19 Numerical Analysis

The most important principles of numerical analysis will be the subject of this chapter. The solution of practical problems usually requires the application of a professional *numerical library* of numerical methods, developed for computers. Some of them will be introduced at the end of Section 19.8.3. The special computer algebra system Mathematica will be discussed with its numerical programs in Chapter 20, p. 1023 and in Section 19.8.4.2, p. 1016. Error propagation and computation errors will be examined in Section 19.8.2, p. 1004.

19.1 Numerical Solution of Non-Linear Equations in a Single Unknown

Every equation with one unknown can be transformed into one of the normal forms:

Zero form: $f(x) = 0.$ (19.1)

Fixed point form: $x = \varphi(x).$ (19.2)

Suppose equations (19.1) and (19.2) can be solved. The solutions are denoted by x^*. To get a first approximation of x^*, we can try to transfom the equation into the form $f_1(x) = f_2(x)$, where the curves of the functions $y = f_1(x)$ and $y = f_2(x)$ are more or less simple to sketch.

■ $f(x) = x^2 - \sin x = 0$. From the shapes of the curves $y = x^2$ and $y = \sin x$ it can be seen that $x_1^* = 0$ and $x_2^* \approx 0.87$ are roots (**Fig. 19.1**).

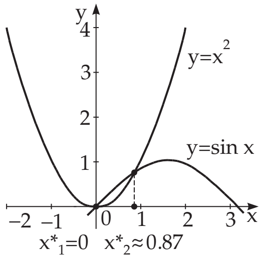


Figure 19.1

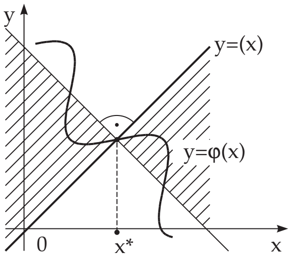


Figure 19.2

19.1.1 Iteration Method

The general idea of iterative methods is that starting with known initial approximations x_k $(k = 0, 1, \ldots, n)$ a sequence of further and better approximations is formed, step by step, hence the solution of the given equation is approached by *iteration*, by a convergent sequence. A sequence is tried to be created with convergence as fast as possible.

19.1.1.1 Ordinary Iteration Method

To solve an equation given in or transformed into the fixed point form $x = \varphi(x)$, the iteration rule

$$x_{n+1} = \varphi(x_n) \quad (n = 0, 1, 2, \ldots;\ x_0 \ \text{given}) \tag{19.3}$$

is used which is called the *ordinary iteration method*. It converges to a solution x^* if there is a neighborhood of x^* (**Fig. 19.2**) such that

$$\left|\frac{\varphi(x) - \varphi(x^*)}{x - x^*}\right| \le K < 1 \quad (K \ \text{const}) \tag{19.4}$$

holds, and the initial approximation x_0 is in this neighborhood. If $\varphi(x)$ is differentiable, then the corresponding condition is

$$|\varphi'(x)| \le K < 1. \tag{19.5}$$

The convergence of the ordinary iteration method becomes faster with smaller values of K.

■ $x^2 = \sin x$, i.e., $x_{n+1} = \sqrt{\sin x_n}$.

n	0	1	2	3	4	5
x_n	$\underline{0.87}$	$\underline{0.87}42$	$\underline{0.87}58$	$\underline{0.876}4$	$\underline{0.876}6$	$\underline{0.8767}$
$\sin x_n$	0.7643	0.7670	0.7681	0.7684	0.7686	0.7686

Remark 1: In the case of complex solutions, $x = u + \mathrm{i}v$ is substituted. Separating the real and the imaginary part, a system of two equations is obtained for the real unknowns u and v.

Remark 2: The iterative solution of non-linear equation systems can be found in 19.2.2, p. 961.

19.1.1.2 Newton's Method

1. Formula of the Newton Method

To solve an equation given in the form $f(x) = 0$, mostly the *Newton method* is used which has the formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (n = 0, 1, 2, \ldots\ ;\ x_0 \text{ is given}), \tag{19.6}$$

i.e., to get a new approximation x_{n+1}, the values of the function $f(x)$ and its first derivative $f'(x)$ at x_n are needed.

2. Convergence of the Newton Method

The condition

$$f'(x) \ne 0 \tag{19.7a}$$

is necessary for convergence of the Newton method, and the condition

$$\left|\frac{f(x)f''(x)}{f'^2(x)}\right| \le K < 1 \quad (K \text{ const}) \tag{19.7b}$$

is sufficient. The conditions (19.7a,b) must be fulfilled in a neighborhood of x^* such that it contains all the points x_n and x^* itself. If the Newton method is convergent, it converges very fast. It has quadratic convergence, which means that the error of the $(n+1)$-st approximation is less than a constant multiple of the square of the error of the n-th approximation. In the decimal system, this means that after a while the number of exact digits will double step by step.

■ The solution of the equation $f(x) = x^2 - a = 0$, i.e., the calculation of $x = \sqrt{a}$ ($a > 0$ is given), with the Newton method results in the iteration formula

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right). \tag{19.8}$$

We get for $a = 2$:

n	0	1	2	3
x_n	$\underline{1.}5$	$\underline{1.4}16\,666\,6$	$\underline{1.414\,21}5\,7$	$\underline{1.414\,213\,6}$

3. Geometric Interpretation

The geometric interpretation of the Newton method is represented in **Fig. 19.3**. The basic idea of the Newton method is the local approximation of the curve $y = f(x)$ by its tangent line.

4. Modified Newton Method

If the values of $f'(x_n)$ barely change during the iteration, it can be kept as constant for a while, and the so-called modified Newton method can be used:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_m)} \quad (m \text{ fixed},\ m < n). \tag{19.9}$$

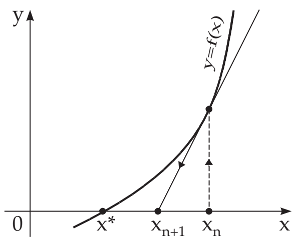


Figure 19.3

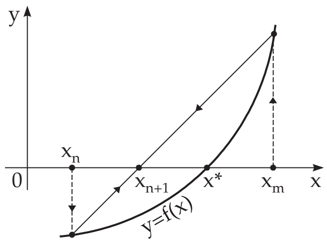


Figure 19.4

The goodness of the convergence is hardly modified by this simplification.

5. Differentiable Functions with Complex Argument

The Newton method also works for differentiable functions with complex arguments.

19.1.1.3 Regula Falsi

1. Formula for Regula Falsi

To solve the equation $f(x) = 0$, the *regula falsi* method has the rule:

$$x_{n+1} = x_n - \frac{x_n - x_m}{f(x_n) - f(x_m)} f(x_n) \quad (n = 1, 2, \dots\,; \;\; m < n\,; \, x_0, x_1 \text{ are given}). \tag{19.10}$$

Only the function values are calculated. The method follows from the Newton method (19.6) by approximating the derivative $f'(x_n)$ by the finite difference of $f(x)$ between x_n and a previous approximation x_m $(m < n)$.

