18. Scientific computing

This title refers to a fast developing interdisciplinary area between mathematics, computers and applications. The subject is also often called as Computational Science and Engineering. Its aim is the efficient use of computer algorithms to solve engineering and scientific problems. One can say with a certain simplification that our subject is related to numerical mathematics, software engineering, computer graphics and applications. Here we can deal only with some basic elements of the subject such as the fundamentals of the *floating point computer arithmetic*, *error analysis*, the basic numerical methods of *linear algebra* and related mathematical software.

18.1. Floating point arithmetic and error analysis

18.1.1. Classical error analysis

Let x be the exact value and let a be an approximation of x ($a \approx x$). The error of the approximation a is defined by the formula $\Delta a = x - a$ (sometimes with opposite sign). The quantity $\delta a \geq 0$ is called an ***(absolute) error (bound)*** of approximation a, if $|x - a| = |\Delta a| \leq \delta a$. For example, the error of the approximation $\sqrt{2} \approx 1.41$ is at most 0.01. In other words, the error bound of the approximation is 0.01. The quantities x and a (and accordingly Δa and δa) may be vectors or matrices. In such cases the absolute value and relation operators must be understood componentwise. We also measure the error by using matrix and vector norms. In such cases, the quantity $\delta a \in \mathbb{R}$ is an error bound, if the inequality $\|\Delta a\| \leq \delta a$ holds.

The absolute error bound can be irrelevant in many cases. For example, an approximation with error bound 0.05 has no value in estimating a quantity of order 0.001. The goodness of an approximation is measured by the ***relative error*** $\delta a / |x|$ ($\delta a / \|x\|$ for vectors and matrices), which compares the error bound to the approximated quantity. Since the exact value is generally unknown, we use the approximate relative error $\delta a / |a|$ ($\delta a / \|a\|$). The committed error is proportional to the quantity $(\delta a)^2$, which can be neglected, if the absolute value (norm) of x and a is much greater than $(\delta a)^2$. The relative error is often expressed in percentages.

In practice, the (absolute) error bound is used as a substitute for the generally unknown true error.

In the *classical error analysis* we assume input data with given error bounds, exact computations (operations) and seek for the error bound of the final result. Let x and y be exact values with approximations a and b, respectively. Assume that the absolute error bounds of approximations a and b are δa and δb, respectively. Using the classical error analysis approach we obtain the following error bounds for the four basic arithmetic operations:

$$\begin{aligned}
\delta(a+b) &= \delta a + \delta b, & \frac{\delta(a+b)}{|a+b|} &= \max\left\{\frac{\delta a}{|a|}, \frac{\delta b}{|b|}\right\} \quad (ab>0)\,,\\
\delta(a-b) &= \delta a + \delta b, & \frac{\delta(a-b)}{|a-b|} &= \frac{\delta a + \delta b}{|a-b|} \quad (ab>0)\,,\\
\delta(ab) &\approx |a|\,\delta b + |b|\,\delta a & \frac{\delta(ab)}{|ab|} &\approx \frac{\delta a}{|a|} + \frac{\delta b}{|b|} \quad (ab\neq 0)\,,\\
\delta(a/b) &\approx \frac{|a|\,\delta b + |b|\,\delta a}{|b|^2} & \frac{\delta(a/b)}{|a/b|} &\approx \frac{\delta a}{|a|} + \frac{\delta b}{|b|} \quad (ab\neq 0)\,.
\end{aligned}$$

We can see that the division with a number near to 0 can make the absolute error arbitrarily big. Similarly, if the result of subtraction is near to 0, then its relative error can become arbitrarily big. One has to avoid these cases. Especially the subtraction operation can be quite dangerous.

Example 18.1 Calculate the quantity $\sqrt{1996} - \sqrt{1995}$ with approximations $\sqrt{1996} \approx 44.67$ and $\sqrt{1995} \approx 44.66$ whose common absolute and relative error bounds are 0.01 and 0.022%, respectively. One obtains the approximate value $\sqrt{1996} - \sqrt{1995} \approx 0.01$, whose relative error bound is

$$\frac{0.01+0.01}{0.01} = 2,$$

that is 200%. The true relative error is about 10.66%. Yet it is too big, since it is approximately 5×10^2 times bigger than the relative error of the initial data. We can avoid the subtraction operation by using the following trick

$$\sqrt{1996} - \sqrt{1995} = \frac{1996-1995}{\sqrt{1996}+\sqrt{1995}} = \frac{1}{\sqrt{1996}+\sqrt{1995}} \approx \frac{1}{89.33} \approx 0.01119.$$

Here the nominator is exact, while the absolute error of the denominator is 0.02. Hence the relative error (bound) of the quotient is about $0.02/89.33 \approx 0.00022 = 0.022\%$. The latter result is in agreement with the relative error of the initial data and it is substantially smaller than the one obtained with direct subtraction operation.

The first order error terms of twice differentiable functions can be obtained by their first order *Taylor polynomial*:

$$\begin{aligned}
\delta(f(a)) &\approx \left|f'(a)\right|\delta a, \quad f:\mathbb{R}\to\mathbb{R},\\
\delta(f(a)) &\approx \sum_{i=1}^{n}\left|\frac{\partial f(a)}{\partial x_i}\right|\delta a_i, \quad f:\mathbb{R}^n\to\mathbb{R}.
\end{aligned}$$

The numerical sensitivity of functions at a given point is characterized by the ***condition number***, which is the ratio of the relative errors of approximate function value and the input

$$\begin{tikzcd}
x \arrow[r] \arrow[d] \arrow[dr, dashed] & y = f(x) \arrow[d] \\
x + \Delta x \arrow[r] & \widehat{y} = f(x + \Delta x)
\end{tikzcd}$$

Figure 18.1. Forward and backward error.

data (the *Jacobian matrix* of functions $F : \mathbb{R}^n \to \mathbb{R}^m$ is denoted by $F'(a)$ at the point $a \in \mathbb{R}^n$):

$$c(f,a) = \frac{|f'(a)|\,|a|}{|f(a)|}, \quad f : \mathbb{R} \to \mathbb{R},$$
$$c(F,a) = \frac{\|a\|\,\|F'(a)\|}{\|F(a)\|}, \quad F : \mathbb{R}^n \to \mathbb{R}^m.$$

We can consider the condition number as the magnification number of the input relative error. Therefore the functions is considered ***numerically stable*** (or ***well-conditioned***) at the point a, if $c(f,a)$ is „small”. Otherwise f is considered as ***numerically unstable*** (***ill-conditioned***). The condition number depends on the point a. A function can be well-conditioned at point a, while it is ill-conditioned at point b. The term „small” is relative. It depends on the problem, the computer and the required precision.

The condition number of matrices can be defined as the upper bound of a function condition number. Let us define the mapping $F : \mathbb{R}^n \to \mathbb{R}^n$ by the solution of the equation $Ay = x$ ($A \in \mathbb{R}^{n\times n}$, $\det(A) \neq 0$), that is, let $F(x) = A^{-1}x$. Then $F' \equiv A^{-1}$ and

$$c(F,a) = \frac{\|a\|\,\|A^{-1}\|}{\|A^{-1}a\|} = \frac{\|Ay\|\,\|A^{-1}\|}{\|y\|} \le \|A\|\,\|A^{-1}\| \quad (Ay = a)\,.$$

The upper bound of the right side is called the ***condition number of the matrix*** A. This bound is sharp, since there exists a vector $a \in \mathbb{R}^n$ such that $c(F,a) = \|A\|\,\|A^{-1}\|$.

18.1.2. Forward and backward errors

Let us investigate the calculation of the function value $f(x)$. If we calculate the approximation $\hat{y}$ instead of the exact value $y = f(x)$, then the forward error $\Delta y = \hat{y} - y$. If for a value $x + \Delta x$ the equality $\hat{y} = f(x + \Delta x)$ holds, that is, $\hat{y}$ is the exact function value of the perturbed input data $\hat{x} = x + \Delta x$, then Δx is called the ***backward error***. The connection of the two concepts is shown on the Figure 18.1.

The continuous line shows exact value, while the dashed one indicates computed value. The analysis of the backward error is called the ***backward error analysis*** . If there exist more than one backward error, then the estimation of the smallest one is the most important.

An algorithm for computing the value $y = f(x)$ is called ***backward stable***, if for any x it gives a computed value $\hat{y}$ with small backward error Δx. Again, the term „small” is relative to the problem environment.

The connection of the forward and backward errors is described by the approximate thumb rule

$$\frac{\delta \hat{y}}{|y|} \lessapprox c(f, x) \frac{\delta \hat{x}}{|x|}, \tag{18.1}$$

which means that

relatíve forward error ≤ condition number × relative backward error.

This inequality indicates that the computed solution of an ill-conditioned problem may have a big relative forward error. An algorithm is said to be ***forward stable*** if the forward error is small. A forward stable method is not necessarily backward stable. If the forward error and the condition number are small, then the algorithm is forward stable.

Example 18.2 Consider the function $f(x) = \log x$ the condition number of which is $c(f, x) = c(x) = 1/\left|\log x\right|$. For $x \approx 1$ the condition number $c(f, x)$ is big. Therefore the relative forward error is big for $x \approx 1$.

18.1.3. Rounding errors and floating point arithmetic

The classical error analysis investigates only the effects of the input data errors and assumes exact arithmetic operations. The digital computers however are representing the numbers with a finite number of digits, the arithmetic computations are carried out on the elements of a finite set F of such numbers and the results of operations belong to F. Hence the computer representation of the numbers may add further errors to the input data and the results of arithmetic operations may also be subject to further rounding. If the result of operation belongs to F, then we have the exact result. Otherwise we have three cases:

(i) rounding to representable (nonzero) number;

(ii) underflow (rounding to 0);

(iii) overflow (in case of results whose moduli too large).

The most of the scientific-engineering calculations are done in *floating point arithmetic* whose generally accepted model is the following:

Definition 18.1 *The set of floating point numbers is given by*

$$F(\beta, t, L, U) = \\ = \left\{\pm m \times \beta^e \mid \tfrac{1}{\beta} \le m < 1,\ m = 0.d_1 d_2 \dots d_t,\ L \le e \le U\right\} \cup \{0\},$$

where
- β is the base (or radix) of the number system,
- m is the mantissa in the number system with base β,
- e is the exponent,
- t is the length of mantissa (the precision of arithmetic),
- L is the smallest exponent (underflow exponent),
- U is the biggest exponent (overflow exponent).

The parameters of the three most often used number systems are indicated in the following table

Name	β	Machines
binary	2	most computer
decimal	10	most calculators
hexadecimal	16	IBM mainframe computers

The mantissa can be written in the form

$$m = 0.d_1d_2\dots d_t = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t}. \tag{18.2}$$

We can observe that condition $1/\beta \le m < 1$ implies the inequality $1 \le d_1 \le \beta - 1$ for the first digit d_1. The remaining digits must satisfy $0 \le d_i \le \beta - 1$ $(i = 2, \dots, t)$. Such arithmetic systems are called ***normalized***. The zero digit and the dot is not represented. If $\beta = 2$, then the first digit is 1, which is also unrepresented. Using the representation (18.2) we can give the set $F = F(\beta, t, L, U)$ in the form

$$F = \left\{ \pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t} \right) \beta^e \mid L \le e \le U \right\} \cup \{0\}, \tag{18.3}$$

where $0 \le d_i \le \beta - 1$ $(i = 1, \dots, t)$ and $1 \le d_1$.

Example 18.3 The set $F(2, 3, -1, 2)$ contains 33 elements and its positive elements are given by

$$\left\{ \frac{1}{4}, \frac{5}{16}, \frac{6}{16}, \frac{7}{16}, \frac{1}{2}, \frac{5}{8}, \frac{6}{8}, \frac{7}{8}, 1, \frac{10}{8}, \frac{12}{8}, \frac{14}{8}, 2, \frac{20}{8}, 3, \frac{28}{8} \right\}.$$

The elements of F are not equally distributed on the real line. The distance of two consecutive numbers in $[1/\beta, 1] \cap F$ is β^{-t}. Since the elements of F are of the form $\pm m \times \beta^e$, the distance of two consecutive numbers in F is changing with the exponent. The maximum distance of two consecutive floating point numbers is β^{U-t}, while the minimum distance is β^{L-t}.

For the mantissa we have $m \in [1/\beta, 1 - 1/\beta^t]$, since

$$\frac{1}{\beta} \le m = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t} \le \frac{\beta - 1}{\beta} + \frac{\beta - 1}{\beta^2} + \cdots + \frac{\beta - 1}{\beta^t} = 1 - \frac{1}{\beta^t}.$$

Using this observation we can easily prove the following result on the range of floating point numbers.

Theorem 18.2 *If $a \in F$, $a \ne 0$, then $M_L \le |a| \le M_U$, where*

$$M_L = \beta^{L-1}, \quad M_U = \beta^U (1 - \beta^{-t}).$$

Let $a, b \in F$ and denote $\square$ any of the four arithmetic operations $(+, -, *, /)$. The following cases are possible:

(1) $a \square b \in F$ (exact result),

(2) $|a \square b| > M_U$ (arithmetic overflow),

(3) $0 < |a \square b| < M_L$ (arithmetic underflow),

(4) $a \square b \notin F$, $M_L < |a \square b| < M_U$ (not representable result).

In the last two cases the *floating point arithmetic* is rounding the result $a\Box b$ to the nearest floating point number in F. If two consecutive floating point numbers are equally distant from $a\Box b$, then we generally round to the greater number. For example, in a five digit decimal arithmetic, the number 2.6457513 is rounded to the number 2.6458.

Let $G = [-M_U, M_U]$. It is clear that $F \subset G$. Let $x \in G$. The $fl(x)$ denotes an element of F nearest to x. The mapping $x \to fl(x)$ is called rounding. The quantity $|x - fl(x)|$ is called the rounding error. If $fl(x) = 1$, then the rounding error is at most $\beta^{1-t}/2$. The quantity $u = \beta^{1-t}/2$ is called the ***unit roundoff***. The quantity u is the relative error bound of $fl(x)$.

Theorem 18.3 *If $x \in G$, then*

$$fl(x) = x(1+\varepsilon), \quad |\varepsilon| \le u.$$

Proof. Without loss of generality we can assume that $x > 0$. Let $m_1\beta^e$, $m_2\beta^e \in F$ be two consecutive numbers such that

$$m_1\beta^e \le x \le m_2\beta^e.$$

Either $1/\beta \le m_1 < m_2 \le 1 - \beta^{-t}$ or $1 - \beta^{-t} = m_1 < m_2 = 1$ holds. Since $m_2 - m_1 = \beta^{-t}$ holds in both cases, we have

$$|fl(x) - x| \le \frac{|m_2 - m_1|}{2}\beta^e = \frac{\beta^{e-t}}{2}$$

either $fl(x) = m_1\beta^e$ or $fl(x) = m_2\beta^e$. It follows that

$$\frac{|fl(x) - x|}{|x|} \le \frac{|fl(x) - x|}{m_1\beta^e} \le \frac{\beta^{e-t}}{2m_1\beta^e} = \frac{\beta^{-t}}{2m_1} \le \frac{1}{2}\beta^{1-t} = u.$$

Hence $fl(x) - x = \lambda x u$, where $|\lambda| \le 1$. A simple arrangement yields

$$fl(x) = x(1+\varepsilon) \quad (\varepsilon = \lambda u)$$

Since $|\varepsilon| \le u$, we proved the claim. ■

Thus we proved that the relative error of the rounding is bounded in floating point arithmetic and the bound is the unit roundoff u.

Another quantity used to measure the rounding errors is the so called the ***machine epsilon*** $\epsilon_M = 2u = \beta^{1-t}$ ($\epsilon_M = 2u$). The number ϵ_M is the distance of 1 and its nearest neighbor greater than 1. The following algorithm determines ϵ_M in the case of binary base.

```
Machine-Epsilon
1 $x \leftarrow 1$
2 while $1 + x > 1$
3       do $x \leftarrow x/2$
4 $\epsilon_M \leftarrow 2x$
5 return $\epsilon_M$
```

In the MATLAB system $\epsilon_M \approx 2.2204 \times 10^{-16}$.

For the results of floating point arithmetic operations we assume the following (standard model):

$$fl(a\Box b) = (a\Box b)(1+\varepsilon), \quad |\varepsilon| \le u \quad (a, b \in F). \tag{18.4}$$

The IEEE arithmetic standard satisfies this assumption. It is an important consequence of the assumption that for $a\square b \neq 0$ the relative error of arithmetic operations satisfies

$$\frac{|fl(a\square b)-(a\square b)|}{|a\square b|} \leq u.$$

Hence the relative error of the floating point arithmetic operations is small.

