0, n]$, the Lagrange interpolating polynomial can be written in the form

$$L_n(x) = L_n(a + th) = \sum_{i=0}^{n} f(x_i) \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} = \sum_{i=0}^{n} f(x_i) \prod_{j=0, j\neq i}^{n} \frac{t - j}{i - j}. \tag{8.16}$$

Integrating the right-hand side of this equation, we obtain coefficients of the quadrature rule:

$$A_i = \int_a^b \prod_{j=0,j\neq i}^{n} \frac{x - x_j}{x_i - x_j} dx = h \int_0^n \prod_{j=0,j\neq i}^{n} \frac{t - j}{i - j} dt, \tag{8.17}$$

because, due to the change in variables, $dx = hdt$. The Newton-Cotes quadrature rules are usually written in a slightly different form

$$Q(f; a, b) = \sum_{i=0}^{n} A_i f(x_i) = (b - a) \sum_{i=0}^{n} B_i^{(n)} f(x_i), \tag{8.18}$$

where

$$B_i^{(n)} = \frac{A_i}{b - a} = \frac{1}{n} \int_0^n \prod_{j=0,j\neq i}^{n} \frac{t - j}{i - j} dt. \tag{8.19}$$

The superscript "n" emphasizes that the coefficients defined in this way do not depend on the length $b - a$ of the interval, but they depend on the value of n only. It follows directly from the above definition that $\sum_{i=0}^{n} B_i^{(n)} = 1$.

The second mean-value theorem will be needed for further considerations.

Theorem 8.1. (Second mean-value theorem) *If a function f is continuous over an interval $[a, b]$ and a function g is integrable and non-negative (or non-positive) on $[a, b]$, then there exists a point $\alpha \in (a,\ b)$ such that*

$$\int_a^b f(x)g(x)dx = f(\alpha) \int_a^b g(x)dx. \tag{8.20}$$

Newton-Cotes quadrature rule for *n* = 1. For $n = 1$ the quadrature nodes are $x_0 = a$ and $x_1 = b$, According to (8.19) we obtain the coefficients:

$$\begin{aligned} B_0^{(1)} &= \int_0^1 \frac{t - 1}{0 - 1} dt = \left[t - \frac{1}{2} t^2 \right]_0^1 = \frac{1}{2}, \\ B_1^{(1)} &= \int_0^1 \frac{t - 0}{1 - 0} dt = \left[\frac{1}{2} t^2 \right]_0^1 = \frac{1}{2}, \end{aligned}$$

which results in the well-known *trapezoidal rule*

$$Q(f; a, b) = (b - a) \frac{f(a) + f(b)}{2}, \tag{8.21}$$

illustrated in Fig. 8.1.

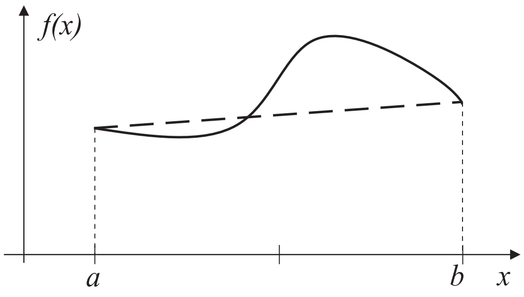


Figure 8.1. A graphical interpretation of the trapezoidal rule

Let us evaluate the error term of this quadrature. Using the formula (5.8), which defines the error term of a polynomial approximation (see Section 5.1), and the second mean-value theorem, we have

$$\begin{aligned} R(f;a,b) &= \int_a^b \frac{f^{(2)}(\alpha(x))}{2!}(x-a)(x-b)dx \\ &= \frac{f^{(2)}(\alpha)}{2!}\int_a^b (x-a)(x-b)dx \\ &= -\frac{(b-a)^3}{12}f^{(2)}(\alpha) = -\frac{1}{12}f^{(2)}(\alpha)h^3. \end{aligned} \tag{8.22}$$

Therefore, we have proved that the trapezoidal rule is of second order (its error term is not zero for polynomials of second order, which have a non-zero (and constant) second order derivative).