2. Geometric Interpretation

The geometric interpretation of the regula falsi method is represented in **Fig. 19.4**. The basic idea of the regula falsi method is the local approximation of the curve $y = f(x)$ by a secant line.

3. Convergence

The method (19.10) is convergent if m is chosen so that $f(x_m)$ and $f(x_n)$ always have different signs. If the convergence already seems to be fast enough during the process, it will speed up if one ignores the change of sign, and substitutes $x_m = x_{n-1}$.

■ $f(x) = x^2 - \sin x = 0$.

n	$\Delta x_n = x_n - x_{n-1}$	x_n	$f(x_n)$	$\Delta y_n = f(x_n) - f(x_{n-1})$	$\frac{\Delta x_n}{\Delta y_n}$
0		$\underline{0.}9$	0.0267		
1	−0.3	$\underline{0.8}7$	−0.0074	−0.0341	0.8798
2	0.0065	$\underline{0.876}5$	−0.000252	0.007148	0.9093
3	0.000229	$\underline{0.87672}9$	0.000003	0.000255	0.8980
4	−0.000003	$\underline{0.87672}6$			

If during the process the value of $\Delta x_n/\Delta y_n$ only barely changes, one does not need to recalculate it again and again.

4. Steffensen Method

Applying the regula falsi method with $x_m = x_{n-1}$ for the equation $f(x) = x - \varphi(x) = 0$ the convergence often can be sped up, especially in the case $\varphi'(x) < -1$. This algorithm is known as the *Steffensen method*.

■ To solve the equation $x^2 = \sin x$ with the Steffensen method, the form $f(x) = x - \sqrt{\sin x} = 0$ should be used.

n	$\Delta x_n = x_n - x_{n-1}$	x_n	$f(x_n)$	$\Delta y = f(x_n) - f(x_{n-1})$	$\dfrac{\Delta x_n}{\Delta y_n}$
0		$\underline{0.}9$	0.014942		
1	−0.03	$\underline{0.8}7$	−0.004259	−0.019201	1.562419
2	0.006654	$\underline{0.876}654$	−0.000046	0.004213	1.579397
3		$\underline{0.87672}7$	0.000001		

19.1.2 Solution of Polynomial Equations

Polynomial equations of n-th degree have the form

$$f(x) = p_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0. \tag{19.11}$$

For their effective solution efficient methods are needed to calculate the function values and the derivative values of the function $p_n(x)$ and an initial estimate of the positions of the roots.

19.1.2.1 Horner's Scheme

1. Real Arguments

To determine the value of a polynomial $p_n(x)$ of n-th degree at the point $x = x_0$ from its coefficients, first the decomposition

$$p_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 = (x - x_0)p_{n-1}(x) + p_n(x_0) \tag{19.12}$$

is considered where $p_{n-1}(x)$ is a polynomial of $(n-1)$-st degree:

$$p_{n-1}(x) = a'_{n-1}x^{n-1} + a'_{n-2}x^{n-2} + \cdots + a'_1 x + a'_0. \tag{19.13}$$

The recursion formula

$$a'_{k-1} = x_0 a'_k + a_k, \quad (k = n, n-1, \ldots, 0;\ a'_n = 0,\ \ a'_{-1} = p_n(x_0)) \tag{19.14}$$

is obtained by coefficient comparison in (19.12) with respect to x^k. (Note that $a'_{n-1} = a_n$.) In this way, the coefficients a'_k of $p_{n-1}(x)$ and the value $p_n(x_0)$ are determined from the coefficients a_k of $p_n(x)$. Furthermore fewer multiplications are required than by the "traditional" way. By repeating this procedure, a decomposition of the polynomial $p_{n-1}(x)$ with the polynomial $p_{n-2}(x)$ is obtained,

$$p_{n-1}(x) = (x - x_0)p_{n-2}(x) + p_{n-1}(x_0) \tag{19.15}$$

etc., and a sequence of polynomials $p_n(x)$, $p_{n-1}(x)$ $\ldots$, $p_1(x)$, $p_0(x)$ is resulted. The calculations of the coefficients and the values of the polynomial is represented in (19.16).

$$\begin{array}{c|llllllll}
 & a_n & a_{n-1} & a_{n-2} & \ldots\ a_3 & a_2 & a_1 & a_0 & \\
x_0 & & x_0a'_{n-1} & x_0a'_{n-2} & \ldots\ x_0a'_3 & x_0a'_2 & x_0a'_1 & x_0a'_0 & \\
\hline
 & a'_{n-1} & a'_{n-2} & a'_{n-3} & \ldots\ a'_2 & a'_1 & a'_0 & \boxed{p_n(x_0)} & \\
x_0 & & x_0a''_{n-2} & x_0a''_{n-3} & \ldots\ x_0a''_2 & x_0a''_1 & x_0a''_0 & & \\
\hline
 & a''_{n-2} & a''_{n-3} & a''_{n-4} & \ldots\ a''_1 & a''_0 & \boxed{p_{n-1}(x_0)} & & \\
 & & \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots & \ldots & & \\
x_0 & & x_0a_0^{(n-1)} & & & & & & \\
\hline
 & a_1^{(n-1)} & \boxed{p_1(x_0)} & & & & & & \\
x_0 & & & & & & & & \\
\hline
 & a_0^{(n)} = p_0(x_0) & & & & & & &
\end{array} \tag{19.16}$$

From scheme (19.16) the value $p_n(x_0)$, and derivatives $p_n^{(k)}(x_0)$ are as:

$$p'_n(x_0) = 1!p_{n-1}(x_0), \ \ p''_n(x_0) = 2!p_{n-2}(x_0), \ldots,\ p_n^{(n)}(x_0) = n!p_0(x_0). \tag{19.17}$$

■ $p_4(x) = x^4 + 2x^3 - 3x^2 - 7$. The substitution value and derivatives of $p_4(x)$ are calculated at $x_0 = 2$ according to (19.16).

$$\begin{array}{c|cccccc}
 & 1 & 2 & -3 & 0 & -7 \\
2 & & 2 & 8 & 10 & 20 \\
\hline
 & 1 & 4 & 5 & 10 & \boxed{13} \\
2 & & 2 & 12 & 34 & \\
\hline
 & 1 & 6 & 17 & \boxed{44} & \\
2 & & 2 & 16 & & \\
\hline
 & 1 & 8 & \boxed{33} & & \\
2 & & 2 & & & \\
\hline
 & 1 & \boxed{10} & & & \\
2 & & & & & \\
\hline
 & 1 & & & &
\end{array}$$

We see:

$p_4(2) = 13,$
$p_4'(2) = 44,$
$p_4''(2) = 66,$
$p_4'''(2) = 60,$
$p_4^{(4)}(2) = 24.$

Remarks:

1. The polynomial $p_n(x)$ can be rearranged with respect to the powers of $x - x_0$, e.g., in the example above there is $p_4(x) = (x-2)^4 + 10(x-2)^3 + 33(x-2)^2 + 44(x-2) + 13$.