There exist computer floating point arithmetics that do not comply with the standard model (18.4). The usual reason for this is that the arithmetic lacks a guard digit in subtraction. For simplicity we investigate the subtraction $1-0.111$ in a three digit binary arithmetic. In the first step we equate the exponents:

$$\begin{array}{ccccccccc} & 2 & \times & 0 & . & 1 & 0 & 0 & \\ - & 2 & \times & 0 & . & 0 & 1 & 1 & 1 \end{array}.$$

If the computation is done with four digits, the result is the following

$$\begin{array}{ccccccccc} & 2^1 & \times & 0 & . & 1 & 0 & 0 & \\ - & 2^1 & \times & 0 & . & 0 & 1 & 1 & 1 \\ \hline & 2^1 & \times & 0 & . & 0 & 0 & 0 & 1 \end{array},$$

from which the normalized result is $2^{-2}\times 0.100$. Observe that the subtracted number is unnormalized. The temporary fourth digit of the mantissa is called a guard digit. Without a guard digit the computations are the following:

$$\begin{array}{cccccccc} & 2^1 & \times & 0 & . & 1 & 0 & 0 \\ - & 2^1 & \times & 0 & . & 0 & 1 & 1 \\ \hline & 2^1 & \times & 0 & . & 0 & 0 & 1 \end{array}.$$

Hence the normalized result is $2^{-1}\times 0.100$ with a relative error of 100%. Several CRAY computers and pocket calculators lack guard digits.

Without the guard digit the floating point arithmetic operations satisfy only the weaker conditions

$$fl(x\pm y) = x(1+\alpha)\pm y(1+\beta), \quad |\alpha|,|\beta|\leq u, \tag{18.5}$$

$$fl(x\square y) = (x\square y)(1+\delta), \quad |\delta|\leq u, \quad \square = *,/. \tag{18.6}$$

Assume that we have a guard digit and the arithmetic complies with standard model (18.4). Introduce the following notations:

$$|z| = [|z_1|,\dots,|z_n|]^T \quad (z\in\mathbb{R}^n), \tag{18.7}$$

$$|A| = \left[\left|a_{ij}\right|\right]_{i,j=1}^{m,n} \quad \left(A\in\mathbb{R}^{m\times n}\right), \tag{18.8}$$

$$A\leq B \Leftrightarrow a_{ij}\leq b_{ij} \quad \left(A,B\in\mathbb{R}^{m\times n}\right). \tag{18.9}$$

The following results hold:

$$\left|fl\left(x^Ty\right)-x^Ty\right| \leq 1.01nu\,|x|^T\,|y| \quad (nu\leq 0.01), \tag{18.10}$$

$$fl(\alpha A) = \alpha A + E \quad (|E| \le u\,|\alpha A|)\,, \tag{18.11}$$

$$fl(A + B) = (A + B) + E \quad (|E| \le u\,|A + B|)\,, \tag{18.12}$$

$$fl(AB) = AB + E \quad \left(|E| \le nu\,|A|\,|B| + O\left(u^2\right)\right), \tag{18.13}$$

where E denotes the error (matrix) of the actual operation.

The standard *floating point arithmetics* have many special properties. It is an important property that the addition is not associative because of the rounding.

Example 18.4 If $a = 1, b = c = 3 \times 10^{-16}$, then using MATLAB and AT386 type PC we obtain

$$1.000000000000000e + 000 = (a + b) + c \neq a + (b + c) = 1.000000000000001e + 000.$$

We can have a similar result on Pentium1 machine with the choice $b = c = 1.15 \times 10^{-16}$.

The example also indicates that for different (numerical) processors may produce different computational results for the same calculations. The commutativity can also be lost in addition. Consider the computation of the sum $\sum_{i=1}^{n} x_i$. The usual algorithm is the recursive summation.

```
Recursive-Summation($n$, $x$)
1  $s \leftarrow 0$
2  for $i \leftarrow 1$ to $n$
3       do $s \leftarrow s + x_i$
4  return $s$
```

Example 18.5 Compute the sum

$$s_n = 1 + \sum_{i=1}^{n} \frac{1}{i^2 + i}$$

for $n = 4999$. The recursive summation algorithm (and MATLAB) gives the result

$$1.999800000000002e + 000.$$

If the summation is done in the reverse (increasing) order, then the result is

$$1.999800000000000e + 000.$$

If the two values are compared with the exact result $s_n = 2 - 1/(n + 1)$, then we can see that the second summation gives better result. In this case the sum of smaller numbers gives significant digits to the final result unlike in the first case.

The last example indicates that the summation of a large number of data varying in modulus and sign is a complicated task. The following algorithm of W. Kahan is one of the most interesting procedures to solve the problem.

```
Compensated-Summation($n$, $x$)
1  $s \leftarrow 0$
2  $e \leftarrow 0$
3  for $i \leftarrow 1$ to $n$
4      do $t \leftarrow s$
5         $y \leftarrow x_i + e$
6         $s \leftarrow t + y$
7         $e \leftarrow (t - s) + y$
8  return $s$
```

18.1.4. The floating point arithmetic standard

The ANSI/IEEE Standard 754-1985 of a binary ($\beta = 2$) *floating point arithmetic system* was published in 1985. The standard specifies the basic arithmetic operations, comparisons, rounding modes, the arithmetic exceptions and their handling, and conversion between the different arithmetic formats. The square root is included as a basic operation. The standard does not deal with the exponential and transcendent functions. The standard defines two main floating point formats:

Type	Size	Mantissa	e	u	$[M_L, M_U] \approx$
Single	32 bits	23 + 1 bits	8 bits	$2^{-24} \approx 5.96 \times 10^{-8}$	$10^{\pm 38}$
Double	64 bits	52 + 1 bits	11 bits	$2^{-53} \approx 1.11 \times 10^{-16}$	$10^{\pm 308}$

In both formats one bit is reserved as a sign bit. Since the floating point numbers are normalized and the first digit is always 1, this bit is not stored. This hidden bit is denoted by the „+1" in the table.

The arithmetic standard contains the handling of arithmetic exceptions.

Exception type	Example	Default result
Invalid operation	$0/0,\ 0 \times \infty,\ \sqrt{-1}$	NaN (Not a Number)
Overflow	$\lvert x\square y\rvert > M_U$	$\pm\infty$
Divide by zero	Finite nonzero/0	$\pm\infty$
Underflow	$0 < \lvert x\square y\rvert < M_L$	Subnormal numbers
Inexact	$fl\,(x\square y) \neq x\square y$	Correctly rounded result

(The numbers of the form $\pm m \times \beta^{L-t}$, $0 < m < \beta^{t-1}$ are called ***subnormal numbers***.) The IEEE arithmetic is a closed system. Every arithmetic operations has a result, whether it is expected mathematically or not. The exceptional operations raise a signal and continue. The arithmetic standard conforms with the standard model (18.4).

The first hardware implementation of the IEEE standard was the Intel 8087 mathematical coprocessor. Since then it is generally accepted and used.

Remark. In the single precision we have about 7 significant digit precision in the decimal system. For double precision we have approximately 16 digit precision in decimals. There also exists an extended precision format of 80 bits, where $t = 63$ and the exponential has 15 bits.

Exercises

18.1-1 The measured values of two resistors are $R_1 = 110.2 \pm 0.3\Omega$ and $R_2 = 65.6 \pm 0.2\Omega$. We connect the two resistors parallel and obtain the circuit resistance $R_e = R_1R_2/(R_1 + R_2)$. Calculate the relative error bounds of the initial data and the approximate value of the resistance R_e. Evaluate the absolute and relative error bounds δR_e and $\delta R_e/R_e$, respectively in the following three ways:
(i) Estimate first δR_e using only the absolute error bounds of the input data, then estimate the relative error bound $\delta R_e/R_e$.
(ii) Estimate first the relative error bound $\delta R_e/R_e$ using only the relative error bounds of the input data, then estimate the absolute error bound δR_e.
(iii) Consider the circuit resistance as a two variable function $R_e = F(R_1, R_2)$.
18.1-2 Assume that $\sqrt{2}$ is calculated with the absolute error bound 10^{-8}. The following two expressions are theoretically equal:
(i) $1/\left(1 + \sqrt{2}\right)^6$;
(ii) $99 - 70\sqrt{2}$.
Which expression can be calculated with less relative error and why?
18.1-3 Consider the arithmetic operations as two variable functions of the form $f(x, y) = x\square y$, where $\square \in \{+, -, *, /\}$.
(i) Derive the *error bounds* of the arithmetic operations from the error formula of two variable functions.
(ii) Derive the *condition numbers* of these functions. When are they ill-conditioned?
(iii) Derive *error bounds* for the power function assuming that both the base and the exponent have errors. What is the result if the exponent is exact?
(iv) Let $y = 16x^2$, $x \approx a$ and $y \approx b = 16a^2$. Determine the smallest and the greatest value of a as a function of x such that the *relative error bound* of b should be at most 0.01.
18.1-4 Assume that the number $C = \mathrm{EXP}(4\pi^2/\sqrt{83})$ $(= 76.1967868\ldots)$ is calculated in a 24 bit long mantissa and the exponential function is also calculated with 24 significant bits. Estimate the *absolute error* of the result. Estimate the *relative error* without using the actual value of C.
18.1-5 Consider the emphfloating point number set $F(\beta, t, L, U)$ and show that
(i) Every arithmetic operation can result *arithmetic overflow*;
(ii) Every arithmetic operation can result *arithmetic underflow*.
18.1-6 Show that the following expressions are *numerically unstable* for $x \approx 0$:
(i) $(1 - \cos x)/\sin^2 x$;
(ii) $\sin(100\pi + x) - \sin(x)$;
(iii) $2 - \sin x - \cos x - e^{-x}$.
Calculate the values of the above expressions for $x = 10^{-3}, 10^{-5}, 10^{-7}$ and estimate the error. Manipulate the expressions into *numerically stable* ones and estimate the error as well.
18.1-7 How many elements does the set $F = F(\beta, t, L, U)$ have? How many *subnormal numbers* can we find?
18.1-8 If $x, y \geq 0$, then $(x + y)/2 \geq \sqrt{xy}$ and equality holds if and only if $x = y$. Is it true numerically? Check the inequality experimentally for various data (small and large numbers, numbers close to each other or different in magnitude).

18.2. Linear systems of equations

The general form of linear algebraic systems with n unknowns and m equations is given by

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1j}x_j + \cdots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{i1}x_1 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n &= b_i \\ &\vdots \\ a_{m1}x_1 + \cdots + a_{mj}x_j + \cdots + a_{mn}x_n &= b_m \end{aligned} \tag{18.14}$$

This system can be written in the more compact form

$$Ax = b, \tag{18.15}$$

where

$$A = \left[a_{ij}\right]_{i,j=1}^{m,n} \in \mathbb{R}^{m\times n},\ x \in \mathbb{R}^n,\ b \in \mathbb{R}^m.$$

The systems is called *underdetermined* if $m < n$. For $m > n$, the systems is called *overdetermined*. Here we investigate only the case $m = n$, when the coefficient matrix A is square. We also assume that the inverse matrix A^{-1} exists (or equivalently $\det(A) \neq 0$). Under this assumption the linear system $Ax = b$ has exactly one solution: $x = A^{-1}b$.

18.2.1. Direct methods for solving linear systems

Triangular linear systems

Definition 18.4 *The matrix $A = [a_{ij}]_{i,j=1}^n$ is **upper triangular** if $a_{ij} = 0$ for all $i > j$. The matrix A is **lower triangular** if $a_{ij} = 0$ for all $i < j$.*

For example the general form of the upper triangular matrices is the following:

$$\begin{bmatrix} * & * & \cdots & \cdots & * \\ 0 & * & & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * \\ 0 & \cdots & \cdots & 0 & * \end{bmatrix}.$$

We note that the diagonal matrices are both lower and upper triangular. It is easy to show that $\det(A) = a_{11}a_{22}\ldots a_{nn}$ holds for the upper or lower triangular matrices. It is easy to solve linear systems with triangular coefficient matrices. Consider the following upper triangular linear system:

$$\begin{array}{ccccccc} a_{11}x_1+ & \cdots & +a_{1i}x_i+ & \cdots & +a_{1n}x_n & = & b_1 \\ & \ddots & \vdots & & \vdots & & \vdots \\ & & a_{ii}x_i+ & \cdots & +a_{in}x_n & = & b_i \\ & & & \ddots & \vdots & & \vdots \\ & & & & a_{nn}x_n & = & b_n \end{array}$$

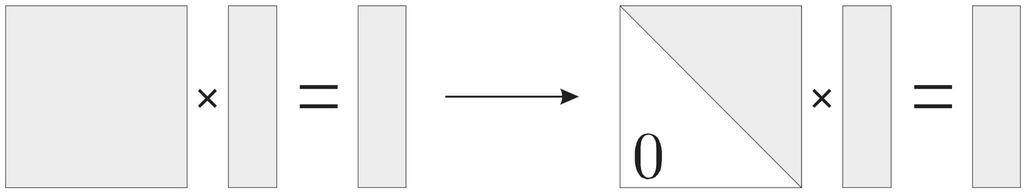

Figure 18.2. Gaussian elimination.

This can be solved by the so called *back substitution* algorithm.

```
Back-Substitution($A, b, n$)
1  $x_n \leftarrow b_n/a_{nn}$
2  for $i \leftarrow n-1$ downto 1
3      do $x_i \leftarrow (b_i - \sum_{j=i+1}^{n} a_{ij}x_j)/a_{ii}$
4  return $x$
```

The solution of lower triangular systems is similar.

The Gauss method

The Gauss method or Gaussian elimination (GE) consists of two phases:
I. The linear system $Ax = b$ is transformed to an equivalent upper triangular system using elementary operations (see Figure 18.2).
II. The obtained upper triangular system is then solved by the back substitution algorithm.

The first phase is often called the elimination or forward phase. The second phase of GE is called the backward phase. The elementary operations are of the following three types:

1. Add a multiple of one equation to another equation.
2. Interchange two equations.
3. Multiply an equation by a nonzero constant.

The elimination phase of GE is based on the following observation. Multiply equation k by $\gamma \neq 0$ and subtract it from equation i:

$$(a_{i1} - \gamma a_{k1})\, x_1 + \cdots + \left(a_{ij} - \gamma a_{kj}\right) x_j + \cdots + (a_{in} - \gamma a_{kn})\, x_n = b_i - \gamma b_k .$$

If $a_{kj} \neq 0$, then by choosing $\gamma = a_{ij}/a_{kj}$, the coefficient of x_j becomes 0 in the new equivalent equation, which replaces equation i. Thus we can eliminate variable x_j (or coefficient a_{ij}) from equation i.

The Gauss method eliminates the coefficients (variables) under the main diagonal of A in a systematic way. First variable x_1 is eliminated from equations $i = 2, \ldots, n$ using equation 1, then x_2 is eliminated from equations $i = 3, \ldots, n$ using equation 2, and so on.

Assume that the unknowns are eliminated in the first $(k-1)$ columns under the main diagonal and the resulting linear system has the form

$$\begin{array}{cccccccccc}
a_{11}x_1 + & \cdots & \cdots & +a_{1k}x_k & + & \cdots & + & a_{1n}x_n & = & b_1 \\
 & \ddots & & \vdots & & & & \vdots & & \vdots \\
 & & \ddots & \vdots & & & & \vdots & & \vdots \\
 & & & a_{kk}x_k & + & \cdots & + & a_{kn}x_n & = & b_k \\
 & & & \vdots & & & & \vdots & & \vdots \\
 & & & a_{ik}x_k & + & \cdots & + & a_{in}x_n & = & b_i \\
 & & & \vdots & & & & \vdots & & \vdots \\
 & & & a_{nk}x_k & + & \cdots & + & a_{nn}x_n & = & b_n
\end{array}$$

If $a_{kk} \neq 0$, then multiplying row k by γ and subtracting it from equation i we obtain

$$(a_{ik} - \gamma a_{kk})x_k + (a_{i,k+1} - \gamma a_{k,k+1})x_{k+1} + \cdots + (a_{in} - \gamma a_{kn})x_n = b_i - \gamma b_k .$$

Since $a_{ik} - \gamma a_{kk} = 0$ for $\gamma = a_{ik}/a_{kk}$, we eliminated the coefficient a_{ik} (variable x_k) from equation $i > k$. Repeating this process for $i = k + 1, \ldots, n$ we can eliminate the coefficients under the main diagonal entry a_{kk}. Next we denote by $A[i, j]$ the element a_{ij} of matrix A and by $A[i, j : n]$ the vector $\left[a_{ij}, a_{i,j+1}, \ldots, a_{in}\right]$. The Gauss method has the following form (where the *pivoting* discussed later is also included):

```
Gauss-Method($A, b$)
▷ Forward phase:
$n \leftarrow rows[A]$
**for** $k \leftarrow 1$ **to** $n - 1$
    **do** { pivoting and interchange of rows and columns}
       **for** $i \leftarrow k + 1$ **to** $n$
           **do** $\gamma_{ik} \leftarrow A[i, k] / A[k, k]$
              $A[i, k + 1 : n] \leftarrow A[i, k + 1 : n] - \gamma_{ik} * A[k, k + 1 : n]$
              $b_i \leftarrow b_i - \gamma_{ik} b_k$
▷ Backward phase: see the back substitution algorithm.
**return** $x$
```

The algorithm overwrites the original matrix A and vector b. It does not write however the zero entries under the main diagonal since these elements are not necessary for the second phase of the algorithm. Hence the lower triangular part of matrix A can be used to store information for the *LU* decomposition of matrix A.