Newton-Cotes quadrature for *n* = 2. For $n = 2$ the quadrature nodes are $x_0 = a$, $x_1 = \frac{a+b}{2}$ and $x_2 = b$, according to (8.19) we get the coefficients:

$$\begin{aligned} B_0^{(2)} &= \frac{1}{2}\int_0^2 \frac{(t-1)(t-2}{(0-1)(0-2)}dt = \frac{1}{6}, \\ B_1^{(2)} &= \frac{1}{2}\int_0^2 \frac{(t-0)(t-2)}{(1-0)(1-2)}dt = \frac{4}{6}, \\ B_2^{(2)} &= B_0^{(2)} = \frac{1}{6}, \end{aligned}$$

where the symmetry of the quadratures has been used (i.e., the general property $B_i^{(n)} = B_{n-i}^{(n)}$). The obtained coefficients define the well-known *Simpson's rule*:

$$Q(f;a,b) = (b-a)\frac{f(a) + 4f(\frac{a+b}{2}) + f(b)}{6}, \tag{8.23}$$

where $h = \frac{b-a}{2}$. The Simpson's rule is illustrated in Fig. 8.2.

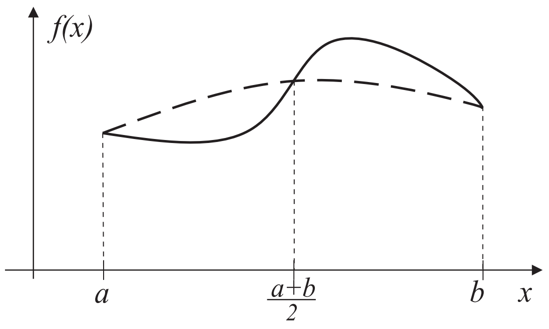


Figure 8.2. A graphical interpretation of the Simpson's rule

It can be proved that the Newton-Cotes quadrature rules using n nodes are of order $p = n+1$ for odd values of n, and of order $p = n+2$ for even values of n. Moreover, it can be shown that the error terms of these quadratures are described by

$$R(f;a,b) = R^{(n)}(f) = -cf^{(p)}(\alpha)h^{p+1}, \tag{8.24}$$

where $\alpha \in (a,\ b)$ (we have shown it earlier for $n=1$). Remember that $h = \frac{a-b}{n}$, i.e., for $n = 1$: $h = a - b$, for $n = 2$: $h = \frac{a-b}{2}$, etc. Values of the coefficients $B_i^{(n)}$, of the orders p and of the constants c of the error terms of the Newton-Cotes quadratures for $n = 1, 2, ..., 6$ are given in Table 8.1. For values of n larger than 6 the Newton-Cotes quadrature rules are not usable, because negative coefficients occur (except for $n = 9$).

Composite Newton-Cotes quadrature rules

It is not sufficient to use a single quadrature rule for a numerical calculation of an integral over a closed interval of any length, with a small error. Generally, the interval is divided into many smaller subintervals, and on each subinterval a single (simple) quadrature rule of low order is used, usually the trapezoidal or the Simpson's rule. In this way, *the composite quadrature rules* are generated.

Table 8.1. Coefficients and parameters of Newton-Cotes quadrature rules

n	$B_i^{(n)}$	p	c_n	quadrature
1	$\frac{1}{2}, \frac{1}{2}$	2	$\frac{1}{12}$	trapezoidal
2	$\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$	4	$\frac{1}{90}$	Simpson's
3	$\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}$	4	$\frac{3}{80}$	Simpson's $\frac{3}{8}$
4	$\frac{7}{90}, \frac{32}{90}, \frac{12}{90}, \frac{32}{90}, \frac{7}{90}$	6	$\frac{8}{945}$	Boole's
5	$\frac{19}{288}, \frac{75}{288}, \frac{50}{288}, \frac{50}{288}, \frac{75}{288}, \frac{19}{288}$	6	$\frac{275}{12096}$	Bode's
6	$\frac{41}{440}, \frac{216}{840}, \frac{27}{840}, \frac{272}{840}, \frac{27}{840}, \frac{216}{840}, \frac{41}{440}$	8	$\frac{9}{1400}$	Weddle's