2. The Horner scheme can also be used for complex coefficients a_k. In this case for every coefficient we have to compute a real and an imaginary column according to (19.16).

2. Complex Arguments

If the coefficients a_k in (19.11) are real, then the calculation of $p_n(x_0)$ for complex values $x_0 = u_0 + \mathrm{i}v_0$ can be made real. In order to show this, $p_n(x)$ is decomposed as follows:

$$\begin{aligned} p_n(x) &= a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \\ &= (x^2 - px - q)(a'_{n-2}x^{n-2} + \cdots + a'_0) + r_1 x + r_0 \quad \text{with} \end{aligned} \qquad (19.18a)$$

$$x^2 - px - q = (x - x_0)(x - \overline{x}_0), \text{ i.e., } p = 2u_0,\ q = -({u_0}^2 + {v_0}^2). \qquad (19.18b)$$

Then,

$$p_n(x_0) = r_1 x + r_0 = (r_1 u_0 + r_0) + \mathrm{i} r_1 v_0. \qquad (19.18c)$$

To find (19.18a) the so-called *two-row Horner scheme* introduced by Collatz can be constructed:

$$\begin{array}{c|cccccccc}
 & a_n & a_{n-1} & a_{n-2} & \ldots & a_3 & a_2 & a_1 & a_0 \\
q & & & qa'_{n-2} & \ldots & qa'_3 & qa'_2 & qa'_1 & qa'_0 \\
p & & pa'_{n-2} & pa'_{n-3} & \ldots & pa'_2 & pa'_1 & pa'_0 & \\
\hline
 & a'_{n-2} & a'_{n-3} & a'_{n-4} & \ldots & a'_1 & a'_0 & \multicolumn{1}{|c}{r_1} & r_0 \\
 & = a_n & & & & & & &
\end{array} \qquad (19.18d)$$

■ $p_4(x) = x^4 + 2x^3 - 3x^2 - 7$. Calculate the value of p_4 at $x_0 = 2 - \mathrm{i}$, i.e., for $p = 4$ and $q = -5$.

$$\begin{array}{r|rrrrr}
 & 1 & 2 & -3 & 0 & -7 \\
-5 & & & -5 & -30 & -80 \\
4 & & 4 & 24 & 64 & \\
\hline
 & 1 & 6 & 16 & \multicolumn{1}{|r}{34} & -87
\end{array}$$

It results in:
$p_4(x_0) = 34x_0 - 87 = -19 - 34\mathrm{i}$.

19.1.2.2 Positions of the Roots

1. Real Roots, Sturm Sequence

The *Cartesian rule of signs* gives a first idea of whether the polynomial equation (19.11) has a real root, or not.

a) The number of positive roots is equal to the number of sign changes in the sequence of the cooefficients

$$a_n,\ a_{n-1}, \ldots,\ a_1,\ a_0 \qquad (19.19a)$$

or it is less by an even number.

b) The number of negative roots is equal to the number of sign changes in the coefficient sequence

$$a_0,\ -a_1,\ a_2, \ldots,\ (-1)^n a_n \tag{19.19b}$$

or it is less by an even number.

■ $p_5(x) = x^5 - 6x^4 + 10x^3 + 13x^2 - 15x - 16$ has 1 or 3 positive roots and 0 or 2 negative roots.

To determine the number of real roots in any given interval (a, b), *Sturm sequences* are used (see 1.6.3.2, **2.**, p. 44).

After computing the function values $y_\nu = p_n(x_\nu)$ at a uniformly distributed set of nodes $x_\nu = x_0 + \nu \cdot h$ (h constant, $\nu = 0, 1, \ldots$) (which can be easily performed by using the Horner scheme) a good guess of the graph of the function and the locations of roots are obtained. If $p_n(c)$ and $p_n(d)$ have different signs, there is at least one real root between c and d.

2. Complex Roots

In order to localize the real or complex roots into a bounded region of the complex plane the following polynomial equation is considered which is a simple consequence of (19.11):

$$f^*(x) = |a_{n-1}|r^{n-1} + |a_{n-2}|r^{n-2} + \cdots + |a_1|r + |a_0| = |a_n|r^n \tag{19.20}$$

and an upper bound r_0 is determined for the positive roots of (19.20), e.g., by systematic repeated trial and error. Then, for all roots x_k^* $(k = 1, 2, \ldots, n)$ of (19.11),

$$|x_k^*| \le r_0. \tag{19.21}$$

■ $f(x) = p_4(x) = x^4 + 4.4x^3 - 20.01x^2 - 50.12x + 29.45 = 0$, $f^*(x) = 4.4r^3 + 20.01r^2 + 50.12r + 29.45 = r^4$. Some trials are

$$\begin{aligned}
r = 6&:\quad f^*(6) = 2000.93 > 1296 = r^4,\\
r = 7&:\quad f^*(7) = 2869.98 > 2401 = r^4,\\
r = 8&:\quad f^*(8) = 3963.85 < 4096 = r^4.
\end{aligned}$$

From this it follows that $|x_k^*| < 8$ $(k = 1, 2, 3, 4)$. Actually, for the root x_1^* with maximal absolute value $-7 < x_1^* < -6$ holds.

Remark: A special method has been developed in electrotechnics in the so-called *root locus theory* for the determination of the number of complex roots with negative real parts. It is used to examine stability (see [19.11], [19.31]).

19.1.2.3 Numerical Methods

1. General Methods

The methods discussed in Section 19.1.1, p. 949, can be used to find real roots of polynomial equations. The Newton method is well suited for polynomial equations because of its fast convergence, and the fact that the values of $f(x_n)$ and $f'(x_n)$ can be easily computed by using Horner's rule. By assuming that an approximation x_n of the root x^* of a polynomial equation $f(x) = 0$ is sufficiently good, then the correction term $\delta = x^* - x_n$ can be iteratively improved by using the fixed-point equation

$$\delta = -\frac{1}{f'(x_n)}\left[f(x_n) + \frac{1}{2!}f''(x_n)\delta^2 + \cdots\right] = \varphi(\delta). \tag{19.22}$$

2. Special Methods

The *Bairstow method* is well applicable to find root pairs, especially complex conjugate pairs of roots. It starts with finding a quadratic factor of the given polynomial like the Horner scheme (19.18a–d) by determinung the coefficients p and q which make the coefficients of the linear remainder r_0 and r_1 equal to zero (see [19.30], [19.11], [19.31]).