The above version of the Gauss method can be performed only if the elements a_{kk} occurring in the computation are not zero. For this and numerical stability reasons we use the Gaussian elimination with *pivoting*.

The Gauss method with pivoting

If $a_{kk} = 0$, then we can interchange row k with another row, say i, so that the new entry (a_{ki}) at position (k, k) should be nonzero. If this is not possible, then all the coefficients $a_{kk}, a_{k+1,k}, \ldots, a_{nk}$ are zero and $\det(A) = 0$. In the latter case $Ax = b$ has no unique solution. The element a_{kk} is called the k^{th} ***pivot element***. We can always select new pivot elements by interchanging the rows. The selection of the pivot element has a great influence on the reliability of the computed results. The simple fact that we divide by the pivot element indicates this influence. We recall that $\delta(a/b)$ is proportional to $1/|b|^2$. It is considered advantageous if the pivot element is selected so that it has the greatest possible modulus. The process of selecting the pivot element is called *pivoting*. We mention the following two pivoting processes.

Partial pivoting: At the k^{th} step, interchange the rows of the matrix so the largest remaining element, say a_{ik}, in the k^{th} column is used as pivot. After the pivoting we have

$$|a_{kk}| = \max_{k\le i\le n} |a_{ik}| .$$

Complete pivoting: At the k^{th} step, interchange both the rows and columns of the matrix so that the largest element, say a_{ij}, in the remaining matrix is used as pivot After the pivoting we have

$$|a_{kk}| = \max_{k\le i,j\le n} \left|a_{ij}\right| .$$

Note that the interchange of two columns implies the interchange of the corresponding unknowns. The significance of pivoting is well illustrated by the following

Example 18.6 The exact solution of the linear system

$$\begin{array}{cccccc} 10^{-17}x & + & y & = & 1 \\ x & + & y & = & 2 \end{array}$$

is $x = 1/(1-10^{-17})$ and $y = 1-10^{-17}/(1-10^{-17})$. The MATLAB program gives the result $x = 1, y = 1$ and this is the best available result in standard double precision arithmetic. Solving this system with the Gaussian elimination without pivoting (also in double precision) we obtain the catastrophic result $x = 0$ and $y = 1$. Using partial pivoting with the Gaussian elimination we obtain the best available numerical result $x = y = 1$.

Remark 18.5 *Theoretically we do not need pivoting in the following cases: 1. If A is symmetric and positive definite ($A \in \mathbb{R}^{n\times n}$ is positive definite $\Leftrightarrow x^TAx > 0$, $\forall x \in \mathbb{R}^n$, $x \ne 0$). 2. If A is diagonally dominant in the following sense:*

$$|a_{ii}| > \sum_{j\ne i} \left|a_{ij}\right| \quad (1 \le i \le n) .$$

In case of symmetric and positive definite matrices we use the Cholesky method which is a special version of the Gauss-type methods.

During the Gaussian elimination we obtain a sequence of equivalent linear systems

$$A^{(0)}x = b^{(0)} \to A^{(1)}x = b^{(1)} \to \cdots \to A^{(n-1)}x = b^{(n-1)},$$

where

$$A^{(0)} = A, \quad A^{(k)} = \left[a_{ij}^{(k)}\right]_{i,j=1}^{n} .$$

Note that matrices $A^{(k)}$ are stored in the place of $A = A^{(0)}$. The last coefficient matrix of phase I has the form

$$A^{(n-1)} = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & a_{nn}^{(n-1)} \end{bmatrix},$$

where $a_{kk}^{(k-1)}$ is the kth pivot element. The ***growth factor of pivot elements*** is given by

$$\rho = \rho_n = \max_{1\le k\le n} \left| a_{kk}^{(k-1)} / a_{11}^{(0)} \right| .$$

Wilkinson proved that the error of the computed solution is proportional to the growth factor ρ and the bounds

$$\rho \le \sqrt{n}\left(2 \cdot 3^{\frac{1}{2}} \cdots n^{\frac{1}{n-1}}\right)^{\frac{1}{2}} \sim cn^{\frac{1}{2}} n^{\frac{1}{4}log(n)}$$

and

$$\rho \le 2^{n-1}$$

hold for complete and partial pivoting, respectively. Wilkinson conjectured that $\rho \le n$ for complete pivoting. This has been proved by researchers for small values of n. Statistical investigations on random matrices ($n \le 1024$) indicate that the average of ρ is $\Theta\left(n^{2/3}\right)$ for the partial pivoting and $\Theta\left(n^{1/2}\right)$ for the complete pivoting. Hence the case $\rho > n$ hardly occurs in the statistical sense.

We remark that Wilkinson constructed a linear system on which $\rho = 2^{n-1}$ for the partial pivoting. Hence Wilkinson's bound for ρ is sharp in the case of partial pivoting. There also exist examples of linear systems concerning discretizations of differential and integral equations, where ρ is increasing exponentially if Gaussian elimination is used with partial pivoting.

The growth factor ρ can be very large, if the Gaussian elimination is used without pivoting. For example, $\rho = \rho_4(A) = 1.23 \times 10^5$, if

$$A = \begin{bmatrix} 1.7846 & -0.2760 & -0.2760 & -0.2760 \\ -3.3848 & 0.7240 & -0.3492 & -0.2760 \\ -0.2760 & -0.2760 & 1.4311 & -0.2760 \\ -0.2760 & -0.2760 & -0.2760 & 0.7240 \end{bmatrix}.$$

Operations counts

The Gauss method gives the solution of the linear system $Ax = b$ ($A \in \mathbb{R}^{n\times n}$) in a finite number of steps and arithmetic operations ($+, -, *, /$). The amount of necessary arithmetic operations is an important characteristic of the direct linear system solvers, since the CPU time is largely proportional to the number of arithmetic operations. It was also observed that the number of additive and multiplicative operations are nearly the same in the numerical algorithms of linear algebra. For measuring the cost of such algorithms C.B. Moler introduced the concept of flop.

Definition 18.6 *One (old) flop is the computational work necessary for the operation* $s = s + x * y$ *(1 addition + 1 multiplication).* ***One (new) flop*** *is the computational work necessary for any of the arithmetic operations* $+, -, *, /$.

The new flop can be used if the computational time of additive and multiplicative operations are approximately the same. Two new flops equals to one old flop. Here we use the notion of old flop.

For the Gauss method a simple counting gives the number of additive and multiplicative operations.

Theorem 18.7 *The computational cost of the Gauss method is $n^3/3 + \Theta(n^2)$ flops.*

V.V. Klyuyev and N. Kokovkin-Shcherbak proved that if only elementary row and column operations (multiplication of row or column by a number, interchange of rows or columns, addition of a multiple of row or column to another row or column) are allowed, then the linear system $Ax = b$ cannot be solved in less than $n^3/3 + \Omega(n^2)$ flops.

Using fast matrix inversion procedures we can solve the $n \times n$ linear system $Ax = b$ in $O(n^{2.808})$ flops. These theoretically interesting algorithms are not used in practice since they are considered as numerically unstable.

The LU-decomposition

In many cases it is easier to solve a linear system if the coefficient matrix can be decomposed into the product of two triangular matrices.

Definition 18.8 *The matrix $A \in \mathbb{R}^{n\times n}$ has an **LU-decomposition**, if $A = LU$, where $L \in \mathbb{R}^{n\times n}$ is lower and $U \in \mathbb{R}^{n\times n}$ is upper triangular matrix.*

The LU-decomposition is not unique. If a nonsingular matrix has an LU-decomposition, then it has a particular LU-decomposition, where the main diagonal of a given component matrix consists of 1's. Such triangular matrices are called ***unit (upper or lower) triangular*** matrices. The LU decomposition is unique, if L is set to be lower unit triangular or U is set to be unit upper triangular.

The LU-decomposition of nonsingular matrices is closely related to the Gaussian elimination method. If $A = LU$, where L is unit lower triangular, then $l_{ik} = \gamma_{ik}$ $(i > k)$, where γ_{ik} is given by the Gauss algorithm. The matrix U is the upper triangular part of the matrix we obtain at the end of the forward phase. The matrix L can also be derived from this matrix, if the columns of the lower triangular part are divided by the corresponding main diagonal elements. We remind that the first phase of the Gaussian elimination does not annihilate the matrix elements under the main diagonal. It is clear that a nonsingular matrix has LU-decomposition if and only if $a_{kk}^{(k-1)} \neq 0$ holds for each pivot element for the Gauss method without pivoting.

Definition 18.9 *A matrix $P \in \mathbb{R}^{n\times n}$ whose every row and column has one and only one non-zero element, that element being 1, is called a **permutation matrix**.*

In case of partial pivoting we permute the rows of the coefficient matrix (multiply A by a permutation matrix on the left) so that $a_{kk}^{(k-1)} \neq 0$ $(k = 1, \ldots, n)$ holds for a nonsingular matrix. Hence we have

Theorem 18.10 *If $A \in \mathbb{R}^{n\times n}$ is nonsingular then there exists a permutation matrix P such that PA has an LU-decomposition.*

The the algorithm of LU-decomposition is essentially the Gaussian elimination method. If pivoting is used then the interchange of rows must also be executed on the elements under the main diagonal and the permutation matrix P must be recorded. A vector containing the actual order of the original matrix rows is obviously sufficient for this purpose.

The LU- and Cholesky-methods

Let $A = LU$ and consider the equation $Ax = b$. Since $Ax = LUx = L(Ux) = b$, we can decompose $Ax = b$ into the equivalent linear system $Ly = b$ and $Ux = b$, where L is lower triangular and U is upper triangular.

```
LU-Method(A, b)

1  Determine the LU-decomposition $A = LU$.
2  Solve $Ly = b$.
3  Solve $Ux = y$.
4  return $x$
```

Remark. In case of partial pivoting we obtain the decomposition $\hat{A} = PA = LU$ and we set $\hat{b} = Pb$ instead of b.

In the first phase of the Gauss method we produce decomposition $A = LU$ and the equivalent linear system $Ux = L^{-1}b$ with upper triangular coefficient matrix. The latter is solved in the second phase. In the LU-method we decompose the first phase of the Gauss method into two steps. In the first step we obtain only the decomposition $A = LU$. In the second step we produce the vector $y = L^{-1}b$. The third step of the algorithm is identical with the second phase of the original Gauss method.

The LU-method is especially advantageous if we have to solve several linear systems with the same coefficient matrix:

$$Ax = b_1,\ Ax = b_2, \ldots,\ Ax = b_k.$$

In such a case we determine the LU-decomposition of matrix A only once, and then we solve the linear systems $Ly_i = b_i$, $Ux_i = y_i$ $(x_i, y_{i,} b_i \in \mathbf{R}^n$, $i = 1, \ldots, k)$. The computational cost of this process is $n^3/3 + kn^2 + \Theta(kn)$ flops.

The *inversion of a matrix* $A \in \mathbb{R}^{n\times n}$ can be done as follows:

1. Determine the LU-decomposition $A = LU$. .

2. Solve $Ly_i = e_i$, $Ux_i = y_i$ (e_i is the i^{th} unit vector $i = 1, \ldots, n$).

The inverse of A is given by $A^{-1} = [x_1, \ldots, x_n]$. The computational cost of the algorithm is $4n^3/3 + \Theta\left(n^2\right)$ flops.

The LU-method with pointers

This implementation of the LU-method is known since the 60's. Vector P contains the indices of the rows. At the start we set $P[i] = i$ $(1 \le i \le n)$. When exchanging rows we exchange only those components of vector P that correspond to the rows.

```
LU-Method-with-Pointers($A, b$)
$n \leftarrow rows[A]$
$P \leftarrow [1, 2, \ldots, n]$
for $k \leftarrow 1$ to $n - 1$
    do compute index $t$ such that $|A[P[t], k]| = \max_{k \le i \le n} |A[P[i], k]|$ .
       if $k < t$
          then exchange the components $P[k]$ and $P[t]$.
       for $i \leftarrow k + 1$ to $n$
           do $A[P[i], k] \leftarrow A[P[i], k] / A[P[k], k]$
              $A[P[i], k+1:n] \leftarrow A[P[i], k+1:n] - A[P[i], k] * A[P[k], k+1:n]$
for $i \leftarrow 1$ to $n$
    do $s \leftarrow 0$
       for $j \leftarrow 1$ to $i - 1$
           do $s \leftarrow s + A[P[i], j] * x[j]$
       $x[i] \leftarrow b[P[i]] - s$
for $i \leftarrow n$ downto 1
    do $s \leftarrow 0$
       for $j \leftarrow i + 1$ to $n$
           $s \leftarrow s + A[P[i], j] * x[j]$
       $x[i] \leftarrow (x[i] - s) / A[P[i], i]$
return $x$
```

If $A \in \mathbb{R}^{n \times n}$ is *symmetric and positive definite*, then it can be decomposed in the form $A = LL^T$, where L is lower triangular matrix. The LL^T-decomposition is called the ***Cholesky-decomposition*** . In this case we can save approximately half of the storage place for A and half of the computational cost of the LU-decomposition (LL^T-decomposition). Let

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

Observing that only the first k elements may be nonzero in the k^{th} column of L^T we obtain that

$$\begin{aligned} a_{kk} &= l_{k1}^2 + l_{k2}^2 + \cdots + l_{k,k-1}^2 + l_{kk}^2, \\ a_{ik} &= l_{i1}l_{k1} + l_{i2}l_{k2} + \cdots + l_{i,k-1}l_{k,k-1} + l_{ik}l_{kk} \quad (i = k+1, \ldots, n). \end{aligned}$$

This gives the formulae

$$\begin{aligned} l_{kk} &= (a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2)^{1/2}, \\ l_{ik} &= (a_{ik} - \sum_{j=1}^{k-1} l_{ij}l_{kj})/l_{kk} \quad (i = k+1, \ldots, n). \end{aligned}$$

Using the notation $\sum_{j=i}^{k} s_j = 0$ $(k < i)$ we can formulate the Cholesky-method as follows.

```
Cholesky-Method(A)
1  $n \leftarrow rows[A]$
2  for $k \leftarrow 1$ to $n$
3      do $a_{kk} \leftarrow (a_{kk} - \sum_{j=1}^{k-1} a_{kj}^2)^{1/2}$
4         for $i \leftarrow k+1$ to $n$
5             do $a_{ik} \leftarrow (a_{ik} - \sum_{j=1}^{k-1} a_{ij}a_{kj})/a_{kk}$
6  return $A$
```

The lower triangular part of A contains L. The computational cost of the algorithm is $n^3/6 + \Theta(n^2)$ flops and n square roots. The algorithm, which can be considered as a special case of the Gauss-methods, does not require pivoting, at least in principle.

The *LU*- and Cholesky-methods on banded matrices

It often happens that linear systems have *banded coefficient matrices.*

Definition 18.11 *Matrix $A \in \mathbb{R}^{n \times n}$ is banded with lower bandwidth p and upper bandwidth q if*

$$a_{ij} = 0, \quad \text{if } i > j + p \text{ or } j > i + q .$$

The possibly non-zero elements a_{ij} $(i - p \le j \le i + q)$ form a band like structure. Schematically A has the form

$$A = \begin{bmatrix}
a_{11} & a_{12} & \cdots & \cdots & a_{1,1+q} & 0 & \cdots & \cdots & 0 \\
a_{21} & a_{22} & & & & \ddots & & & \vdots \\
\vdots & & \ddots & & & & \ddots & & \vdots \\
a_{1+p,1} & & & \ddots & & & & \ddots & 0 \\
0 & \ddots & & & \ddots & & & & a_{n-q,n} \\
\vdots & & \ddots & & & \ddots & & & \vdots \\
\vdots & & & \ddots & & & \ddots & & \vdots \\
\vdots & & & & \ddots & & & \ddots & a_{n-1,n} \\
0 & \cdots & \cdots & \cdots & 0 & a_{n,n-p} & \cdots & a_{n,n-1} & a_{nn}
\end{bmatrix} .$$

The *banded matrices* yield very efficient algorithms if p and q are significantly less than n. If a banded matrix A with lower bandwidth p and upper bandwidth q has an LU-decomposition, then both L and U are banded with lower bandwidth p and upper bandwidth q, respectively.