The composite trapezoidal rule. For $n=2$, assume that the points $x_i = a+ih$ are chosen over the interval $[a,b]$, dividing it into N equal subintervals $[x_i, x_{i+1}]$, $i = 0,1,...,N-1$, each of the length $h = \frac{b-a}{N}$. Using for every $i = 0,1,...,N-1$ the simple trapezoidal rule given by (8.21) and the formula (8.22) for the errors of the approximation, we get

$$\begin{aligned}\int\limits_a^b f(x)dx &= \sum_{i=0}^{N-1} \int\limits_{x_i}^{x_{i+1}} f(x)dx = \sum_{i=0}^{N-1} \left[\frac{h}{2}\left(f(x_i)+f(x_{i+1})\right) - \frac{1}{12} f^{(2)}(\alpha_i)h^3\right] \\ &= \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{N-1} f(a+ih) + f(b)\right) - \frac{h^3}{12}\sum_{i=0}^{N-1} f^{(2)}(\alpha_i),\end{aligned}$$

where $\alpha_i \in (x_i, x_{i+1})$. We obtained in this way a composite quadrature rule $Q_N(f;a,b) = T_N(f;a,b)$, which is called *the composite trapezoidal rule*,

$$T_N(f;a,b) = \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{N-1} f(a+ih) + f(b)\right). \tag{8.25}$$

Assuming that second order derivative of the function f is continuous over $[a,b]$, the error of this composite quadrature rule can be expressed as follows:

$$R_{TN}(f;a,b) = -\frac{h^3 N}{12} \cdot \frac{1}{N}\sum_{i=0}^{N-1} f^{(2)}(\alpha_i) = -N \cdot \frac{h^3}{12} f^{(2)}(\alpha_s).$$

Taking into account the relation between the length of the interval $[a,b]$ and the value of N, we can express the error in the form

$$R_{TN}(f;a,b) = -(a-b)\frac{h^2}{12} f^{(2)}(\alpha_s), \tag{8.26}$$

where the last term $f^{(2)}(\alpha_s)$ can be treated as as a mean value of the second order derivative of f over $[a, b]$ (the larger N, the more accurate this mean value).

The composite Simpson's quadrature rule. For $n = 3$, let us choose equally spaced quadrature nodes $x_i, i = 0, 1, 2, ..., N$, which divide the interval $[a, b]$ into N equal subintervals of a length $h = \frac{a-b}{N}$, assume also that N is an even number. Over each subinterval of the length $2h$, i.e., $[a, a + 2h]$, $[a + 2h, a + 4h]$, etc., the simple Simpson's rule (8.23) will be applied. Using also the formula (8.24) for the order $p = n+1 = 4$ with $c = \frac{1}{90}$, we obtain for every $k = 1, 2, ..., \frac{N}{2}$ (k indexes the subintervals $[x_{2(k-1)}, x_{2k}]$ of length $2h$):

$$\begin{aligned}
\int_a^b f(x)\,dx &= \sum_{k=1}^{N/2}\left[\frac{2h}{6}\left(f(x_{2k-2}) + 4f(x_{2k-1}) + f(x_{2k})\right) - \frac{1}{90}f^{(4)}(\alpha_k)h^5\right] \\
&= \frac{h}{3}\Big(\, f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots \\
&\qquad \cdots + 4f(x_{N-1}) + f(b)\,\Big) - \sum_{k=1}^{N/2}\frac{1}{90}f^{(4)}(\alpha_k)h^5 \\
&= \frac{h}{3}\left(f(a) + 4\sum_{k=1}^{N/2} f(x_{2k-1}) + 2\sum_{k=1}^{N/2-1} f(x_{2k}) + f(b)\right) - \frac{h^5}{90}\sum_{k=1}^{N/2} f^{(4)}(\alpha_k) \\
&= S_N(f; a, b) + R_{SN}(f; a, b), \qquad (8.27)
\end{aligned}$$