If the computation of the root with largest or smallest absolute value is required, then the *Bernoulli method* is the choice (see [19.19]).

The *Graeffe method* has some historical importance. It gives all roots simultaneously including complex conjugate roots; however the computation costs are tremendous (see [19.11], [19.31]).

19.2 Numerical Solution of Systems of Equations

In several practical problems, there are m conditions for the n unknown quantities x_i $(i = 1, 2, \ldots, n)$ in the form of equations:

$$\begin{aligned}
F_1(x_1, x_2, \ldots, x_n) &= 0,\\
F_2(x_1, x_2, \ldots, x_n) &= 0,\\
\vdots \qquad\qquad & \vdots\\
F_m(x_1, x_2, \ldots, x_n) &= 0.
\end{aligned} \tag{19.23}$$

The unknowns x_i are to be determined so that they form a solution of the system of equations (19.23). Mostly $m = n$ holds, i.e., the number of unknowns and the number of equations are equal to each other. In the case of $m > n$, (19.23) is called an *over-determined system*; in the case of $m < n$ it is an *under-determined system*.

Over-determined systems usually have no solutions. Then one looks for the "best" solution of (19.23), in the Euclidean metric with the *least squares method*

$$\sum_{i=1}^{m} F_i^2(x_1, x_2, \ldots, x_n) = \min! \tag{19.24}$$

or in other metrics as another extreme value problem. Usually, the values of $n-m$ variables of an under-determined problem can be chosen freely, so the solution of (19.23) depends on $n - m$ parameters. It is called an $(n - m)$-dimensional *manifold of solutions*.

Linear and *non-linear systems of equations* are distinguished, depending on whether the equations are only linear or also non-linear in the unknowns.

19.2.1 Systems of Linear Equations

Consider the linear system of equations

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1,\\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2,\\
\vdots \qquad\qquad\qquad & \vdots\\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n.
\end{aligned} \tag{19.25}$$

The system (19.25) can be written in matrix form

$$\mathbf{A}\,\underline{\mathbf{x}} = \underline{\mathbf{b}} \tag{19.26a}$$

with

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n}\\ a_{21} & a_{22} & \cdots & a_{2n}\\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \qquad \underline{\mathbf{b}} = \begin{pmatrix} b_1,\\ b_2,\\ \vdots\\ b_n \end{pmatrix}, \qquad \underline{\mathbf{x}} = \begin{pmatrix} x_1,\\ x_2,\\ \vdots\\ x_n \end{pmatrix}. \tag{19.26b}$$

Suppose the quadratic matrix $\mathbf{A} = (a_{ik})$ $(i, k = 1, 2, \ldots, n)$ is regular, so system (19.25) has a unique solution (see 4.5.2.1, **2.**, p. 309). In the practical solution of (19.25) two types of solution methods are distinguished:

1. Direct Methods are based on elementary transformations, from which the solution can be obtained immediately. These are the pivoting techniques (see 4.5.1.2, p. 307) and the methods given in 19.2.1.1–19.2.1.3.

2. Iteration methods start with a known initial approximation of the solution, and form a sequence of approximations that converges to the solution of (19.25) (see 19.2.1.4, p. 960).

19.2.1.1 Triangular Decomposition of a Matrix

1. Principle of the Gauss Elimination Method

By elementary transformations

1. interchanging rows,
2. multiplying a row by a non-zero number and
3. adding a multiple of a row to another row,
the system $\mathbf{A}\,\underline{\mathbf{x}} = \underline{\mathbf{b}}$ is transformed into the so-called *row echelon form*

$$\mathbf{R}\,\underline{\mathbf{x}} = \underline{\mathbf{c}} \text{ with } \mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1n} \\ & r_{22} & r_{23} & \dots & r_{2n} \\ & & r_{33} & \dots & r_{3n} \\ & 0 & & \ddots & \vdots \\ & & & & r_{nn} \end{pmatrix}. \tag{19.27}$$

Since only equivalent transformations were made, the system of equations $\mathbf{R}\,\underline{\mathbf{x}} = \underline{\mathbf{c}}$ has the same solutions as $\mathbf{A}\,\underline{\mathbf{x}} = \underline{\mathbf{b}}$. From (19.27) it follows:

$$x_n = \frac{c_n}{r_{nn}}, \quad x_i = \frac{1}{r_{ii}}\left(c_i - \sum_{k=i+1}^{n} r_{ik}x_k\right) \quad (i = n-1, n-2, \dots, 1)\,. \tag{19.28}$$

The rule given in (19.28) is called *backward substitution*, since the equations of (19.27) are used in the opposite order as they follow each other.
The transition from $\mathbf{A}$ to $\mathbf{R}$ is made by $n-1$ so-called *elimination steps*, whose procedure is shown by the first step. This step transforms matrix $\mathbf{A}$ into matrix $\mathbf{A}_1$:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \qquad \mathbf{A}_1 = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} \dots a_{1n}^{(1)} \\ 0 & \boxed{\begin{matrix} a_{22}^{(1)} \dots a_{2n}^{(1)} \\ a_{32}^{(1)} \dots a_{3n}^{(1)} \\ \vdots \\ a_{n2}^{(1)} \dots a_{nn}^{(1)} \end{matrix}} \\ 0 & \\ \vdots & \\ 0 & \end{pmatrix}. \tag{19.29}$$

Then:

1. An $a_{r1} \neq 0$ is chosen (according to (19.33)). If there is none, stop: $\mathbf{A}$ is singular. Otherwise a_{r1} is called the *pivot*.

2. The first and the r-th row of $\mathbf{A}$ are interchanged. The result is $\overline{\mathbf{A}}$.

3. The l_{i1} $(i = 2, 3, \dots, n)$ multiple of the first row is subtracted from the i-th row of the matrix $\overline{\mathbf{A}}$.
The result is the matrix $\mathbf{A}_1$ and analogously the new right-hand side $\underline{\mathbf{b}}_1$ with the elements

$$a_{ik}^{(1)} = \overline{a}_{ik} - l_{i1}\overline{a}_{1k} \text{ with } l_{i1} = \frac{\overline{a}_{i1}}{\overline{a}_{11}},$$
$$b_i^{(1)} = \overline{b}_i - l_{i1}\overline{b}_1 \quad (i, k = 2, 3, \dots, n). \tag{19.30}$$

The framed submatrix in $\mathbf{A}_1$ (see (19.29)) is of type $(n-1, n-1)$ and it will be handled analogously to $\mathbf{A}$, etc. This method is called the *Gaussian elimination method* or the *Gauss algorithm* (see 4.5.2.4, p. 312).