Next we give the LU-method for banded matrices in three parts.

```
The-LU-Decomposition-of-Banded-Matrix($A, n, p, q$)
1  for $k \leftarrow 1$ to $n-1$
2      do for $i \leftarrow k+1$ to $\min\{k+p, n\}$
3            do $a_{ik} \leftarrow a_{ik}/a_{kk}$
4               for $j \leftarrow k+1$ to $\min\{k+q, n\}$
5                   do $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj}$
6  return $A$
```

Entry a_{ij} is overwritten by l_{ij}, if $i > j$ and by u_{ij}, if $i \leq j$. The computational cost of is $c(p,q)$ flops, where

$$c(p,q) = \begin{cases} npq - \frac{1}{2}pq^2 - \frac{1}{6}p^3 + pn, & p \leq q \\ npq - \frac{1}{2}qp^2 - \frac{1}{6}q^3 + qn, & p > q \end{cases}$$

The following algorithm overwrites b by the solution of equation $Ly = b$.

```
Solution of banded unit lower triangular system($L, b, n, p$)
1  for $i \leftarrow 1$ to $n$
2      do $b_i \leftarrow b_i - \sum_{j=\max\{1,i-p\}}^{i-1} l_{ij}b_j$
3  return $b$
```

The total cost of the algorithm is $np - p^2/2$ flops. The next algorithm overwrites vector b by the solution of $Ux = b$.

```
Solution-of-Banded-Upper-Triangular-System($U, b, n, q$)
1  for $i \leftarrow n$ downto 1
2      do $b_i \leftarrow \left(b_i - \sum_{j=i+1}^{\min\{i+q,n\}} u_{ij}b_j\right)/u_{ii}$
3  return $b$
```

The computational cost is $n(q+1) - q^2/2$ flops.

Assume that $A \in \mathbb{R}^{n\times n}$ is symmetric, positive definite and banded with lower bandwidth p. The banded version of the Cholesky-methods is given by

```
Cholesky-decomposition-of-Banded-Matrices($A, n, p$)
1  for $i \leftarrow 1$ to $n$
2      do for $j \leftarrow \max\{1, i-p\}$ to $i-1$
3            do $a_{ij} \leftarrow \left(a_{ij} - \sum_{k=\max\{1,i-p\}}^{j-1} a_{ik}a_{jk}\right)/a_{jj}$
4         $a_{ii} \leftarrow \left(a_{ii} - \sum_{k=\max\{1,i-p\}}^{i-1} a_{ik}^2\right)^{1/2}$
5  return $A$
```

The elements a_{ij} are overwritten by l_{ij} $(i \geq j)$. The total amount of work is given by $\left(np^2/2\right) - \left(p^3/3\right) + (3/2)\left(np - p^2\right)$ flops és n square roots.

Remark. If $A \in \mathbb{R}^{n\times n}$ has lower bandwidth p and upper bandwidth q and partial pivoting takes place, then the upper bandwidth of U increases up to $\hat{q} = p + q$.

18.2.2. Iterative methods for linear systems

There are several iterative methods for solving linear systems of algebraic equations. The best known iterative algorithms are the classical *Jacobi-*, the *Gauss-Seidel-* and the *relaxation methods*. The greatest advantage of these iterative algorithms is their easy implementation to large systems. At the same time they usually have slow convergence. However for parallel computers the *multisplitting* iterative algorithms seem to be efficient.

Consider the iteration

$$x_i = Gx_{i-1} + b \quad (i = 1, 2, \ldots)$$

where $G \in \mathbb{R}^{n\times n}$ és $x_0, b \in \mathbb{R}^n$. It is known that $\{x_i\}_{i=0}^{\infty}$ converges for all $x_0, b \in \mathbb{R}^n$ if and only if the spectral radius of G satisfies $\rho(G) < 1$ ($\rho(G) = \max|\lambda| \mid \lambda$ is an eigenvalue of G). In case of convergence $x_i \to x^* = (I-G)^{-1} b$, that is we obtain the solution of the equation $(I-G)x = b$. The speed of convergence depends on the spectral radius $\rho(G)$. Smaller the spectral radius$\rho(G)$, faster the convergence.

Consider now the linear system

$$Ax = b,$$

where $A \in \mathbb{R}^{n\times n}$ is nonsingular. The matrices $M_l, N_l, E_l \in \mathbf{R}^{n\times n}$ form a multisplitting of A if

(i) $A = M_i - N_i$, $i = 1, 2, \ldots, L$,

(ii) M_i is nonsingular, $i = 1, 2, \ldots, L$,

(iii) E_i is non-negative diagonal matrix, $i = 1, 2, \ldots, L$,

(iv) $\sum_{i=1}^{L} E_i = I$.

Let $x_0 \in \mathbb{R}^n$ be a given initial vector. The multisplitting iterative method is the following.

```
MULTISPLITTING-ITERATION($x_0, b, L, M_l, N_l, E_l, l = 1, \ldots, L$)
1 $i \leftarrow 0$
2 while exit condition=FALSE
3       do $i \leftarrow i + 1$
4          for $l \leftarrow 1$ to $L$
5              do $M_l y_l \leftarrow N_l x_{i-1} + b$
6          $x_i \leftarrow \sum_{l=1}^{L} E_l y_l$
7 return $x_i$
```

It is easy to show that $y_l = M_l^{-1} N_l x_{i-1} + M_l^{-1} b$ and

$$\begin{aligned} x_i = \sum_{l=1}^{L} E_l y_l &= \sum_{l=1}^{L} E_l M_l^{-1} N_l x_{i-1} + \sum_{l=1}^{L} E_l M_l^{-1} b \\ &= Hx_{i-1} + c. \end{aligned}$$

Thus the condition of convergence is $\rho(H) < 1$. The *multisplitting iteration* is a true *parallel algorithm* because we can solve L linear systems parallel in each iteration (synchronized parallelism). The bottleneck of the algorithm is the computation of iterate x_i.

The selection of matrices M_i and E_i is such that the solution of the linear system $M_i y = c$ should be cheap. Let $S_1, S_2, \ldots, S_L$ be a partition of $\{1, \ldots, n\}$, that is $S_i \neq \emptyset$, $S_i \cap S_j = \emptyset$ $(i \neq j)$ and $\cup_{i=1}^{L} S_i = \{1, \ldots, n\}$. Furthermore let $S_i \subseteq T_i \subseteq \{1, \ldots, n\}$ $(i = 1, \ldots, L)$ be such that $S_l \neq T_l$ for at least one l.

The ***non-overlapping block Jacobi splitting*** of A is given by

$$M_l = \left[M_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad M_{ij}^{(l)} = \begin{cases} a_{ij}, & \text{if } i,j \in S_l \\ a_{ii}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases},$$

$$N_l = M_l - A,$$

$$E_l = \left[E_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad E_{ij}^{(l)} = \begin{cases} 1, & \text{if } i = j \in S_l \\ 0, & \text{otherwise} \end{cases}$$

for $l = 1, \ldots, L$.

Define now the simple splitting

$$A = M - N,$$

where M is nonsingular,

$$M = \left[M_{ij}\right]_{i,j=1}^{n}, \quad M_{ij} = \begin{cases} a_{ij}, & \text{if } i,j \in S_l \text{ for some } l \in \{1,\ldots,n\} \\ 0, & \text{otherwise} \end{cases}.$$

It can be shown that

$$H = \sum_{l=1}^{L} E_l M_l^{-1} N_l = M^{-1} N$$

holds for the non-overlapping block Jacobi multisplitting.

The ***overlapping block Jacobi multisplitting*** of A is defined by

$$\widetilde{M}_l = \left[\widetilde{M}_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad \widetilde{M}_{ij}^{(l)} = \begin{cases} a_{ij}, & \text{if } i,j \in T_l \\ a_{ii}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases},$$

$$\widetilde{N}_l = \widetilde{M}_l - A,$$

$$\widetilde{e}_l = \left[\widetilde{e}_{ij}^{(l)}\right]_{i,j=1}^{n}, \quad E_{ii}^{(l)} = 0, \text{ if } i \notin T_l$$

for $l = 1, \ldots, L$.

A nonsingular matrix $A \in \mathbb{R}^{n\times n}$ is called an M-matrix, if $a_{ij} \leq 0$ $(i \neq j)$ and all the elements of A^{-1} are nonnegative.

Theorem 18.12 *Assume that $A \in \mathbb{R}^{n\times n}$ is nonsingular M-matrix, $\{M_i, N_i, E_i\}_{i=1}^{L}$ is a non-overlapping, $\left\{\widetilde{M}_i, \widetilde{N}_i, E_i\right\}_{i=1}^{L}$ is an overlapping block Jacobi multisplitting of A, where the weighting matrices E_i are the same. The we have*

$$\rho\left(\widetilde{H}\right) \leq \rho(H) < 1,$$

where $H = \sum_{l=1}^{L} E_l M_l^{-1} N_l$ and $\widetilde{H} = \sum_{l=1}^{L} E_l \widetilde{M}_l^{-1} \widetilde{N}_l$.

We can observe that both iteration procedures are convergent and the convergence of the overlapping multisplitting is not slower than that of the non-overlapping procedure. The theorem remains true if we use block Gauss-Seidel multisplittings instead of the block Jacobi multisplittings. In this case we replace the above defined matrices M_i and $\widetilde{M}_i$ with their lower triangular parts.

The multisplitting algorithm has multi-stage and asynchronous variants as well.

18.2.3. Error analysis of linear algebraic systems

We analyze the direct and inverse errors. We use the following notations and concepts. The exact (theoretical) solution of $Ax = b$ is denoted by x, while any approximate solution is denoted by $\hat{x}$. The direct error of the approximate solution is given by $\Delta x = \hat{x} - x$. The quantity $r = r(y) = Ay - b$ is called the ***residual error***. For the exact solution $r(x) = 0$, while for the approximate solution

$$r(\hat{x}) = A\hat{x} - b = A(\hat{x} - x) = A\Delta x.$$

We use various models to estimate the inverse error. In the most general case we assume that the computed solution $\hat{x}$ satisfies the linear system$\hat{A}\hat{x} = \hat{b}$, where $\hat{A} = A + \Delta A$ and $\hat{b} = b + \Delta b$. The quantities ΔA and Δb are called the inverse errors.

One has to distinguish between the sensitivity of the problem and the stability of the solution algorithm. By ***sensitivity of a problem*** we mean the sensitivity of the solution to changes in the input parameters (data). By the ***stability (or sensitivity) of an algorithm*** we mean the influence of computational errors on the computed solution. We measure the sensitivity of a problem or algorithm in various ways. One such characterization is the *condition number*„condition number", which compares the relative errors of the input and output values.

The following general principles are used when applying any algorithm:

- We use only stable or well-conditioned algorithms.

- We cannot solve an unstable (ill-posed or ill-conditioned) problem with a general purpose algorithm, in general.

Sensitivity analysis

Assume that we solve the perturbed equation

$$A\hat{x} = b + \Delta b \tag{18.16}$$

instead of the original $Ax = b$. Let $\widehat{x} = x + \Delta x$ and investigate the differene of the two solutions.

Theorem 18.13 *If A is nonsingular and $b \neq 0$, then*

$$\frac{\|\Delta x\|}{\|x\|} \leq \mathrm{cond}(A)\frac{\|\Delta b\|}{\|b\|} = \mathrm{cond}(A)\frac{\|r(\hat{x})\|}{\|b\|}, \tag{18.17}$$

where $\mathrm{cond}(A) = \|A\|\left\|A^{-1}\right\|$ *is the condition number of A.*

Here we can see that the condition number of A may strongly influence the relative error of the perturbed solution $\widehat{x}$. A linear algebraic system is said to be *well-conditioned* if cond(A) is small, and *ill-conditioned*, if cond(A) is big. It is clear that the terms „small" and „big" are relative and the condition number depends on the norm chosen. We identify the applied norm if it is essential for some reason. For example $\mathrm{cond}_\infty(A) = \|A\|_\infty \left\|A^{-1}\right\|_\infty$. The next example gives possible geometric characterization of the condition number.

Example 18.7 The linear system

$$\begin{array}{rcrcl} 1000x_1 & + & 999x_2 & = & b_1 \\ 999x_1 & + & 998x_2 & = & b_2 \end{array}$$

is ill-conditioned ($\mathrm{cond}_\infty(A) = 3.99 \times 10^6$). The two lines, whose meshpoint defines the system, are almost parallel. Therefore if we perturb the right hand side, the new meshpoint of the two lines will be far from the previous meshpoint.

The inverse error is Δb in the sensitivity model under investigation. Theorem 18.13 gives an estimate of the direct error which conforms with the thumb rule. It follows that we can expect a small relative error of the perturbed solution $\widehat{x}$, if the condition number of A is small.

Example 18.8 Consider the linear system $Ax = b$ with

$$A = \begin{bmatrix} 1+\epsilon & 1 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad x = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Let $\hat{x} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$. Then $r = \begin{bmatrix} 2\epsilon \\ 0 \end{bmatrix}$ and $\|r\|_\infty / \|b\|_\infty = 2\epsilon$, but $\|\hat{x} - x\|_\infty / \|x\|_\infty = 2$.

Consider now the perturbed linear system

$$(A + \Delta A)\,\hat{x} = b \tag{18.18}$$

instead of $Ax = b$. It can be proved that for this perturbation model there exist more than one *inverse errors*„inverse error" among which $\Delta A = -r\left(\widehat{x}\right)\widehat{x}^T / \widehat{x}^T\widehat{x}$ is the inverse error with minimal spectral norm, provided that $\widehat{x}, r\left(\widehat{x}\right) \neq 0$.

The following theorem establish that for small relative residual error the relative inverse error is also small.

Theorem 18.14 *Assume that $\hat{x} \neq 0$ is the approximate solution of $Ax = b$,* $\det(A) \neq 0$ *and $b \neq 0$. If $\|r(\hat{x})\|_2 / \|b\|_2 = \alpha < 1$, the the matrix $\Delta A = -r(\hat{x})\,\hat{x}^T / \hat{x}^T\hat{x}$ satisfies $(A + \Delta A)\,\hat{x} = b$ and $\|\Delta A\|_2 / \|A\|_2 \leq \alpha / (1 - \alpha)$.*

If the relative inverse error and the condition number of A are small, then the relative residual error is small.

Theorem 18.15 *If $r(\hat{x}) = A\hat{x} - b$, $(A + \Delta A)\,\hat{x} = b$, $A \neq 0$, $b \neq 0$ and* $\mathrm{cond}(A)\frac{\|\Delta A\|}{\|A\|} < 1$*, then*

$$\frac{\|r(\hat{x})\|}{\|b\|} \leq \frac{\mathrm{cond}(A)\frac{\|\Delta A\|}{\|A\|}}{1 - \mathrm{cond}(A)\frac{\|\Delta A\|}{\|A\|}}. \tag{18.19}$$

If A is ill-conditioned, then Theorem 18.15 is not true.

Example 18.9 Let $A = \begin{bmatrix} 1+\epsilon & 1 \\ 1 & 1-\epsilon \end{bmatrix}$, $\Delta A = \begin{bmatrix} 0 & 0 \\ 0 & \epsilon^2 \end{bmatrix}$ and $b = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $(0 < \epsilon \ll 1)$. Then $\operatorname{cond}_\infty(A) = (2+\epsilon)^2/\epsilon^2 \approx 4/\epsilon^2$ and $\|\Delta A\|_\infty / \|A\|_\infty = \epsilon^2/(2+\epsilon) \approx \epsilon^2/2$. Let

$$\hat{x} = (A + \Delta A)^{-1} b = \frac{1}{\epsilon^3} \begin{bmatrix} 2-\epsilon+\epsilon^2 \\ -2-\epsilon \end{bmatrix} \approx \begin{bmatrix} 2/\epsilon^3 \\ -2/\epsilon^3 \end{bmatrix}.$$

Then $r(\hat{x}) = A\hat{x} - b = \begin{bmatrix} 0 \\ 2/\epsilon + 1 \end{bmatrix}$ and $\|r(\hat{x})\|_\infty / \|b\|_\infty = 2/\epsilon + 1$, which is not small.

In the most general case we solve the perturbed equation

$$(A + \Delta A)\hat{x} = b + \Delta b \tag{18.20}$$

instead of $Ax = b$. The following general result holds.

Theorem 18.16 *If A is nonsingular,* $\operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|} < 1$ *and* $b \neq 0$*, then*

$$\frac{\|\Delta x\|}{\|x\|} \le \frac{\operatorname{cond}(A)\left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|}\right)}{1 - \operatorname{cond}(A)\frac{\|\Delta A\|}{\|A\|}}. \tag{18.21}$$

This theorem implies the following „thumb rule".

Thumb rule. *Assume that $Ax = b$. If the entries of A and b are accurate to about s decimal places and* $\operatorname{cond}(A) \sim 10^t$*, where $t < s$, then the entries of the computed solution are accurate to about $s - t$ decimal places.*

The assumption $\operatorname{cond}(A)\|\Delta A\|/\|A\| < 1$ of Theorem 18.16 guarantees that that matrix $A + \Delta A$ is nonsingular. The inequality $\operatorname{cond}(A)\|\Delta A\|/\|A\| < 1$ is equivalent with the inequality $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$ and the distance of A from the nearest singular matrix is just $1/\left\|A^{-1}\right\|$. Thus we can give a new characterization of the condition number:

$$\frac{1}{\operatorname{cond}(A)} = \min_{A+\Delta A \text{ is singular}} \frac{\|\Delta A\|}{\|A\|}. \tag{18.22}$$

Thus if a matrix is ill-conditioned, then it is close to a singular matrix. Earlier we defined the condition numbers of matrices as the condition number of the mapping $F(x) = A^{-1}x$.