where $\alpha_k \in [x_{2k-1}, x_{2k}],\ k = 1, ..., N/2$ and the composite error $R_{SN}(f; a, b)$ is defined by the last sum. Similarly as in the previous case, assuming that the fourth order derivative of the function f is continuous over $[a, b]$, the error $R_{SN}(f; a, b)$ of this composite quadrature can be expressed as follows:

$$\begin{aligned}
R_{SN}(f; a, b) &= -\frac{h^5 N}{180}\cdot\frac{1}{N/2}\sum_{k=1}^{N/2} f^{(4)}(\alpha_k) \\
&= -N\cdot\frac{h^5}{180}f^{(4)}(\alpha_s) = -(a-b)\frac{h^4}{180}f^{(4)}(\alpha_s). \qquad (8.28)
\end{aligned}$$

When comparing the formulae (8.26) and (8.28), it can be easily seen that the composite Simpson's quadrature is more accurate: using the same number of points (thus the same step h) the error of the composite Simpson's quadrature is $\frac{15}{h^2}$ times smaller. For example: for $h = 0.1$, $\frac{15}{h^2} = 1500$. On the other hand, when assuming the same error we must use the composite trapezoidal rule with a much smaller step h.

In genaral, rather a relative than an absolute accuracy of a numerical calculation of an integral is required. This means that for an assumed accuracy parameter ϵ, it is required that

$$\frac{|Q(f;a,b) - I(f;a,b)|}{|I(f;a,b)|} < \epsilon. \tag{8.29}$$

To evaluate the accuracy of the quadrature, an estimation of its truncation error can also be used. This enables an estimate of the number of points N at which the integrand values need to be calculated – but requires an estimation of the mean value of the second order (for the trapezoidal rule) or fourth order (for the Simpson's rule) derivative of the function over the interval $[a, b]$ (for simplicity or more reliability a maximal value of the derivative can also be estimated). However, this may be difficult or lead to conservative estimations. Therefore, a practical approach is to calculate a series of composite quadratures for an increasing number of points, i.e., $Q_{N_1}(f;a,b)$, $Q_{N_2}(f;a,b)$, $Q_{N_3}(f;a,b)$, etc., for $N_1 < N_2 < N_3 < \cdots$, where $Q_{N_i}(f;a,b)$ denotes a composite quadrature using N_i points. The calculations are terminated when

$$\frac{|Q_{N_{j-1}}(f;a,b) - Q_{N_j}(f;a,b)|}{|Q_{N_j}(f;a,b)|} < \epsilon, \tag{8.30}$$

and, for more confidence, a value of ϵ smaller than that assumed for (8.29) can be applied. It is important that subsequent values of N_j are chosen in a way assuring that *all previously calculated values of the integrated function are utilized.* We shall show how this can be obtained for the composite trapezoidal and Simpson's quadrature rules.

If we take $N_{j+1} = 2N_j$, then we have $h_{N_{j+1}} = h_{N_j}/2$ and the following *recurrent formula for the composite trapezoidal rule* is obtained:

$$T_{N_{j+1}}(f;a,b) = \frac{1}{2}T_{N_j}(f;a,b) + h_{N_{j+1}} \sum_{i=1}^{N_{j+1}/2} f(a + (2i-1)h_{N_{j+1}}), \tag{8.31}$$

where $h_{N_j} = \frac{b-a}{N_j}$.