2. Triangular Decomposition

The result of the Gauss elimination method can be formulated as follows: To every regular matrix $\mathbf{A}$ there exists a so-called *triangular decomposition* or *LU factorization* of the form

$$\mathbf{P}\,\mathbf{A} = \mathbf{L}\,\mathbf{R} \tag{19.31}$$

with

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1n} \\ & r_{22} & r_{23} & \dots & r_{2n} \\ & & r_{33} & \dots & r_{3n} \\ & 0 & & \ddots & \vdots \\ & & & & r_{nn} \end{pmatrix}, \quad \mathbf{L} = \begin{pmatrix} 1 & & & & \\ l_{21} & 1 & & 0 & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & & \ddots & \\ l_{n1} & l_{n2} & \dots & l_{n,n-1} & 1 \end{pmatrix}. \tag{19.32}$$

Here $\mathbf{R}$ is called an *upper triangular matrix*, $\mathbf{L}$ is a *lower triangular matrix* and $\mathbf{P}$ is a so-called *permutation matrix*. A permutation matrix is a quadratic matrix which has exactly one 1 in every row and every column, and the other elements are zeros. The multiplication $\mathbf{P}\,\mathbf{A}$ results in row interchanges in $\mathbf{A}$, which comes from the choices of the pivot elements during the elimination procedure.

■ The Gauss elimination method should be used for the system $\begin{pmatrix} 3 & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 7 \\ 4 \end{pmatrix}$. In schematic form, where the coefficient matrix and the vector from the right-hand side are written next to each other (into the so-called *extended coefficient matrix*), the calculations are:

$$(\mathbf{A}, \underline{\mathbf{b}}) = \left(\boxed{\begin{array}{ccc} \boxed{3} & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{array}} \;\middle|\; \begin{array}{c} 2 \\ 7 \\ 4 \end{array} \right) \Rightarrow \left(\begin{array}{c|c} \begin{array}{ccc} 3 & 1 & 6 \end{array} & 2 \\ \begin{array}{c} 2/3 \\ 1/3 \end{array} \boxed{\begin{array}{cc} 1/3 & -1 \\ \boxed{2/3} & -1 \end{array}} & \begin{array}{c} 17/3 \\ 10/3 \end{array} \end{array} \right) \Rightarrow \left(\begin{array}{ccc|c} 3 & 1 & 6 & 2 \\ 1/3 & 2/3 & -1 & 10/3 \\ 2/3 & 1/2 & \boxed{-1/2} & 4 \end{array} \right), \text{ i.e.,}$$

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \Rightarrow \mathbf{P}\,\mathbf{A} = \begin{pmatrix} 3 & 1 & 6 \\ 1 & 1 & 1 \\ 2 & 1 & 3 \end{pmatrix}, \mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 1/2 & 1 \end{pmatrix}, \mathbf{R} = \begin{pmatrix} 3 & 1 & 6 \\ 0 & 2/3 & -1 \\ 0 & 0 & -1/2 \end{pmatrix}.$$

In the extended coefficient matrices, the matrices $\mathbf{A}$, $\mathbf{A}_1$ and $\mathbf{A}_2$, and also the pivots are shown in boxes. Solution: $x_3 = -8$, $x_2 = -7$, $x_1 = 19$.

3. Application of Triangular Decomposition

With the help of triangular decomposition, the solution of the linear system of equations $\mathbf{A}\,\underline{\mathbf{x}} = \underline{\mathbf{b}}$ can be described in three steps:

1. $\mathbf{P}\,\mathbf{A} = \mathbf{L}\,\mathbf{R}$: Determination of the triangular decomposition and substitution $\mathbf{R}\,\underline{\mathbf{x}} = \underline{\mathbf{c}}$.

2. $\mathbf{L}\,\underline{\mathbf{c}} = \mathbf{P}\,\underline{\mathbf{b}}$: Determination of the auxiliary vector $\underline{\mathbf{c}}$ by forward substitution.

3. $\mathbf{R}\,\underline{\mathbf{x}} = \underline{\mathbf{c}}$: Determination of the solution $\underline{\mathbf{x}}$ by backward substitution.

If the solution of a system of linear equations is handled by the expanded coefficient matrix $(\mathbf{A}, \underline{\mathbf{b}})$, as in the above example, by the Gauss elimination method, then the lower triangular matrix $\mathbf{L}$ is not needed explicitly. This can be especially useful if several systems of linear equations are to be solved after each other with the same coefficient matrix, with different right-hand sides.

4. Choice of the Pivot Elements

Theoretically, every non-zero element $a_{i1}^{(k-1)}$ of the first column of the matrix $\mathbf{A}_{k-1}$ could be used as a pivot element at the k-th elimination step. In order to improve the accuracy of solution (to decrease the accumulated rounding errors of the operations), the following strategies are recommended.

1. Diagonal Strategy The diagonal elements are chosen successively as pivot elements if possible, i.e., there is no row interchange. This kind of choice of the pivot element makes sense if the absolute value of the elements of the main diagonal are fairly large compared to the others in the same row.

2. Column Pivoting To perform the k-th elimination step, such row index r is chosen for which:

$$|a_{rk}^{(k-1)}| = \max_{i \geq k} |a_{ik}^{(k-1)}|. \tag{19.33}$$

If $r \neq k$, then the r-th and the k-th rows will be interchanged. It can be proven that this strategy makes the accumulated rounding errors smaller.

19.2.1.2 Cholesky's Method for a Symmetric Coefficient Matrix

In several cases, the coefficient matrix $\mathbf{A}$ in (19.26a) is not only symmetric, but also *positive definite*, i.e., for the corresponding *quadratic form* $Q(\underline{\mathbf{x}})$ holds

$$Q(\underline{\mathbf{x}}) = \underline{\mathbf{x}}^{\mathrm{T}} \mathbf{A}\, \underline{\mathbf{x}} = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} x_i x_k > 0 \tag{19.34}$$

for every $\underline{\mathbf{x}} \in \mathbb{R}^n$, $\underline{\mathbf{x}} \neq \underline{\mathbf{0}}$. Since for every symmetric positive definite matrix $\mathbf{A}$ there exists a unique triangular decomposition

$$\mathbf{A} = \mathbf{L}\,\mathbf{L}^{\mathrm{T}} \tag{19.35}$$

with

$$\mathbf{L} = \begin{pmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & 0 & \\ l_{31} & l_{32} & l_{33} & & \\ \vdots & & & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{pmatrix}, \tag{19.36a}$$

$$l_{kk} = \sqrt{a_{kk}^{(k-1)}}, \quad l_{ik} = \frac{a_{ik}^{(k-1)}}{l_{kk}} \quad (i = k, k+1, \dots, n)\,; \tag{19.36b}$$