Let us introduce the following definition.

Definition 18.17 *A linear system solver is said to be **weakly stable** on a matrix class H, if for all well-conditioned $A \in H$ and for all b, the computed solution $\widehat{x}$ of the linear system $Ax = b$ has small relative error $\|\hat{x} - x\|/\|x\|$.*

Putting together Theorems 18.13–18.16 we obtain the following.

Theorem 18.18 (Bunch). *A linear system solver is weakly stable on a matrix class H, if for all well-conditioned $A \in H$ and for all b, the computed solution $\widehat{x}$ of the linear system $Ax = b$ satisfies any of the following conditions:*
(1) $\|\hat{x} - x\|/\|x\|$ is small;
(2) $\|r(\hat{x})\|/\|b\|$ is small;
(3) There exists ΔA such that $(A + \Delta A)\hat{x} = b$ and $\|\Delta A\|/\|A\|$ are small.

The estimate of Theorem 18.16 can be used in practice if we know estimates of $\Delta b, \Delta A$ and cond(A). If no estimates are available, then we can only make a posteriori error estimates.

In the following we study the componentwise error estimates. We first give an estimate for the absolute error of the approximate solution using the components of the inverse error.

Theorem 18.19 (Bauer, Skeel). *Let $A \in \mathbb{R}^n$ be nonsingular and assume that the approximate solution $\widehat{x}$ of $Ax = b$ satisfies the linear system $(A + E)\widehat{x} = b + e$. If $S \in \mathbb{R}^{n\times n}$, $s \in \mathbb{R}^n$ and $\varepsilon > 0$ are such that $S \geq 0$, $s \geq 0$, $|E| \leq \varepsilon S$, $|e| \leq \varepsilon s$ and $\varepsilon \left\| \left|A^{-1}\right| S \right\|_\infty < 1$, then*

$$\left\|\widehat{x} - x\right\|_\infty \leq \frac{\varepsilon \left\| \left|A^{-1}\right| (S\,|x| + s) \right\|_\infty}{1 - \varepsilon \left\| \left|A^{-1}\right| S \right\|_\infty}. \tag{18.23}$$

If $e = 0$ $(s = 0)$, $S = |A|$ and

$$k_r(A) = \left\| \left|A^{-1}\right| |A| \right\|_\infty < 1, \tag{18.24}$$

then we obtain the estimate

$$\left\|\widehat{x} - x\right\|_\infty \leq \frac{\varepsilon k_r(A)}{1 - \varepsilon k_r(A)}. \tag{18.25}$$

The quantity $k_r(A)$ is said to be ***Skeel-norm*** , although it is not a norm in the earlier defined sense. The Skeel-norm satisfies the inequality

$$k_r(A) \leq \mathrm{cond}_\infty(A) = \|A\|_\infty \left\|A^{-1}\right\|_\infty. \tag{18.26}$$

Therefore the above estimate is not worse than the traditional one that uses the standard condition number.

The inverse error can be estimated componentwise by the following result of Oettli and Prager. Let $A, \delta A \in \mathbb{R}^{n\times n}$ and $b, \delta b \in \mathbb{R}^n$. Assume that $\delta A \geq 0$ and $\delta b \geq 0$. Furthermore let

$$D = \left\{\Delta A \in \mathbb{R}^{n\times n} : |\Delta A| \leq \delta A\right\}, \quad G = \left\{\Delta b \in \mathbb{R}^n : |\Delta b| \leq \delta b\right\}.$$

Theorem 18.20 (Oettli, Prager). *The computed solution $\hat{x}$ satisfies a perturbed equation $(A + \Delta A)\hat{x} = b + \Delta b$ with $\Delta A \in D$ and $\Delta b \in G$, if*

$$|r(\hat{x})| = |A\hat{x} - b| \leq \delta A\,|\hat{x}| + \delta b. \tag{18.27}$$

We do not need the condition number to apply this theorem. In practice the entries δA and δb are proportional to the machine epsilon.

Theorem 18.21 (Wilkinson). *The approximate solution $\widehat{x}$ of $Ax = b$ obtained by the Gauss method in floating point arithmetic satisfies the perturbed linear equation*

$$(A + \Delta A)\widehat{x} = b \tag{18.28}$$

with

$$\|\Delta A\|_\infty \leq 8n^3 \rho_n \|A\|_\infty u + O(u^2), \tag{18.29}$$

where ρ_n denotes the groth factor of the pivot elements and u is the unit roundoff.

Since ρ_n is small in practice, the realtive error

$$\frac{\|\Delta A\|_\infty}{\|A\|_\infty} \leq 8n^3 \rho_n u + O(u^2)$$

is also small. Therefore Theorem18.18 implies that the Gauss method is weakly stable both for full and partial pivoting.

Wilkinson's theorem implies that

$$\operatorname{cond}_\infty(A)\frac{\|\Delta A\|_\infty}{\|A\|_\infty} \leq 8n^3 \rho_n \operatorname{cond}_\infty(A)\, u + O\left(u^2\right).$$

For a small condition number we can assume that $1-\operatorname{cond}_\infty(A)\,\|\Delta A\|_\infty / \|A\|_\infty \approx 1$. Using Theorems 18.21 and 18.16 (case $\Delta b = 0$) we obtain the following estimate of the direct error:

$$\frac{\|\Delta x\|_\infty}{\|x\|_\infty} \leq 8n^3 \rho_n \operatorname{cond}_\infty(A)\, u. \tag{18.30}$$

The obtained result supports the thumb rule in the case of the Gauss method.

Example 18.10 Consider the following linear system whose coefficients can be represented exactly:

$$\begin{aligned} 888445x_1 + 887112x_2 &= 1, \\ 887112x_1 + 885781x_2 &= 0. \end{aligned}$$

Here $\operatorname{cond}(A)_\infty$ is big, but $\operatorname{cond}_\infty(A)\|\Delta A\|_\infty/\|A\|_\infty$ is negligible. The exact solution of the problem is $x_1 = 885781$, $x_2 = -887112$. The MATLAB gives the approximate solution $\hat{x}_1 = 885827.23$, $\hat{x}_2 = -887158.30$ with the relative error

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} = 5.22 \times 10^{-5}.$$

Since $s \approx 16$ and $\operatorname{cond}(A)_\infty \approx 3.15 \times 10^{12}$, the result essentially corresponds to the Wilkinson theorem or the thumb rule. The Wilkinson theorem gives the bound

$$\|\Delta A\|_\infty \leq 1.26 \times 10^{-8}$$

for the inverse error. If we use the Oettli-Prager theorem with the choice $\delta A = \epsilon_M |A|$ and $\delta b = \epsilon_M |b|$, then we obtain the estimate $|r(\hat{x})| \leq \delta A\, |\hat{x}| + \delta b$. Since $\||\delta A|\|_\infty = 3.94 \times 10^{-10}$, this estimate is better than that of Wilkinson.

Scaling and preconditioning

Several matrices that occur in applications are ill-conditioned if their order n is large. For example the famous Hilbert-matrix

$$H_n = \left[\frac{1}{i+j-1}\right]_{i,j=1}^{n} \tag{18.31}$$

has $\operatorname{cond}_2(H_n) \approx e^{3.5n}$, if $n \to \infty$. There exist $2n \times 2n$ matrices with integer entries that can be represented exactly in standard IEEE754 floating point arithmetic while their condition number is approximately 4×10^{32n}.

We have two main techniques to solve linear systems with large condition numbers.

Either we use multiple precision arithmetic or decrease the condition number. There are two known forms of decreasing the condition number.

1. Scaling We replace the linear system $Ax = b$ with the equation

$$(RAC)\,y = (Rb)\,, \tag{18.32}$$

where R and C are diagonal matrices.

We apply the Gauss method to this scaled system and get the solution y. The quantity $x = Cy$ defines the requested solution. If the condition number of the matrix RAC is smaller then we expect a smaller error in y and consequently in x. Various strategies are given to choose the scaling matrices R and C. One of the best known strategies is the ***balancing*** which forces every column and row of RAC to have approximately the same norm. For example, if

$$\hat{D} = \mathrm{diag}\left(\frac{1}{\left\|a_1^T\right\|_2}, \dots, \frac{1}{\left\|a_n^T\right\|_2}\right)$$

where a_i^T is the i^{th} row vector of A, the Euclidean norms of the rows of $\hat{D}A$ will be 1 and the estimate

$$\mathrm{cond}_2\left(\hat{D}A\right) \le \sqrt{n} \min_{D \in D_+} \mathrm{cond}_2\left(DA\right)$$

holds with $D_+ = \{\mathrm{diag}\,(d_1, \dots, d_n) \mid d_1, \dots, d_n > 0\}$. This means that $\hat{D}$ optimally scales the rows of A in an approximate sense.

The next example shows that the scaling may lead to bad results.

Example 18.11 Consider the matrix

$$A = \begin{bmatrix} \epsilon/2 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

for $0 < \epsilon \ll 1$. It is easy to show that $\mathrm{cond}_\infty\,(A) = 12$. Let

$$R = C = \begin{bmatrix} 2/\sqrt{\epsilon} & 0 & 0 \\ 0 & \sqrt{\epsilon}/2 & 0 \\ 0 & 0 & \sqrt{\epsilon}/2 \end{bmatrix}.$$

Then the scaled matrix

$$RAR = \begin{bmatrix} 2 & 1 & 1 \\ 1 & \epsilon/4 & \epsilon/4 \\ 1 & \epsilon/4 & \epsilon/2 \end{bmatrix},$$

has the condition number $\mathrm{cond}_\infty\,(RAR) = 32/\epsilon$, which a very large value for small ϵ.

2. Preconditioning The preconditioning is very close to scaling. We rewrite the linear system $Ax = b$ with the equivalent form

$$\widetilde{A}x = (MA)\,x = Mb = \tilde{b}, \tag{18.33}$$

where matrix M is such that $\mathrm{cond}\left(M^{-1}A\right)$ is smaller and $Mz = y$ is easily solvable.

The preconditioning is often used with iterative methods on linear systems with symmetric and positive definite matrices.

A posteriori error estimates

The a posteriori estimate of the error of an approximate solution is necessary to get some information on the reliability of the obtained result. There are plenty of such estimates. Here we show three estimates whose computational cost is $\Theta\left(n^2\right)$ flops. This cost is acceptable when comparing to the cost of direct or iterative methods ($\Theta\left(n^3\right)$ or $\Theta\left(n^2\right)$ per iteration step).

The estimate of the direct error with the residual error

Theorem 18.22 (Auchmuty). *Let $\widehat{x}$ be the approximate solution of $Ax = b$. Then*

$$\|x - \hat{x}\|_2 = \frac{c\,\|r(\hat{x})\|_2^2}{\left\|A^T r(\hat{x})\right\|_2},$$

where $c \geq 1$.

The error constant c depends on A and the direction of error vector $\hat{x} - x$. Furthermore

$$\frac{1}{2}\mathrm{cond}_2\left(A\right) \approx C_2\left(A\right) = \frac{1}{2}\left(\mathrm{cond}_2\left(A\right) + \frac{1}{\mathrm{cond}_2\left(A\right)}\right) \leq \mathrm{cond}_2\left(A\right).$$

The error constant c takes the upper value $C_2\left(A\right)$ only in exceptional cases. The computational experiments indicate that the average value of c grows slowly with the order of A and it depends more strongly on n than the condition number of A. The following experimental estimate

$$\left\|x - \widehat{x}\right\|_2 \lessapprox 0.5\dim\left(A\right)\left\|r\left(\widehat{x}\right)\right\|_2^2 / \left\|A^T r\left(\widehat{x}\right)\right\|_2 \tag{18.34}$$

seems to hold with a high degree of probability.

The LINPACK estimate of $\left\|A^{-1}\right\|$

The famous LINPACK program package uses the following process to estimate $\left\|A^{-1}\right\|$. We solve the linear systems $A^T y = d$ and $Aw = y$. Then the estimate of $\left\|A^{-1}\right\|$ is given by

$$\left\|A^{-1}\right\| \approx \frac{\|w\|}{\|y\|} \quad \left(\leq \left\|A^{-1}\right\|\right). \tag{18.35}$$

Since

$$\frac{\|w\|}{\|y\|} = \frac{\left\|A^{-1}\left(A^{-T}d\right)\right\|}{\left\|A^{-T}d\right\|},$$

we can interpret the process as an application of the power method of the eigenvalue problem. The estimate can be used with the $1-, 2-$ and ∞-norms. The entries of vector d are ± 1 possibly with random signs.

If the linear system $Ax = b$ is solved by the LU-method, then the solution of further linear systems costs $\Theta(n^2)$ flops per system. Thus the total cost of the LINPACK estimate remains small. Having the estimate $\left\|A^{-1}\right\|$ we can easily estimate cond(A) and the error of the approximate solution (cf. Theorem 18.16 or the thumb rule). We remark that several similar processes are known in the literature.

The Oettli-Prager estimate of the inverse error

We use the Oettli-Prager theorem in the following form. Let $r(\hat{x}) = A\hat{x} - b$ be the residual error, $E \in \mathbb{R}^{n\times n}$ and $f \in \mathbb{R}^n$ are given such that $E \geq 0$ and $f \geq 0$. Let

$$\omega = \max_i \frac{|r(\hat{x})_i|}{(E|\hat{x}| + f)_i},$$

where $0/0$ is set to 0-nak, $\rho/0$ is set to ∞, if $\rho \neq 0$. Symbol $(y)_i$ denotes the i^{th} component of the vector y. If $\omega \neq \infty$, then there exist a matrix ΔA and a vector Δb for which

$$|\Delta A| \leq \omega E, \quad |\Delta b| \leq \omega f$$

holds and

$$(A + \Delta A)\hat{x} = b + \Delta b.$$

Moreover ω is the smallest number for which ΔA and Δb exist with the above properties. The quantity ω measures the relative inverse error in terms of E and f. If for a given E, f and $\hat{x}$, the quantity ω is small, then the perturbed problem (and its solution) are close to the original problem (and its solution). In practice, the choice $E = |A|$ and $f = |b|$ is preferred

Iterative refinement

Denote by $\hat{x}$ the approximate solution of $Ax = b$ and let $r(y) = Ay - b$ be the residual error at the point y. The precision of the approximate solution $\hat{x}$ can be improved with the following method.

```
ITERATIVE-REFINEMENT($A, b, \hat{x}, tol$)

1  $k \leftarrow 1$
2  $x_1 \leftarrow \hat{x}$
3  $\hat{d} \leftarrow$ inf
4  while $= \|\hat{d}\| / \|x_k\| > tol$
5      do $r \leftarrow Ax_k - b$
6         Compute the approximate solution $\hat{d}$ of $Ad = r$ with the $LU$-method.
7         $x_{k+1} \leftarrow x_k - \hat{d}$
8         $k \leftarrow k + 1$
9  return $x_k$
```

There are other variants of this process. We can use other linear solvers instead of the LU-method.

Let η be the smallest bound of relative inverse error with

$$(A + \Delta A)\hat{x} = b + \Delta b, \quad |\Delta A| \leq \eta|A|, \quad |\Delta b| \leq \eta|b|.$$

Furthermore let

$$\sigma(A, x) = \max_k (|A||x|)_k / \min_k (|A||x|)_k, \quad \min_k (|A||x|)_k > 0.$$

Theorem 18.23 (Skeel). *If $k_r\left(A^{-1}\right)\sigma(A, x) \leq c_1 < 1/\epsilon_M$, then for sufficiently large k we have*

$$(A + \Delta A)x_k = b + \Delta b, \quad |\Delta A| \leq 4\eta\epsilon_M|A|, \quad |\Delta b| \leq 4\eta\epsilon_M|b|. \tag{18.36}$$

This result often holds after the first iteration, i.e. for $k = 2$. Jankowski and Wozniakowski investigated the iterative refinement for any method ϕ which produces an approximate solution $\widehat{x}$ with relative error less than 1. They showed that the iterative refinement improves the precision of the approximate solution even in single precision arithmetic and makes method ϕ to be weakly stable.

Exercises

18.2-1 Prove Theorem 18.7.

18.2-2 Consider the linear systems $Ax = b$ and $Bx = b$, where

$$A = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1/3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & -1/2 \\ 1/2 & 1/3 \end{bmatrix}$$

and $b \in \mathbb{R}^2$. Which equation is more sensitive to the perturbation of b? What should be the *relative error* of b in the more sensitive equation in order to get the solutions of both equations with the same precision?