To derive an analogous *recurrent formula for the composite Simpson's rule* (8.27), let us first reformulate the composite Simpson's formula into the following equivalent form

$$\begin{aligned} S_N(f;a,b) &= \frac{h}{3}\left(f(a) + 2\sum_{k=1}^{N/2} f(x_{2k-1}) + 2\sum_{k=1}^{N/2-1} f(x_{2k}) + f(b)\right) + \\ &\qquad + 2\frac{h}{3}\sum_{k=1}^{N/2} f(x_{2k-1}) \\ &= S_N^1(f;a,b) + S_N^2(f;a,b), \end{aligned} \tag{8.32}$$

where the term $S_N^2(f; a, b)$ is defined by the last sum (in the second line of (8.32)). Now, if we assume $N_{j+1} = 2N_j$, i.e., $h_{N_{j+1}} = h_{N_j}/2$, then all new points added due to division of the step h by 2 constitute the set of all nodes with odd indices of the new quadrature rule and all nodes of the previous quadrature rule become all nodes with even indices of the new quadrature rule, therefore

$$\begin{aligned} S_{N_{j+1}}(f; a, b) &= \left[\frac{1}{2}S_{N_j}^1(f; a, b) + 2\frac{h_{N_{j+1}}}{3}\sum_{k=1}^{N_{j+1}/2} f(x_{2k-1})\right] + \\ &\qquad + 2\frac{h_{N_{j+1}}}{3}\sum_{k=1}^{N_{j+1}/2} f(x_{2k-1}) \\ &= S_{N_{j+1}}^1(f; a, b) + S_{N_{j+1}}^2(f; a, b). \end{aligned} \tag{8.33}$$

Example 8.3. We shall calculate, using MATLAB, a series of integral approximations using the composite trapezoidal and Simpson's rules, for the definite integral of the function $f(x) = \sin(x)$ over the interval $[0, \pi/2]$. As it can be easily calculated, the accurate value of this integral is 1. For the composite trapezoidal rule, the values of subsequent integral approximations $T_N(\sin; 0, \pi/2)$, together with the quadrature errors $R_{TN}(\sin; 0, \pi/2)$ and the values of error estimates (8.30) and the corresponding values of $N = 2, 4, 8, ...$, are as follows:

N	$T_N(\sin; 0, \pi/2)$	$R_{TN}(\sin; 0, \pi/2)$	$\lvert(T_{N/2} - T_N)/T_N\rvert$
2	0.948059448968520	0.051940551031480	
4	0.987115800972775	0.012884199027225	0.039566129896580
8	0.996785171886170	0.003214828113830	0.009700556535264
16	0.999196680485072	0.000803319514928	0.002413447368272
32	0.999799194320019	0.000200805679981	0.000602634847447
64	0.999949800092101	0.000050199907899	0.000150613332858
128	0.999987450117527	0.000012549882473	0.000037650497935
256	0.999996862535288	0.000003137464712	0.000009412447293
512	0.999999215634191	0.000000784365809	0.000002353100748
1024	0.999999803908570	0.000000196091430	0.000000588274494
$\vdots$	$\vdots$	$\vdots$	$\vdots$
2^{20}	0.999999999999747	0.000000000000253	0.000000000000535

The corresponding values of numerical integral approximations obtained using the composite Simpson's rule $S_N(\sin; 0, \pi/2)$, together with the quadrature errors

$R_{SN}(\sin;0,\pi/2)$ and with the values of the error estimates (8.30) and the corresponding values of $N = 2, 4, 8, ..., 1024$, are as follows:

N	$S_N(\sin;0,\pi/2)$	$R_{SN}(\sin;0,\pi/2)$	$\lvert(S_{N/2} - S_N)/S_N\rvert$
2	1.002279877492210	−0.002279877492210	0
4	1.000134584974194	−0.000134584974194	0.002145003832731
8	1.000008295523968	−0.000008295523968	0.000126288402598
16	1.000000516684707	−0.000000516684707	0.000007778835242
32	1.000000032265001	−0.000000032265001	0.000000484419690
64	1.000000002016129	−0.000000002016129	0.000000030248872
128	1.000000000126001	−0.000000000126001	0.000000001890127
256	1.000000000007875	−0.000000000007875	0.000000000118126
512	1.000000000000492	−0.000000000000492	0.000000000007383
1024	1.000000000000031	−0.000000000000031	0.000000000000462