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - l_{ik} l_{jk} \quad (i, j = k+1, k+2, \dots, n), \tag{19.36c}$$

the solution of the corresponding linear system of equations $\mathbf{A}\,\underline{\mathbf{x}} = \underline{\mathbf{b}}$ can be determined by the *Cholesky method* by the following steps:

1. $\mathbf{A} = \mathbf{L}\,\mathbf{L}^{\mathrm{T}}$: Determination of the so-called *Cholesky decomposition* and substitution $\mathbf{L}^{\mathrm{T}}\underline{\mathbf{x}} = \underline{\mathbf{c}}$.
2. $\mathbf{L}\,\underline{\mathbf{c}} = \underline{\mathbf{b}}$: Determination of the auxiliary vector $\underline{\mathbf{c}}$ by forward substitution.
3. $\mathbf{L}^{\mathrm{T}}\underline{\mathbf{x}} = \underline{\mathbf{c}}$: Determination of the solution $\underline{\mathbf{x}}$ by backward substitution.
For large values of n the computation cost of the Cholesky method is approximately half of that of the LU decomposition given in (19.31), p. 956.

19.2.1.3 Orthogonalization Method

1. Linear Fitting Problem

Suppose an *over-determined linear system of equations*

$$\sum_{k=1}^{n} a_{ik} x_k = b_i \quad (i = 1, 2, \dots, m;\ m > n), \tag{19.37}$$

is given in matrix form

$$\mathbf{A}\underline{\mathbf{x}} = \underline{\mathbf{b}}. \tag{19.38}$$

Suppose the coefficient matrix $\mathbf{A} = (a_{ik})$ with size $(m \times n)$ has full rank n, i.e., its columns are linearly independent. Since an over-determined linear system of equations usually has no solution, instead of (19.37) the so-called *error equations* are considered

$$r_i = \sum_{k=1}^{n} a_{ik} x_k - b_i \quad (i = 1, 2, \dots, m;\ m > n) \tag{19.39}$$

with *residues* r_i, and the sum of their squares should be minimalized:

$$\sum_{i=1}^{m} {r_i}^2 = \sum_{i=1}^{m} \left[\sum_{k=1}^{n} a_{ik} x_k - b_i \right]^2 = F(x_1, x_2, \dots, x_n) = \min! \tag{19.40}$$

The problem (19.40) is called a *linear fitting problem* or a *linear least squares problem* (see also 6.2.5.5, p. 456). The necessary condition for the relative minimum of the *sum of residual squares* $F(x_1, x_2, \ldots, x_n)$ is

$$\frac{\partial F}{\partial x_k} = 0 \quad (k = 1, 2, \ldots, n) \tag{19.41}$$

and it leads to the linear system of equations

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\underline{\mathbf{x}} = \mathbf{A}^{\mathrm{T}}\underline{\mathbf{b}}. \tag{19.42}$$

The transition from (19.38) to (19.42) is called a *Gauss transformation*, since the system (19.42) arises by applying the *Gaussian least squares method* (see 6.2.5.5, p. 456) for (19.38). Since $\mathbf{A}$ is supposed to be of full rank, $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is a positive definite matrix of size $(n \times n)$, and the so-called *normal equations* (19.42) can be solved numerically by the Cholesky method (see 19.2.1.2, p. 958).
One can have numerical difficulties with the solution of the normal equations (19.42) if the *condition number* (see [19.24]) of the matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is too large. The solution $\underline{\mathbf{x}}$ can then have a large relative error. Because of this problem, it is better to use the orthogonalization method for solving numerically linear fitting problems.

2. Orthogonalization Method

The following facts are the basis of the following orthogonalization method for solving a linear least squares problem (19.40):

1. The length of a vector does not change during an orthogonal transformation, i.e., the vectors $\underline{\mathbf{x}}$ and $\underline{\tilde{\mathbf{x}}} = \mathbf{Q}_0\underline{\mathbf{x}}$ with

$$\mathbf{Q}_0^{\mathrm{T}}\mathbf{Q}_0 = \mathbf{E} \tag{19.43}$$

have the same length.

2. For every matrix $\mathbf{A}$ of size (m, n) with maximal rank n $(n < m)$ there exists an orthogonal matrix $\mathbf{Q}$ of size (m, m) such that

$$\mathbf{A} = \mathbf{Q}\hat{\mathbf{R}} \quad (19.44) \qquad \text{with} \quad \mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{E} \quad \text{and} \quad \hat{\mathbf{R}} = \begin{pmatrix} \mathbf{R} \\ \mathbf{O} \end{pmatrix} = \left(\begin{array}{cccc} r_{11} & r_{12} & \ldots & r_{1n} \\ & r_{22} & \ldots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \\ \hline & & \mathbf{O} & \end{array}\right). \tag{19.45}$$

Here $\mathbf{R}$ is an upper triangular matrix of size (n, n), and $\mathbf{O}$ is a zero matrix of size $(m - n, n)$.
The factored form (19.44) of matrix $\mathbf{A}$ is called the *QR decomposition*. So, the error equations (19.39) can be transformed into the equivalent system

$$\begin{array}{rcl} r_{11}x_1 + r_{12}x_2 + \ldots + r_{1n}x_n - \hat{b}_1 & = & \hat{r}_1, \\ r_{22}x_2 + \ldots + r_{2n}x_n - \hat{b}_2 & = & \hat{r}_2, \\ \ddots \quad \vdots \quad \vdots & = & \vdots \\ r_{nn}x_n - \hat{b}_n & = & \hat{r}_n, \\ - \hat{b}_{n+1} & = & \hat{r}_{n+1}, \\ \vdots & & \vdots \\ - \hat{b}_m & = & \hat{r}_m \end{array} \tag{19.46}$$

without changing the sum of the squares of the residuals. From (19.46) it follows that the sum of the squares is minimal for $\hat{r}_1 = \hat{r}_2 = \cdots = \hat{r}_n = 0$ and the minimum value is equal to the sum of the squares of $\hat{r}_{n+1}$ to $\hat{r}_m$. The required solution $\underline{\mathbf{x}}$ can be got by backward substitution

$$\mathbf{R}\underline{\mathbf{x}} = \hat{\underline{\mathbf{b}}}_0, \tag{19.47}$$

where $\underline{\hat{\mathbf{b}}}_0$ is the vector with components $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n$ obtained from (19.46).
There are two methods most often used for a stepwise transition of (19.39) into (19.46):

1. Givens transformation,

2. Householder transformation.

The first one results in the QR decomposition of matrix $\mathbf{A}$ by *rotations*, the other one by *reflections*. The numerical implementations can be found in [19.23].

Practical problems in linear least squares approximations are solved mostly by the Householder transformation, where the frequently occurring special *band structure* of the coefficient matrix $\mathbf{A}$ can be used.