18.2-3 Let $\chi = 3/2^{29}$, $\zeta = 2^{14}$ and

$$A = \begin{bmatrix} \chi\zeta & -\zeta & \zeta \\ \zeta^{-1} & \zeta^{-1} & 0 \\ \zeta^{-1} & -\chi\zeta^{-1} & \zeta^{-1} \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1+\varepsilon \\ 1 \end{bmatrix}.$$

Solve the linear systems $Ax = b$ for $\varepsilon = 10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}, 10^{-10}$. Explain the results.

18.2-4 Let A be a 10×10 matrix and choose the band matrix consisting of the main and the neighboring two subdiagonals of A as a preconditioning matrix. How much does the *condition number* of A improves if (i) A is a random matrix; (ii) A is a Hilbert matrix?

18.2-5 Let

$$A = \begin{bmatrix} 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \\ 1/4 & 1/5 & 1/6 \end{bmatrix},$$

and assume that ε is the common error bound of every component of $b \in \mathbb{R}^3$. Give the sharpest possible error bounds for the solution $[x_1, x_2, x_3]^T$ of the equation $Ax = b$ and for the sum $(x_1 + x_2 + x_3)$.

18.2-6 Consider the linear system $Ax = b$ with the approximate solution $\hat{x}$.

(i) Give an error bound for $\hat{x}$, if $(A + E)\hat{x} = b$ holds exactly and both A and $A + E$ is nonsingular.

(ii) Let

$$A = \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix}, \quad b = \begin{bmatrix} 25 \\ 18 \\ 24 \end{bmatrix}$$

and consider the solution of $Ax = b$. Give (if possible) a relative error bound for the entries of A such that the integer part of every solution component remains constant within the range of this relative error bound.

18.3. Eigenvalue problems

The set of complex n-vectors will be denoted by $\mathbb{C}^n$. Similarly, $\mathbb{C}^{m\times n}$ denotes the set of *complex $m \times n$ matrices.*

Definition 18.24 *Let $A \in \mathbb{C}^{n\times n}$ be an arbitrary matrix. The number $\lambda \in \mathbb{C}$ is the **eigenvalue** of A if there is vector $x \in \mathbb{C}^n$ ($x \neq 0$) such that*

$$Ax = \lambda x. \tag{18.37}$$

Vector x is called the (right) eigenvector of A that belongs to the eigenvalue λ.

Equation $Ax = \lambda x$ can be written in the equivalent form $(A-\lambda I)x = 0$, where I is the unit matrix of appropriate size. The latter homogeneous linear system has a nonzero solution x if and only if

$$\phi(\lambda) = \det(A - \lambda I) = \det\left(\begin{bmatrix} a_{11}-\lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22}-\lambda & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn}-\lambda \end{bmatrix}\right) = 0\,. \tag{18.38}$$

Equation (18.38) is called the ***characteristic equation*** of matrix A. The roots of this equation are the eigenvalues of matrix A. Expanding $\det(A - \lambda I)$ we obtain a polynomial of degree n:

$$\phi(\lambda) = (-1)^n(\lambda^n - p_1\lambda^{n-1} - \dots - p_{n-1}\lambda - p_n).$$

This polynomial called the ***characteristic polynomial*** of A. It follows from the fundamental theorem of algebra that any matrix $A \in \mathbb{C}^{n\times n}$ has exactly n eigenvalues with multiplicities. The eigenvalues may be complex or real. Therefore one needs to use complex arithmetic for eigenvalue calculations. If the matrix is real and the computations are done in real arithmetic, the complex eigenvalues and eigenvectors can be determined only with special techniques.

If $x \neq 0$ is an eigenvector, $t \in \mathbb{C}$ ($t \neq 0$), then tx is also eigenvector. The number of linearly independent eigenvectors that belong to an eigenvalue λ_k does not exceed the multiplicity of λ_k in the characteristic equation (18.38). The eigenvectors that belong to different eigenvalues are linearly independent.

The following results give estimates for the size and location of the eigenvalues.

Theorem 18.25 *Let λ be any eigenvalue of matrix A. The upper estimate $|\lambda| \leq \|A\|$ holds in any induced matrix norm.*

Theorem 18.26 (Gersgorin). *Let $A \in \mathbb{C}^{n\times n}$,*

$$r_i = \sum_{j=1, j\neq i}^{n} \left|a_{ij}\right| \quad (i = 1, \dots, n)$$

and

$$D_i = \{z \in \mathbf{C} \mid |z - a_{ii}| \leq r_i\} \quad (i = 1, \dots, n)\,.$$

Then for any eigenvalue λ of A we have $\lambda \in \cup_{i=1}^{n} D_i$.

For certain matrices the solution of the characteristic equation (18.38) is very easy. For example, if A is a triangular matrix, then its eigenvalues are entries of the main diagonal. In most cases however the computation of all eigenvalues and eigenvectors is a very difficult task. Those transformations of matrices that keeps the eigenvalues unchanged have practical significance for this problem. Later we see that the eigenvalue problem of transformed matrices is simpler.

Definition 18.27 *The matrices $A, B \in \mathbb{C}^{n\times n}$ are similar if there is a matrix T such that $B = T^{-1}AT$. The mapping $A \to T^{-1}AT$ is said to be* ***similarity transformation*** *of A.*

Theorem 18.28 *Assume that* $\det(T) \neq 0$. *Then the eigenvalues of A and $B = T^{-1}AT$ are the same. If x is the eigenvector of A, then $y = T^{-1}x$ is the eigenvector of B.*

Similar matrices have the same eigenvalues.

The difficulty of the eigenvalue problem also stems from the fact that the eigenvalues and eigenvectors are very sensitive (unstable) to changes in the matrix entries. The eigenvalues of A and the perturbed matrix $A + \delta A$ may differ from each other significantly. Besides the multiplicity of the eigenvalues may also change under perturbation. The following theorems and examples show the very sensitivity of the eigenvalue problem.

Theorem 18.29 (Ostrowski, Elsner). *For every eigenvalue λ_i of matrix $A \in \mathbb{C}^{n\times n}$ there exists an eigenvalue μ_k of the perturbed matrix $A + \delta A$ such that*

$$|\lambda_i - \mu_k| \leq (2n-1)\left(\|A\|_2 + \|A + \delta A\|_2\right)^{1-\frac{1}{n}} \|\delta A\|_2^{\frac{1}{n}} .$$

We can observe that the eigenvalues are changing continuously and the size of change is proportional to the n^{th} root of $\|\delta A\|_2$.

Example 18.12 Consider the following perturbed *Jordan matrix* of the size $r \times r$:

$$\begin{bmatrix} \mu & 1 & 0 & \dots & 0 \\ 0 & \mu & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \mu & 1 \\ \epsilon & 0 & \dots & 0 & \mu \end{bmatrix} .$$

The characteristic equation is $(\lambda - \mu)^r = \epsilon$, which gives the r different eigenvalues

$$\lambda_s = \mu + \epsilon^{1/r}\left(\cos(2s\pi/r) + i\sin(2s\pi/r)\right) \quad (s = 0, \dots, r-1)$$

instead of the original eigenvalue μ with multiplicity r. The size of change is $\epsilon^{1/r}$, which corresponds to Theorem (18.29). If $|\mu| \approx 1$, $r = 16$ and $\epsilon = \epsilon_M \approx 2.2204 \times 10^{-16}$, then the perturbation size of the eigenvalues is ≈ 0.1051. This is a significant change relative to the input perturbation ϵ.

For special matrices and perturbations we may have much better perturbation bounds.

Theorem 18.30 (Bauer, Fike). *Assume that $A \in \mathbb{C}^{n\times n}$ is* ***diagonalizable*** *, that is a matrix X exists such that $X^{-1}AX = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$. Denote μ an eigenvalue of $A + \delta A$. Then*

$$\min_{1\leq i\leq n} |\lambda_i - \mu| \leq \operatorname{cond}_2(X)\,\|\delta A\|_2 . \tag{18.39}$$

This result is better than that of Ostrowski and Elsner. Nevertheless $\mathrm{cond}_2(X)$, which is generally unknown, can be very big.

The eigenvalues are continuous functions of the matrix entries. This is also true for the normalized eigenvectors if the eigenvalues are simple. The following example shows that this property does not hold for multiple eigenvalues.

Example 18.13 Let

$$A(t) = \begin{bmatrix} 1 + t\cos(2/t) & -t\sin(2/t) \\ -t\sin(2/t) & 1 - t\cos(2/t) \end{bmatrix} \quad (t \neq 0).$$

The eigenvalues of $A(t)$ are $\lambda_1 = 1 + t$ and $\lambda_2 = 1 - t$. Vector $[\sin(1/t), \cos(1/t)]^T$ is the eigenvector belonging to λ_1. Vector $[\cos(1/t), -\sin(1/t)]^T$ is the eigenvector belonging to λ_2. If $t \to 0$, then

$$A(t) \to I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \lambda_1, \lambda_2 \to 1,$$

while the eigenvectors do not have limit.

We study the numerical solution of the eigenvalue problem in the next section. Unfortunately it is very difficult to estimate the goodness of numerical approximations. From the fact that $Ax - \lambda x = 0$ holds with a certain error we cannot conclude anything in general.

Example 18.14 Consider the matrix

$$A(\epsilon) = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix},$$

where $\epsilon \approx 0$ is small. The eigenvalues of $A(\epsilon)$ are $1 \pm \sqrt{\epsilon}$, while the corresponding eigenvectors are $[1, \pm\sqrt{\epsilon}]^T$. Let $\mu = 1$ be an approximation of the eigenvalues and let $x = [1, 0]^T$ be the approximate eigenvector. Then

$$\|Ax - \mu x\|_2 = \left\| \begin{bmatrix} 0 \\ \epsilon \end{bmatrix} \right\|_2 = \epsilon.$$

If $\epsilon = 10^{-10}$, then the residual error under estimate the true error 10^{-5} by five order.

Remark 18.31 *We can define the* ***condition number of eigenvalues*** *for simple eigenvalues:*

$$\nu(\lambda_1) \approx \frac{\|x\|_2 \, \|y\|_2}{\left|x^H y\right|},$$

where x and y are the right and left eigenvectors, respectively. For multiple eigenvalues the condition number is not finite.

18.3.1. Iterative solutions of the eigenvalue problem

We investigate only the real eigenvalues and eigenvectors of real matrices. The methods under consideration can be extended to the complex case with appropriate modifications.

The power method

This method is due to von Mieses. Assume that $A \in \mathbb{R}^{n\times n}$ has exactly n different real eigenvalues. Then the eigenvectors $x_1, \dots, x_n$ belonging to the corresponding eigenvalues $\lambda_1, \dots, \lambda_n$ are linearly independent. Assume that the eigenvalues satisfy the condition

$$|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$$

and let $v^{(0)} \in \mathbb{R}^n$ be a given vector. This vector is a unique linear combination of the eigenvectors, that is $v^{(0)} = \alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_n x_n$. Assume that $\alpha_1 \neq 0$ and compute the sequence $v^{(k)} = Av^{(k-1)} = A^k v^{(0)}$ $(k = 1, 2, \dots)$. The initial assumptions imply that

$$\begin{aligned} v^{(k)} \;=\; Av^{(k-1)} &= A(\alpha_1 \lambda_1^{k-1} x_1 + \alpha_2 \lambda_2^{k-1} x_2 + \dots + \alpha_n \lambda_n^{k-1} x_n) \\ &= \alpha_1 \lambda_1^k x_1 + \alpha_2 \lambda_2^k x_2 + \dots + \alpha_n \lambda_n^k x_n \\ &= \lambda_1^k \left(\alpha_1 x_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1}\right)^k x_n \right). \end{aligned}$$

Let $y \in \mathbb{R}^n$ be an arbitrary vector such that $y^T x_1 \neq 0$. Then

$$\frac{y^T A v^{(k)}}{y^T v^{(k)}} = \frac{y^T v^{(k+1)}}{y^T v^{(k)}} = \frac{\lambda_1^{k+1}\left(\alpha_1 y^T x_1 + \sum_{i=2}^n \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^{k+1} y^T x_i\right)}{\lambda_1^k \left(\alpha_1 y^T x_1 + \sum_{i=2}^n \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^k y^T x_i\right)} \to \lambda_1 \; .$$

Given the initial vector $v^{(0)} \in \mathbb{R}^n$, the power method has the following form.

```
Power-Method($A, v^{(0)}$)
1  $k \leftarrow 0$
2  while exit condition = FALSE
3        do $k \leftarrow k + 1$
4           $z^{(k)} \leftarrow Av^{(k-1)}$
5           Select vector $y$ such that $y^T v^{(k-1)} \neq 0$
6           $\gamma_k \leftarrow y^T z^{(k)} / y^T v^{(k-1)}$
7           $v^{(k)} \leftarrow z^{(k)} / \|z^{(k)}\|_\infty$
8  return $\gamma_k, v^{(k)}$
```

It is clear that

$$v^{(k)} \to x_1, \quad \gamma_k \to \lambda_1 .$$

The convergence $v^{(k)} \to x_1$ here means that $\left(v^{(k)}, x_1\right)_\measuredangle \to 0$, that is the action line of $v^{(k)}$ tends to the action line of x_1. There are various strategies to select y. We can select $y = e_i$, where i is defined by $\left|v_i^{(k)}\right| = \left\|v^{(k)}\right\|_\infty$. If we select $y = v^{(k-1)}$, then $\gamma_k = v^{(k-1)T} A v^{(k-1)} / \left(v^{(k-1)T} v^{(k-1)}\right)$ will be identical with the *Rayleigh quotient* $\mathbf{R}\left(v^{(k-1)}\right)$. This choice gives an approximation of λ_1 that have the minimal residual norm (Example 18.14. shows that this choice is not necessarily the best option).

The speed of convergence depends on the quotient $|\lambda_2/\lambda_1|$. The method is very sensitive to the choice of the initial vector $v^{(0)}$. If $\alpha_1 = 0$, then the process does not converge to the dominant eigenvalue λ_1. For certain matrix classes the power method converges with probability 1 if the initial vector $v^{(0)}$ is randomly chosen. In case of complex eigenvalues or multiple λ_1 we have to use modifications of the algorithm. The speed of convergence can be accelerated if the method is applied to the shifted matrix $A-\sigma I$, where σ is an appropriately chosen number. The shifted matrix $A-\sigma I$ has the eigenvalues $\lambda_1-\sigma, \lambda_2-\sigma, \ldots, \lambda_n-\sigma$ and the corresponding convergence factor $|\lambda_2-\sigma| / |\lambda_1-\sigma|$. The latter quotient can be made smaller than $|\lambda_2/\lambda_1|$ with the proper selection of σ.

The usual exit condition of the power method is

$$\|E_k\|_2 = \frac{\|r_k\|_2}{\left\|v^{(k)}\right\|_2} = \frac{\left\|Av^{(k)} - \gamma_k v^{(k)}\right\|_2}{\left\|v^{(k)}\right\|_2} \le \epsilon.$$

If we simultaneously apply the power method to the transposed matrix A^T and $w_k = \left(A^T\right)^k w_0$, then the quantity

$$\nu(\lambda_1) \approx \frac{\left\|w^{(k)}\right\|_2 \left\|v^{(k)}\right\|_2}{\left|w_k^T v_k\right|}$$

gives an estimate for the condition number of λ_1 (see Remark 18.31). In such a case we use the exit condition

$$\nu(\lambda_1)\,\|E_k\|_2 \le \epsilon.$$

The power method is very useful for large sparse matrices. It is often used to determine the largest and the smallest eigenvalue. We can approximate the smallest eigenvalue as follows. The eigenvalues of A^{-1} are $1/\lambda_1, \ldots, 1/\lambda_n$. The eigenvalue $1/\lambda_n$ will be the eigenvalue with the largest modulus. We can approximate this value by applying the power method to A^{-1}. This requires only a small modification of the algorithm. We replace line 4. with the following:

$$\text{Solve equation } Az^{(k)} = v^{(k-1)} \text{ for } z^{(k)}$$

The modified algorithm is called the ***inverse power method***. It is clear that $\gamma_k \to 1/\lambda_n$ and $v^{(k)} \rightharpoonup x_n$ hold under appropriate conditions. If we use the LU-method to solve $Az^{(k)} = v^{(k-1)}$, we can avoid the inversion of A.

If the inverse power method is applied to the shifted matrix $A-\mu I$, then the eigenvalues of $(A-\mu I)^{-1}$ are $(\lambda_i-\mu)^{-1}$. If μ approaches, say, to λ_t, then $\lambda_i-\mu \to \lambda_i-\lambda_t$. Hence the inequality

$$|\lambda_t-\mu|^{-1} > |\lambda_i-\mu|^{-1} \qquad (i \ne t)$$

holds for the eigenvalues of the shifted matrix. The speed of convergence is determined by the quotient

$$q = |\lambda_t-\mu| \,/\, \{\max|\lambda_i-\mu|\}\,.$$

If μ is close enough to λ_t, then q is very small and the inverse power iteration converges very fast. This property can be exploited in the calculation of approximate eigenvectors if an approximate eigenvalue, say μ, is known. Assuming that $\det(A-\mu I) \ne 0$, we apply the inverse power method to the shifted matrix $A-\mu I$. In spite of the fact that matrix $A-\mu I$ is nearly singular and the linear equation $(A-\mu I)\,z^{(k)} = v^{(k)}$ cannot be solved with high precision, the algorithm gives very often good approximations of the eigenvectors.