It should be noted that the results obtained with the composite Simpson's rule are more accurate, while the number of computations is comparable with that for the composite trapezoidal rule, because using the recurrent formulae (8.31) and (8.33), all values of the integrand are calculated only once at each point – the use of these values is only different.

Note also that the estimations of the quadrature errors, calculated according to (8.30), are for the trapezoidal rule comparable with the quadrature errors, while for the Simpson's rule are more conservative, approximately one order of magnitude larger.

One can find below the text of a simple program (in MATLAB) calculating a sequence of integral approximations (for the function from the example) using the composite trapezoidal rule, according to the recurrent formula (8.31):

```
%script SeqCompTrapQuad
n=10;    %number of subsequent composite quadratures
k=1;N=2;h=pi/4;
T=zeros(n,1);
T(1)=0.5*h*(sin(0)+2*sin(pi/4)+sin(pi/2));
for k=2:n
    h=h*0.5;
    N=2*N;
    T(k)=0.5*T(k-1);
    for i=1:N/2
        T(k)=T(k)+h*sin((2*i-1)*h);
    end
end
```

□

The composite Newton-Cotes quadrature rules, as presented in this Section, use equally spaced quadrature nodes over the interval of integration. This approach occurs to be not effective in cases when variability of the integrand is significantly different over different parts of the interval. Therefore, *adaptive quadrature rules* are more effective, which use not equally spaced quadrature nodes. *An adaptive Simpson's rule* is popular, which applies a successive, adaptive division of the integration interval into subintervals and an independent testing of the quadrature accuracy on the subintervals. Only subintervals over which the quadrature errors are too large are further divided. The adaptive Simpson's rule is implemented in the MATLAB procedure "`quad`" for a numerical calculation of definite integrals. For details of the adaptive Simpson's rule, the reader is referred to [2].

Problems

1. Evaluate an estimate of a relative error of the method which approximates the first order derivative using the centered difference formula. Evaluate an optimal step-size for this approximation minimizing the estimate of the overall absolute error (error of the method and numerical error). Assume representation errors of the function values as it was done when deriving the error (8.8) for the backward difference in Section 8.1.
2. Derive the formula (8.12) as a derivative of the interpolating Newton's polynomial.
3. Derive the formula for the first order derivative analogous to (8.12), but formulating the interpolating polynomial for the interpolation nodes $x-h$, x, $x+h$.
4. Applying programs (written in MATLAB) for a recurrent calculation of integral approximations using the composite trapezoidal and Simpson's rules, calculate a sequence of 10 integral approximations (starting from $N = 2$) for the function $1 - e^{-x}\cos(4x)$, over the interval $[0, \pi/4]$, taking every subsequent step as the previous one divided by 2.

Bibliography

[1] K.E. Atkinson: An Introduction to Numerical Analysis. John Wiley & Sons, New York 1989.

[2] J.H. Mathews, K.D. Fink: Numerical Methods Using Matlab (4th ed.). Prentice Hall, 2004.

[3] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery: Numerical Recipes, Third Edition. Cambridge Univ. Press, Cambridge 2007 (electronic version also available).

[4] S. Rosloniec: Fundamental Numerical Methods for Electrical Engineering. Springer, Berlin and Heidelberg, 2008.

[5] J. Stoer, R. Bulirsch: Introduction to Numerical Analysis (3rd ed.). Springer, 2002.

[6] D.S. Watkins: Fundamentals of Matrix Computations. J. Wiley & Sons, Hoboken, New Jersey, 2010.