19.2.1.4 Iteration Methods

1. Jacobi Method

Suppose in the coefficient matrix of the linear system of equations (19.25) every diagonal element a_{ii} $(i = 1, 2, \ldots, n)$ is different from zero. Then the i-th row can be solved for the unknown x_i, and it immediately results the following iteration rule, where μ is the iteration index:

$$x_i^{(\mu+1)} = \frac{b_i}{a_{ii}} - \sum_{\substack{k=1 \\ (k \neq i)}}^{n} \frac{a_{ik}}{a_{ii}} x_k^{(\mu)} \quad (i = 1, 2, \ldots, n) \tag{19.48}$$

$$(\mu = 0, 1, 2, \ldots;\ x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)} \text{ are given initial values}).$$

Formula (19.48) is called the *Jacobi method.* Every component of the new vector $\underline{\mathbf{x}}^{(\mu+1)}$ is calculated from the components of $\underline{\mathbf{x}}^{(\mu)}$. If at least one of the conditions

$$\max_k \sum_{\substack{i=1 \\ (i \neq k)}}^{n} \left| \frac{a_{ik}}{a_{ii}} \right| < 1 \qquad \textit{column sum criterion} \tag{19.49}$$

or

$$\max_i \sum_{\substack{k=1 \\ (k \neq i)}}^{n} \left| \frac{a_{ik}}{a_{ii}} \right| < 1 \qquad \textit{row sum criterion} \tag{19.50}$$

holds, then the Jacobi method is convergent for any initial vector $\underline{\mathbf{x}}^{(0)}$.

2. Gauss-Seidel Method

If the first component $x_1^{(\mu+1)}$ is calculated by the Jacobi method, then this value can be used in the calculation of $x_2^{(\mu+1)}$. While proceeding similarly in the calculation of the further components, the following iteration formula is obtained:

$$x_i^{(\mu+1)} = \frac{b_i}{a_{ii}} - \sum_{k=1}^{i-1} \frac{a_{ik}}{a_{ii}} x_k^{(\mu+1)} - \sum_{k=i+1}^{n} \frac{a_{ik}}{a_{ii}} x_k^{(\mu)} \tag{19.51}$$

$$(i = 1, 2, \ldots, n;\ x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)} \text{ given initial value; } \mu = 0, 1, 2, \ldots).$$

Formula (19.51) is called the *Gauss-Seidel method.* The Gauss-Seidel method usually converges faster than the Jacobi method, but its convergence criterion is more complicated.

■
$$\begin{aligned}
10x_1 - 3x_2 - 4x_3 + 2x_4 &= 14,\\
-3x_1 + 26x_2 + 5x_3 - x_4 &= 22,\\
-4x_1 + 5x_2 + 16x_3 + 5x_4 &= 17,\\
2x_1 + 3x_2 - 4x_3 - 12x_4 &= -20.
\end{aligned}$$

The corresponding iteration formula according to (19.51) is:

$$\begin{aligned}
x_1^{(\mu+1)} &= \frac{1}{10}\left(14 + 3x_2^{(\mu)} + 4x_3^{(\mu)} - 2x_4^{(\mu)}\right),\\
x_2^{(\mu+1)} &= \frac{1}{26}\left(22 + 3x_1^{(\mu+1)} - 5x_3^{(\mu)} + x_4^{(\mu)}\right),\\
x_3^{(\mu+1)} &= \frac{1}{16}\left(17 + 4x_1^{(\mu+1)} - 5x_2^{(\mu+1)} - 5x_4^{(\mu)}\right),\\
x_4^{(\mu+1)} &= \frac{1}{12}\left(-20 + 2x_1^{(\mu+1)} + 3x_2^{(\mu+1)} - 4x_3^{(\mu+1)}\right).
\end{aligned}$$

Some approximations and the solution are given here:

$\underline{\mathbf{x}}^{(0)}$	$\underline{\mathbf{x}}^{(1)}$	$\underline{\mathbf{x}}^{(4)}$	$\underline{\mathbf{x}}^{(5)}$	$\underline{\mathbf{x}}$
0	1.4	1.5053	1.5012	1.5
0	1.0077	0.9946	0.9989	1
0	1.0976	0.5059	0.5014	0.5
0	1.7861	1.9976	1.9995	2

3. Relaxation Method

The iteration formula of the Gauss-Seidel method (19.51) can be written in the so-called *correction form*

$$x_i^{(\mu+1)} = x_i^{(\mu)} + \left(\frac{b_i}{a_{ii}} - \sum_{k=1}^{i-1}\frac{a_{ik}}{a_{ii}}x_k^{(\mu+1)} - \sum_{k=i}^{n}\frac{a_{ik}}{a_{ii}}x_k^{(\mu)}\right), \quad \text{i.e.,}$$

$$x_i^{(\mu+1)} = x_i^{(\mu)} + d_i^{(\mu)} \quad (i = 1, 2, \ldots, n;\ \mu = 0, 1, 2, \ldots). \tag{19.52}$$

By an appropriate choice of a *relaxation parameter* ω and rewriting (19.52) in the form

$$x_i^{(\mu+1)} = x_i^{(\mu)} + \omega d_i^{(\mu)} \quad (i = 1, 2, \ldots, n;\ \mu = 0, 1, 2, \ldots), \tag{19.53}$$

one can try to improve the speed of convergence. It can be shown that convergence is possible only for

$$0 < \omega < 2. \tag{19.54}$$

For $\omega = 1$ we retrieve the Gauss-Seidel method. In the case of $\omega > 1$, which is called over-relaxation, the corresponding iteration method is called the *SOR method* (**s**uccessive **o**ver**r**elaxation). The determination of an optimal relaxation parameter is possible only for some special types of matrices.
Iterative methods are applied to solve linear systems of equations in the first place when the main diagonal elements a_{ii} of the coefficient matrix have an absolute value much larger than the other elements a_{ik} $(i \neq k)$ (in the same row or column), or when the rows of the system of equations can be rearranged in a certain way to get such a form.

19.2.2 System of Non-Linear Equations

Suppose the system of n non-linear equations

$$F_i(x_1, x_2, \ldots, x_n) = 0 \quad (i = 1, 2, \ldots, n) \tag{19.55}$$

for the n unknowns x_1, x_2, …, x_n has a solution. Usually, a numerical solution can be given only by an iteration method.