Finally we note that in principle the von Mieses method can be modified to determine all eigenvalues and eigenvectors.

Orthogonalization processes
We need the following definition and theorem.

Definition 18.32 *The matrix $Q \in \mathbb{R}^{n\times n}$ is said to be **orthogonal** if $Q^T Q = I$.*

Theorem 18.33 (QR-decomposition). *Every matrix $A \in \mathbb{R}^{n\times m}$ having linearly independent column vectors can be decomposed in the product form $A = QR$, where Q is orthogonal and R is upper triangular matrix.*

We note that the QR-decomposition can be applied for solving linear systems of equations, similarly to the LU-decomposition. If the QR-decomposition of A is known, then the equation $Ax = QRx = b$ can be written in the equivalent form $Rx = Q^T b$. Thus we have to solve only an upper triangular linear system.

There are several methods to determine the QR-decomposition of a matrix. In practice the Givens-, the Householder- and the MGS-methods are used.

The MGS (Modified Gram-Schmidt) method is a stabilized, but algebraically equivalent version of the classical Gram-Schmidt orthogonalization algorithm. The basic problem is the following: We seek for an *orthonormal basis* $\left\{q_j\right\}_{j=1}^m$ of the subspace

$$\mathcal{L}\{a_1,\dots,a_m\} = \left\{\sum_{j=1}^m \lambda_j a_j \mid \lambda_j \in \mathbb{R},\ j = 1,\dots,m\right\},$$

where $a_1,\dots,a_m \in \mathbb{R}^n$ $(m \le n)$ are linearly independent vectors. That is we determine the linearly independent vectors $q_1,\dots,q_m$ such that

$$q_i^T q_j = 0 \ \ (i \neq j)\,, \|q_i\|_2 = 1 \ \ (i = 1,\dots,m)$$

and

$$\mathcal{L}\{a_1,\dots,a_m\} = \mathcal{L}\{q_1,\dots,q_m\}\ .$$

The basic idea of the *classical Gram-Schmidt method* is the following:

Let $r_{11} = \|a_1\|_2$ and $q_1 = a_1/r_{11}$. Assume that vectors $q_1,\dots,q_{k-1}$ are already computed and orthonormal. Assume that vector $\tilde{q}_k = a_k - \sum_{j=1}^{k-1} r_{jk}q_j$ is such that $\tilde{q}_k \perp q_i$, that is $\tilde{q}_k^T q_i = a_k^T q_i - \sum_{j=1}^{k-1} r_{jk}q_j^T q_i = 0$ holds for $i = 1,\dots,k-1$. Since $q_1,\dots,q_{k-1}$ are orthonormal, $q_j^T q_i = 0$ $(i \neq j)$ and $r_{ik} = a_k^T q_i \quad (i = 1,\dots,k-1)$. After normalization we obtain $q_k = \tilde{q}_k / \|\tilde{q}_k\|_2$.

The algorithm is formalized as follows.

```
CGS-ORTHOGONALIZATION($m, a_1, \dots, a_m$)
1  for $k \leftarrow 1$ to $m$
2     do for $i \leftarrow 1$ to $k - 1$
3           do $r_{ik} \leftarrow a_k^T a_i$
4              $a_k \leftarrow a_k - r_{ik}a_i$
5        $r_{kk} \leftarrow \|a_k\|_2$
6        $a_k \leftarrow a_k / r_{kk}$
7  return $a_1, \dots, a_m$
```

The algorithm overwrites vectors a_i by the orthonormal vectors q_i. The connection with the QR-decomposition follows from the relation $a_k = \sum_{j=1}^{k-1} r_{jk}q_j + r_{kk}q_k$. Since

$$\begin{aligned} a_1 &= q_1 r_{11}, \\ a_2 &= q_1 r_{12} + q_2 r_{22}, \\ a_3 &= q_1 r_{13} + q_2 r_{23} + q_3 r_{33}, \\ &\vdots \\ a_m &= q_1 r_{1m} + q_2 r_{2m} + \ldots + q_m r_{mm}, \end{aligned}$$

we can write that

$$A = [a_1, \ldots, a_m] = \underbrace{[q_1, \ldots, q_m]}_{Q} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & \ldots & r_{1m} \\ 0 & r_{22} & r_{23} & \ldots & r_{2m} \\ 0 & 0 & r_{33} & \ldots & r_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & r_{mm} \end{bmatrix}}_{R} = QR\,.$$

The *numerically stable MGS method* is given in the following form

```
MGS-orthogonalization($m, a_1, \ldots, a_m$)
1  for $k \leftarrow 1$ to $m$
2     do $r_{kk} \leftarrow \|a_k\|_2$
3        $a_k \leftarrow a_k / r_{kk}$
4        for $j \leftarrow k+1$ to $m$
5           do $r_{kj} \leftarrow a_j^T a_k$
6              $a_j \leftarrow a_j - r_{kj} a_k$
7  return $a_1, \ldots, a_m$
```

The algorithm overwrites vectors a_i by the orthonormal vectors q_i. The MGS method is more stable than the CGS algorithm. Björck proved that for $m = n$ the computed matrix $\hat{Q}$ satisfies

$$\hat{Q}^T \hat{Q} = I + E, \qquad \|E\|_2 \cong \mathrm{cond}(A)\, u,$$

where u is the unit roundoff.

The QR-method

Today the QR-method is the most important numerical algorithm to compute all eigenvalues of a general matrix. It can be shown that the QR-method is a generalization of the power method. The basic idea of the method is the following: Starting from $A_1 = A$ we compute the sequence $A_{k+1} = Q_k^{-1} A_k Q_k = Q_k^T A_k Q_k$, where Q_k is orthogonal, A_{k+1} is orthogonally similar to A_k (A) and the lower triangular part of A_k tends to a diagonal matrix, whose entries will be the eigenvalues of A. Here Q_k is the orthogonal factor of the QR-decomposition $A_k = Q_k R_k$. Therefore $A_{k+1} = Q_k^T (Q_k R_k) Q_k = R_k Q_k$. The basic algorithm is given in the following form.

QR-METHOD(A)

```
1 $k \leftarrow 1$
2 $A_1 \leftarrow A$
3 while exit condition=FALSE
4         do Compute the $QR$-decomposition $A_k = Q_k R_k$
5             $A_{k+1} \leftarrow R_k Q_k$
6             $k \leftarrow k + 1$
7 return $A_k$
```

The following result holds.

Theorem 18.34 (Parlett). *If the matrix A is diagonalizable, $X^{-1}AX = diag(\lambda_1, \lambda_2, \dots, \lambda_n)$, the eigenvalues satisfy*

$$|\lambda_1| > |\lambda_2| > \dots > |\lambda_n| > 0$$

and X has an LU-decomposition, then the lower triangular part of A_k converges to a diagonal matrix whose entries are the eigenvalues of A.

In general, matrices A_k do not necessarily converge to a given matrix. If A has p eigenvalues of the same modulus, the form of matrices A_k converge to the form

$$\begin{bmatrix} \times & & & & & & & \times \\ 0 & \ddots & & & & & & \\ & & \times & & & & & \\ 0 & & 0 & * & \cdots & * & & \\ & & & \vdots & & \vdots & & \\ & & & * & \cdots & * & & \\ 0 & & & & & 0 & \times & \\ & & & & & & \ddots & \\ 0 & & & & & & 0 & \times \end{bmatrix}, \qquad (18.40)$$

where the entries of the submatrix denoted by $*$ do not converge. However the eigenvalues of this submatrix will converge. This submatrix can be identified and properly handled. A real matrix may have real and complex eigenvalues. If there is a complex eigenvalues, than there is a corresponding conjugate eigenvalue as well. For pairs of complex conjugated eigenvalues p is at least 2. Hence the sequence A_k will show this phenomenon .

The QR-decomposition is very expensive. Its cost is $\Theta(n^3)$ flops for general $n \times n$ matrices. If A has upper Hessenberg form, the cost of QR-decomposition is $\Theta(n^2)$ flops.

Definition 18.35 *The matrix $A \in \mathbb{R}^{n \times n}$ has **upper Hessenberg form**, , if*

$$A = \begin{bmatrix} a_{11} & & & \dots & & a_{1n} \\ a_{21} & & & & & \\ 0 & a_{32} & & & & \vdots \\ \vdots & 0 & \ddots & & & \\ & & \ddots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & & \dots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}.$$

The following theorem guarantees that if A has upper Hessenberg form, then every A_k of the QR-method has also upper Hessenberg form.

Theorem 18.36 *If A has upper Hessenberg form and $A = QR$, then RQ has also upper Hessenberg form.*

We can transform a matrix A to a similar matrix of upper Hessenberg form in many ways. One of the cheapest ways, that costs about $5/6n^3$ flops, is based on the Gauss elimination method. Considering the advantages of the upper Hessenberg form the efficient implementation of the QR-method requires first the similarity transformation of A to upper Hessenberg form.

The convergence of the QR-method, similarly to the power method, depends on the quotients $|\lambda_{i+1}/\lambda_i|$. The eigenvalues of the shifted matrix $A-\sigma I$ are $\lambda_1-\sigma, \lambda_2-\sigma, \ldots, \lambda_n-\sigma$. The corresponding eigenvalue ratios are $|(\lambda_{i+1}-\sigma)\,/\,(\lambda_i-\sigma)|$. A proper selection of σ can fasten the convergence.

The usual form of the QR-method includes the transformation to upper Hessenberg form and the shifting.

```
ShiftedQR-Method(A)
1  $H_1 \leftarrow U^{-1}AU$   ($H_1$ is of upper Hessenberg form)
2  $k \leftarrow 1$
3  while exit condition=FALSE
4      do Compute the QR-decomposition $H_k - \sigma_k I = Q_k R_k$
5         $H_{k+1} \leftarrow R_k Q_k + \sigma_k I$
6         $k \leftarrow k + 1$
7  return $H_k$
```

In practice the QR-method is used in shifted form. There are various strategies to select σ_i. The most often used selection is given by $\sigma_k = h_{nn}^{(k)}$ $\left(H_k = \left[h_{ij}^{(k)}\right]_{i,j=1}^n\right)$.

The eigenvectors of A can also be determined by the QR-method. For this we refer to the literature.

Exercises

18.3-1 Apply the *power method* to the matrix $A = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$ with the initial vector $v^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. What is the result of the 20th step?

18.3-2 Apply the *power method*, the *inverse power method* and the QR-method to the matrix

$$\begin{bmatrix} -4 & -3 & -7 \\ 2 & 3 & 2 \\ 4 & 2 & 7 \end{bmatrix}.$$

18.3-3 Apply the shifted QR-method to the matrix of the previous exercise with the choice $\sigma_i = \sigma$ (σ is fixed).

18.4. Numerical program libraries and software tools

We have plenty of devices and tools that support efficient coding and implementation of numerical algorithms. One aim of such developments is to free the programmers from writing the programs of frequently occurring problems. This is usually done by writing safe, reliable and standardized routines that can be downloaded from (public) program libraries. We just mention the LINPACK, EISPACK, LAPACK, VISUAL NUMERICS (former IMSL) and NAG libraries. Another way of developments is to produce software that work as a programming language and makes the programming very easy. Such software systems are the MATLAB and the SciLab.

18.4.1. Standard linear algebra subroutines

The main purpose of the BLAS (Basic Linear Algebra Subprograms) programs is the standardization and efficient implementation the most frequent matrix-vector operations. Although the BLAS routines were published in FORTRAN they can be accessed in optimized machine code form as well. The BLAS routines have three levels:

- BLAS 1 (1979),
- BLAS 2 (1988),
- BLAS 3 (1989).

These levels corresponds to the computation cost of the implemented matrix operations. The BLAS routines are considered as the best implementations of the given matrix operations. The selection of the levels and individual BLAS routines strongly influence the efficiency of the program. A sparse version of BLAS also exists.

We note that the BLAS 3 routines were developed mainly for block parallel algorithms. The standard linear algebra packages LINPACK, EISPACK and LAPACK are built from BLAS routines. The parallel versions can be found in the SCALAPACK package. These programs can be found in the public NETLIB library:

`http:/www.netlib.org/index.html`

BLAS 1 routines

Let $\alpha \in \mathbb{R}$, $x, y \in \mathbb{R}^n$. The BLAS 1 routines are the programs of the most important vector operations ($z = \alpha x$, $z = x+y$, $dot = x^T y$), the computation of $\|x\|_2$, the swapping of variables, rotations and the *saxpy* operation which is defined by

$$z = \alpha x + y.$$

The word *saxpy* means that „scalar alpha x plus y". The *saxpy* operation is implemented in the following way.

```
Saxpy($\alpha, x, y$)
1  $n \leftarrow elements[x]$
2  for $i \leftarrow 1$ to $n$
3      do $z[i] = \alpha x[i] + y[i]$
4  return $z$
```

The *saxpy* is a software driven operation. The cost of BLAS 1 routines is $\Theta(n)$ flops.

BLAS 2 routines

The matrix-vector operations of BLAS 2 requires $\Theta\left(n^2\right)$ flops. These operations are $y = \alpha Ax + \beta y$, $y = Ax$, $y = A^{-1}x$, $y = A^T x$, $A \leftarrow A + xy^T$ and their variants. Certain operations work only with triangular matrices. We analyze two operations in detail. The „outer or dyadic product” update

$$A \leftarrow A + xy^T \quad (A \in \mathbb{R}^{m\times n}, x \in \mathbb{R}^m, y \in \mathbb{R}^n)$$

can be implemented in two ways.

The rowwise or „ij” variant:

```
Outer-Product-Update-Version „ij” $(A, x, y)$
1  $m \leftarrow rows[A]$
2  for $i \leftarrow 1$ to $m$
3      do $A[i, :] \leftarrow A[i, :] + x[i]\, y^T$
4  return $A$
```

The notation „:” denotes all allowed indices. In our case this means the indices $1 \leq j \leq n$. Thus $A[i, :]$ denotes the ith row of matrix A.

The columnwise or „ji” variant:

```
Outer-Product-Update-Version „ji”$(A, x, y)$
1  $n \leftarrow columns[A]$
2  for $j \leftarrow 1$ to $n$
3      do $A[:, j] \leftarrow A[:, j] + y[j]\, x$
4  return $A$
```

Here $A[:, j]$ denotes the jth column of matrix A. Observe that both variants are based on the saxpy operation.

The *gaxpy* operation is defined by

$$z = y + Ax \quad \left(x \in \mathbb{R}^n, y \in \mathbb{R}^m, A \in \mathbb{R}^{m\times n}\right).$$

The word *gaxpy* means that „general A x plus y”. The *gaxpy* operation is also software driven and implemented in the following way:

```
Gaxpy$(A, x, y)$
1  $n \leftarrow columns[A]$
2  $z \leftarrow y$
3  for $j \leftarrow 1$ to $n$
4      do $z \leftarrow z + x[j]\, A[:, j]$
5  return $z$
```

Observe that the computation is done columnwise and the *gaxpy* operation is essentially a generalized *saxpy*.

BLAS 3 routines

These routines are the implementations of $\Theta\left(n^3\right)$ matrix-matrix and matrix-vector operations such as the operations $C \leftarrow \alpha AB + \beta C$, $C \leftarrow \alpha AB^T + \beta C$, $B \leftarrow \alpha T^{-1}B$ (T is upper triangular) and their variants. BLAS 3 operations can be implemented in several forms. For example, the matrix product $C = AB$ can be implemented at least in three ways. Let $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{r \times n}$.

```
MATRIX-PRODUCT-DOT-VERSION($A, B$)
$m \leftarrow rows[A]$
$r \leftarrow columns[A]$
$n \leftarrow columns[B]$
$C[1:m, 1:n] \leftarrow 0$
for $i \leftarrow 1$ to $m$
    do for $j \leftarrow 1$ to $n$
        do for $k \leftarrow 1$ to $r$
            do $C[i,j] \leftarrow C[i,j] + A[i,k]\,B[k,j]$
return $C$
```

This algorithm computes c_{ij} as the dot (inner) product of the ith row of A and the jth column of B. This corresponds to the original definition of matrix products.

Now let A, B and C be partitioned columnwise as follows

$$\begin{aligned} A &= [a_1, \ldots, a_r] \quad (a_i \in \mathbb{R}^m)\,, \\ B &= [b_1, \ldots, b_n] \quad (b_i \in \mathbb{R}^r)\,, \\ C &= [c_1, \ldots, c_n] \quad (c_i \in \mathbb{R}^m)\,. \end{aligned}$$

Then we can write c_j as the linear combination of the columns of A, that is

$$c_j = \sum_{k=1}^{r} b_{kj} a_k \quad (j = 1, \ldots, n)\,.$$

Hence the product can be implemented with *saxpy* operations.

```
MATRIX-PRODUCT-GAXPY-VARIANT($A, B$)
$m \leftarrow rows[A]$
$r \leftarrow columns[A]$
$n \leftarrow columns[B]$
$C[1:m, 1:n] \leftarrow 0$
for $j \leftarrow 1$ to $n$
    do for $k \leftarrow 1$ to $r$
        do for $i \leftarrow 1$ to $m$
            do $C[i,j] \leftarrow C[i,j] + A[i,k]\,B[k,j]$
return $C$
```

The following equivalent form of the „*jki*"-algorithm shows that it is indeed a *gaxpy* based process.

```
Matrix-Product-with-Gaxpy-Call(A, B)
1 $m \leftarrow rows[A]$
2 $n \leftarrow columns[B]$
3 $C[1:m, 1:n] \leftarrow 0$
4 for $j \leftarrow 1$ to $n$
5     do $C[:, j] = gaxpy(A, B[:, j], C[:, j])$
6 return $C$
```

Consider now the partitions $A = [a_1, \dots, a_r]$ $(a_i \in R^m)$ and

$$B = \begin{bmatrix} b_1^T \\ \vdots \\ b_r^T \end{bmatrix} \quad (b_i \in R^n)\,.$$

Then $C = AB = \sum_{k=1}^{r} a_k b_k^T$.

```
Matrix-Product-Outer-Product-Variant(A, B)
1 $m \leftarrow rows[A]$
2 $r \leftarrow columns[A]$
3 $n \leftarrow columns[B]$
4 $C[1:m, 1:n] = 0$
5 for $k \leftarrow 1$ to $r$
6     do for $j \leftarrow 1$ to $n$
7            do for $i \leftarrow 1$ to $m$
8                   $C[i, j] \leftarrow C[i, j] + A[i, k]\, B[k, j]$
9 return $C$
```

The inner loop realizes a *saxpy* operation: it gives the multiple of a_k to the jth column of matrix C.

18.4.2. Mathematical software

These are those programming tools that help easy programming in concise (possibly mathematical) form within an integrated program development system. Such systems were developed primarily for solving mathematical problems. By now they have been extended so that they can be applied in many other fields. For example, Nokia uses MATLAB in the testing and quality control of mobile phones. We give a short review on MATLAB in the next section. We also mention the widely used MAPLE, DERIVE and MATEMATICA systems.

The MATLAB system

The MATLAB software was named after the expression MATrix LABoratory. The name indicates that the matrix operations are very easy to make. The initial versions of MATLAB had only one data type: the complex matrix. In the later versions high dimension arrays, cells, records and objects also appeared. The MATLAB can be learned quite easily and even a beginner can write programs for relatively complicated problems.

The coding of matrix operations is similar to their standard mathematical form. For example if A and B are two matrices of the same size, then their sum is given by the command $C = A + B$. As a programming language the MATLAB contains only four control structures known from other programing languages:

– the simple statement Z =*expression*,

– the if statement of the form

if *expression, commands* {**else/elseif** *commands*} **end**,

– the for loop of the form

for *the values of the loop variable, commands* **end**

– the while loop of the form

while *expression, commands* **end**.

The MATLAB has an extremely large number of built in functions that help efficient programming. We mention the following ones as a sample.

– max(A) selects the maximum element in every column of A,

– $[v, s]$ =eig(A) returns the approximate eigenvalues and eigenvectors of A,

– The command $A\backslash b$ returns the numerical solution of the linear system $Ax = b$.

The entrywise operations and partitioning of matrices can be done very efficiently in MATLAB. For example, the statement

$$A([2,3],:) = 1./A([3,2],:)$$

exchange the second and third rows of A while it takes the reciprocal of each element.

The above examples only illustrate the possibilities and easy programming of MATLAB. These examples require much more programming effort in other languages, say e.g. in PASCAL. The built in functions of MATLAB can be easily supplemented by other programs.

The higher number versions of MATLAB include more and more functions and special libraries (tool boxes) to solve special problems such as optimization, statistics and so on.

There is a built in automatic technique to store and handle sparse matrices that makes the MATLAB competitive in solving large computational problems. The recent versions of MATLAB offer very rich graphic capabilities as well. There is an extra interval arithmetic package that can be downloaded from the WEB site

```
http:/www.ti3.tu-harburg.de\%7Erump\intlab
```

There is a possibility to build certain C and FORTRAN programs into MATLAB. Finally we mention that the system has an extremely well written help system.

Problems

18-1. Without overflow

Write a MATLAB program that computes the norm $\|x\|_2 = \left(\sum_{i=1}^n x_i^2\right)^{1/2}$ without *overflow* in all cases when the result does not make overflow. It is also required that the error of the final result can not be greater than that of the original formula.

18-2. Estimate

Equation $x^3 - 3.330000x^2 + 3.686300x - 1.356531 = 0$ has the solution $x_1 = 1.01$. The perturbed equation $x^3 - 3.3300x^2 + 3.6863x - 1.3565 = 0$ has the solutions $y_1,\ y_2,\ y_3$. Give an estimate for the perturbation $\min_i |x_1 - y_i|$.

18-3. Double word length

Consider an arithmetic system that has double word length such that every number represented with $2t$ digits are stored in two t digit word. Assume that the computer can only add numbers with t digits. Furthermore assume that the machine can recognize overflow.
(i) Find an algorithm that add two positive numbers of $2t$ digit length.
(ii) If the representation of numbers requires the sign digit for all numbers, then modify algorithm (i) so that it can add negative and positive numbers both of the same sign. We can assume that the sum does not overflow.

18-4. Auchmuty theorem

Write a MATLAB program for the Auchmuty error estimate (see Theorem 18.22) and perform the following numerical testing.
(i) Solve the linear systems $Ax = b_i$, where $A \in \mathbb{R}^{n\times n}$ is a given matrix, $b_i = Ay_i$, $y_i \in \mathbb{R}^n$ ($i = 1,\ldots,N$) are random vectors such that $\|y_i\|_\infty \le \beta$. Compare the true errors $\|\widetilde{x}_i - y_i\|$, ($i = 1,\ldots,N$) and the estimated errors $EST_i = \|r(\widetilde{x}_i)\|_2^2 \,/\, \left\|A^T r(\widetilde{x}_i)\right\|_2$, where $\widetilde{x}_i$ is the approximate solution of $Ax = b_i$. What is the minimum, maximum and average of numbers c_i? Use graphic for the presentation of the results. Suggested values are $n \le 200$, $\beta = 200$ and $N = 40$.
(ii) Analyze the effect of condition number and size.
(iii) Repeat problems (i) and (ii) using LINPACK and BLAS.

18-5. Hilbert matrix

Consider the linear system $Ax = b$, where $b = [1,1,1,1]^T$ and A is the fourth order Hilbert matrix, that is $a_{i,j} = 1/(i+j)$. A is ill-conditioned. The inverse of A is approximated by

$$B = \begin{bmatrix} 202 & -1212 & 2121 & -1131 \\ -1212 & 8181 & -15271 & 8484 \\ 2121 & -15271 & 29694 & -16968 \\ -1131 & 8484 & -16968 & 9898 \end{bmatrix}.$$

Thus an x_0 approximation of the true solution x is given by $x_0 = Bb$. Although the true solution is also integer x_0 is not an acceptable approximation. Apply the *iterative refinement* with B instead of A^{-1} to find an acceptable integer solution.

18-6. Consistent norm

Let $\|A\|$ be a *consistent norm* and consider the linear system $Ax = b$
(i) Prove that if $A + \Delta A$ is singular, then $\mathrm{cond}(A) \ge \|A\| \,/\, \|\Delta A\|$.
(ii) Show that for the 2-norm equality holds in (i), if $\Delta A = -bx^T/(b^t x)$ and $\left\|A^{-1}\right\|_2 \|b\|_2 = \left\|A^{-1}b\right\|_2$.
(iii) Using the result of (i) give a lower bound to $\mathrm{cond}_\infty(A)$, if

$$A = \begin{bmatrix} 1 & -1 & 1 \\ -1 & \varepsilon & \varepsilon \\ 1 & \varepsilon & \varepsilon \end{bmatrix}.$$

18-7. Cholesky-method

Use the Cholesky-method to solve the linear system $Ax = b$, where

$$A = \begin{bmatrix} 5.5 & 0 & 0 & 0 & 0 & 3.5 \\ 0 & 5.5 & 0 & 0 & 0 & 1.5 \\ 0 & 0 & 6.25 & 0 & 3.75 & 0 \\ 0 & 0 & 0 & 5.5 & 0 & 0.5 \\ 0 & 0 & 3.75 & 0 & 6.25 & 0 \\ 3.5 & 1.5 & 0 & 0.5 & 0 & 5.5 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Also give the exact *Cholesky-decomposition* $A = LL^T$ and the true solution of $Ax = b$. The approximate Cholesky-factor $\widetilde{L}$ satisfies the relation $\tilde{L}\tilde{L}^T = A + F$. It can proved that in a *floating point arithmetic* with t-digit mantissa and base β the entries of F satisfy the inequality $\left|f_{i,j}\right| \leq e_{i,j}$, where

$$E = \beta^{1-t} \begin{bmatrix} 11 & 0 & 0 & 0 & 0 & 3.5 \\ 0 & 11 & 0 & 0 & 0 & 1.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 11 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 3.5 & 1.5 & 0 & 0.5 & 0 & 11 \end{bmatrix}.$$

Give a bound for the relative error of the approximate solution $\tilde{x}$, if $\beta = 16$ and $t = 14$ (IBM3033).

18-8. Bauer-Fike theorem

Let

$$A = \begin{bmatrix} 10 & 10 & & & & \\ & 9 & 10 & & & \\ & & 8 & 10 & & \\ & & & \ddots & \ddots & \\ & & & & 2 & 10 \\ \varepsilon & & & & & 1 \end{bmatrix}$$

(i) Analyze the perturbation of the eigenvalues for $\varepsilon = 10^{-5}, 10^{-6}, 10^{-7}, 0$.
(ii) Compare the estimate of Bauer-Fike theorem to the matrix $A = A(0)$.

18-9. Eigenvalues

Using the MATLAB eig routine compute the eigenvalues of $B = AA^T$ for various (random) matrices $A \in \mathbb{R}^{n\times n}$ and order n. Also compute the eigenvalues of the perturbed matrices $B + R_i$, where R_i are random matrices with entries from the interval $\left[-10^{-5}, 10^{-5}\right]$ $(i = 1, \ldots, N)$. What is the maximum perturbation of the eigenvalues? How precise is the Bauer-Fike estimate? Suggested values are $N = 10$ and $5 \leq n \leq 200$. How do the results depend on the condition number and the order n? Display the maximum perturbations and the Bauer-Fike estimates graphically.

Chapter notes

The a posteriori error estimates of linear algebraic systems are not completely reliable. Demmel, Diament és Malajovich [13] showed that for the $\Theta\left(n^2\right)$ number estimators there are always cases when the estimate is unreliable (the error of the estimate exceeds a given order). The first appearance of the iterative improvement is due to Fox, Goodwin, Turing and Wilkinson (1946). The experiences show that the decrease of the residual error is not monotone.

Young [1], Hageman and Young [2] give an excellent survey of the theory and application of iterative methods. Barett, Berry et al. [3] give a software oriented survey of the subject. Frommer [4] concentrates on the parallel computations.

The convergence of the QR-method is a delicate matter. It is analyzed in great depth and much better results than Theorem 18.34 exist in the literature. There are QR-like methods that involve double shifting. Batterson [33] showed that there exists a 3×3 Hessenberg matrix with complex eigenvalues such that convergence cannot be achieved even with multiple shifting.

Several other methods are known for solving the eigenvalue problems (see, e.g. [8], [30]). The LR-method is one of the best known ones. It is very effective on positive definite Hermitian matrices. The LR-method computes the Cholesky-decomposition $A_k = LL^*$ and sets $A_{k+1} = L^*L$.

Bibliography

[1] R. Barett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozzo, C. Romine, H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994. 908

[2] S. Batterson. Convergence of the shifted QR algorithm on 3×3 normal matrices. , 58, 1990. 908

[3] J. Demmel, D. Malajovich. On the complexity of computing error bounds. *Foundations of Computational Mathematics*, 1:101–125, 2001. 907

[4] A. Frommer. *Lösung linearer Gleichungssysteme auf Parallelrechnern*. Vieweg Verlag, 1990. 908

[5] D. Watkins. Bulge exchanges in algorithms of QR type. *SIAM Journal on Matrix Analysis and Application*, 19(4):1074–1096, 1998. 908

[6] J. Wilkinson. Convergence of the LR, QR, and related algorithms. *The Computer Journal*, 8(1):77–84, 1965. 908

[7] D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, 1971. 908

Subject Index

Name index

Contents

Bibliography

[1] D.M. Young: Iterative Solution of Large Linear Systems, Academic Press, 1971 908

[2] L.A. Hageman, D.M. Young: Applied Iterative methods, Academic Press, 1981 908

[3] R. Barett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, Philadelphia, 1994 908

[4] A. Frommer: Lösung linearer Gleichungssysteme auf Parallelrechnern, Vieweg, Braunschweig, 1990 908

[5] A. Frommer, B. Pohl: Comparison results for splittings based on overlapping blocks, in: J.G. Lewis (ed.) Proceedings of the Fifth SIAM Conference on Applied Linear Algebra, SIAM, 1994, 29-33

[6] A. Galántai: A Study of Auchmuty's Error Estimate, Computers and Mathematics with Applications, 42, 2001, 1093–1102

[7] J.H. Wilkinson: Rounding Errors in Algebraic Processes, Dover, 1994

[8] J.H. Wilkinson: Convergence of the LR, QR, and related algorithms, The Computer Journal, Vol. 8, No. 1, April 1965 908

[9] F. Chaitin-Chatelin, V. Frayssé: Lectures on Finite Precision Computations, SIAM, 1996

[10] N.J. Higham: Accuracy and Stability of Numerical Algorithms, SIAM, 1996

[11] G. Golub, J.M. Ortega: Scientific Computing: An Introduction with Parallel Computing, Academic Press, 1993

[12] M.L. Overton: Numerical Computing with IEEE Floating Point Arithmetic, SIAM, 2001

[13] J. Demmel, B. Diament, G. Malajovich: On the complexity of computing error bounds, Foundations of Computational Mathematics, 1, 2001, 101-125 907

[14] Anderson, E., Bai, Z., et.al: LAPACK Users' Guide, SIAM, Philadelphia, 1992

[15] Coleman, T.F., Van Loan, C.: Handbook for Matrix Computations, SIAM, Philadelphia, 1988

[16] Forsythe, G.E., Malcolm, M.A., Moler, C.B.: Computer Methods for Mathematical Computations, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1977

[17] Forsythe, G.E., Moler, C.B.: Computer Solution of Linear Algebraic Systems, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1967

[18] Jennings, A., McKeown, J.J.: Matrix Computation, (second edition), John Wiley & Sons, 1992

[19] Móricz, F.: Numerical Methods in Algebra and Analysis, in Hungarian, Polygon, 1997

[20] Popper Gy., Csizmás F.: Numerical Methods for Engineers, in Hungarian, Akadémiai Kiadó, Typotex, 1993

[21] Ralston, A.: A First Course in Numerical Analysis, McGraw-Hill, 1965

[22] Rice, J.E.: Numerical Methods, Software, and Analysis, McGraw-Hill, 1983

[23] Rice, J.E.: Matrix Computations and Mathematical Software, McGraw-Hill, 1983

[24] Rivlin, T.J.: An Introduction to the Approximation of Functions, Dover,1981

[25] Rózsa P.: Linear algebra and its applications, in Hungarian, Műszaki Könyvkiadó, 1974

[26] Stoyan, G., Takó G.: Numerical Methods 1-3, in Hungarian, ELTE-Typotex, 1993, 1995, 1997

[27] Szamarszkij, A.A.: Introduction to Numerical Methods, in Russian, Nauka, Moscow, 1982

[28] Ueberhuber, C.W.: Numerical Computation 1-2 (Methods, Software, and Analysis), Springer, 1997

[29] Watkins, D.S.: Fundamentals of Matrix Computations, John Wiley & Sons, 1991

[30] Watkins, D.S.: Bulge exchanges in algorithms of QR type, SIAM J. Matrix Anal. Appl., Vol. 19, No, 4, pp. 1074-1096, October 1998 908

[31] Galántai A., Jeney A.: Numerical Methods, in Hungarian, Miskolci Egyetemi Kiadó, Miskolc, 2002

[32] Stoyan G. (szerk.): MATLAB Version 4 and 5, in Hungarian, Typotex, 1999

[33] Batterson, S.: Convergence of the shifted QR algorithm on 3 x 3 normal matrices, Numerische Mathematik 58, 341-352 (1990)

908