19.2.2.1 Ordinary Iteration Method

The ordinary iteration method can be used if the equations (19.55) can be transformed into a fixed-point form

$$x_i = f_i(x_1, x_2, \ldots, x_n) \quad (i = 1, 2, \ldots, n). \tag{19.56}$$

Then, starting from estimated approximations $x_1^{(0)}$, $x_2^{(0)}$,…, $x_n^{(0)}$, the improved values are obtained either by

1. iteration with simultaneous steps

$$x_i^{(\mu+1)} = f_i\left(x_1^{(\mu)}, x_2^{(\mu)}, \ldots, x_n^{(\mu)}\right) \quad (i = 1, 2, \ldots, n;\ \mu = 0, 1, 2, \ldots) \tag{19.57}$$

or by

2. iteration with sequential steps

$$x_i^{(\mu+1)} = f_i\left(x_1^{(\mu+1)}, \ldots, x_{i-1}^{(\mu+1)}, x_i^{(\mu)}, x_{i+1}^{(\mu)}, \ldots, x_n^{(\mu)}\right) \quad (i = 1, 2, \ldots, n;\ \mu = 0, 1, 2, \ldots). \tag{19.58}$$

It is of crucial importance for the convergence of this method that in the neighborhood of the solution the functions f_i should depend only weakly on the unknowns, i.e., if f_i are differentiable, the absolute values of the partial derivatives must be rather small. We get as a *convergence condition*

$$K < 1 \text{ with } K = \max_i \left(\sum_{k=1}^{n} \max \left| \frac{\partial f_i}{\partial x_k} \right| \right). \tag{19.59}$$

With this quantity K, the *error estimation* is the following:

$$\max_i \left| x_i^{(\mu+1)} - x_i \right| \le \frac{K}{1-K} \max_i \left| x_i^{(\mu+1)} - x_i^{(\mu)} \right|. \tag{19.60}$$

Here, x_i is the component of the required solution, $x_i^{(\mu)}$ and $x_i^{(\mu+1)}$ are the corresponding μ-th and $(\mu+1)$-th approximations.

19.2.2.2 Newton's Method

The Newton method is used for the problem given in the form (19.55). After finding the initial approximation values $x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}$, the functions F_i are expanded in Taylor form as functions of n independent variables $x_1, x_2, \dots, x_n$ (see p. 471). Terminating the expansion after the linear terms, from (19.55) a linear system of equations is obtained, and iterative improvements can be got by the following formula:

$$F_i\left(x_1^{(\mu)}, x_2^{(\mu)}, \dots, x_n^{(\mu)}\right) + \sum_{k=1}^{n} \frac{\partial F_i}{\partial x_k}\left(x_1^{(\mu)}, \dots, x_n^{(\mu)}\right)\left(x_k^{(\mu+1)} - x_k^{(\mu)}\right) = 0 \tag{19.61}$$

$$(i = 1, 2, \dots, n;\ \mu = 0, 1, 2, \dots).$$

The coefficient matrix of the linear system of equations (19.61), which should be solved in every iteration step, is

$$\mathbf{F}'(\underline{\mathbf{x}}^{(\mu)}) = \left(\frac{\partial F_i}{\partial x_k}\left(x_1^{(\mu)}, x_2^{(\mu)}, \dots, x_n^{(\mu)}\right) \right) \quad (i, k = 1, 2, \dots, n) \tag{19.62}$$

and it is called the *Jacobian matrix*. If the Jacobian matrix is invertible in the neighborhood of the solution, the Newton method is locally quadratically convergent, i.e., its convergence essentially depends on how good the initial approximations are. If $x_k^{(\mu+1)} - x_k^{(\mu)} = d_k^{(\mu)}$ are substituted in (19.61), then the Newton method can be written in the correction form

$$x_k^{(\mu+1)} = x_k^{(\mu)} + d_k^{(\mu)} \quad (i = 1, 2, \dots, n;\ \mu = 0, 1, 2, \dots). \tag{19.63}$$

To reduce the sensitivity to the initial values, analogously to the relaxation method, a so-called *damping* or *step length parameter* γ can be introduced (*damping method*):

$$x_k^{(\mu+1)} = x_k^{(\mu)} + \gamma d_k^{(\mu)} \quad (i = 1, 2, \dots, n;\ \mu = 0, 1, 2, \dots;\ \gamma > 0). \tag{19.64}$$

Methods to determine γ can be found in [19.24].

19.2.2.3 Derivative-Free Gauss-Newton Method

To solve the least squares problem (19.24), one proceeds iteratively in the non-linear case as follows:

1. Starting from a suitable initial approximation $x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}$, the non-linear functions $F_i(x_1, x_2, \dots, x_n)$ $(i = 1, 2, \dots, m; m > n)$ are approximated as in the Newton method (see (19.61)) by linear approximations $\tilde{F}_i(x_1, x_2, \dots, x_n)$, which are calculated in every iteration step according to

$$\tilde{F}_i(x_1, \dots, x_n) = F_i\left(x_1^{(\mu)}, x_2^{(\mu)}, \dots, x_n^{(\mu)}\right) + \sum_{k=1}^{n} \frac{\partial F_i}{\partial x_k}\left(x_1^{(\mu)}, \dots, x_n^{(\mu)}\right)\left(x_k - x_k^{(\mu)}\right)$$

$$(i = 1, 2, \dots, m;\ \mu = 0, 1, 2, \dots). \tag{19.65}$$

2. $d_k^{(\mu)} = x_k - x_k^{(\mu)}$ are substituted in (19.65) and the corrections $d_k^{(\mu)}$ are determined by using the Gaussian least squares method, i.e., by the solution of the linear least squares problem

$$\sum_{i=1}^{m} \tilde{F}_i^2(x_1, \ldots, x_n) = \min, \tag{19.66}$$

e.g., with the help of the normal equations (see (19.42)), or the Householder method (see 19.6.2.2, p. 985).

3. The approximations for the required solution are given by the formulas

$$x_k^{(\mu+1)} = x_k^{(\mu)} + d_k^{(\mu)} \quad \text{or} \tag{19.67a}$$

$$x_k^{(\mu+1)} = x_k^{(\mu)} + \gamma d_k^{(\mu)} \quad (k = 1, 2, \ldots, n), \tag{19.67b}$$

where γ $(\gamma > 0)$ is a step length parameter similar to the Newton method.

By repeating steps 2 and 3 with $x_k^{(\mu+1)}$ instead of $x_k^{(\mu)}$ one gets the *Gauss-Newton method*. It results in a sequence of approximation values, whose convergence strongly depends on the accuracy of the initial approximation. The sum of the error squares can be reduced by introducing a length parameter γ.

If the evaluation of the partial derivatives $\dfrac{\partial F_i}{\partial x_k}\left(x_1^{(\mu)}, \ldots, x_n^{(\mu)}\right)$ $(i = 1, 2, \ldots, m;\ k = 1, 2, \ldots, n)$ requires too much work, the partial derivatives can be approximated by difference quotients :

$$\frac{\partial F_i}{\partial x_k}\left(x_1^{(\mu)}, \ldots, x_k^{(\mu)}, \ldots, x_n^{(\mu)}\right) \approx \frac{1}{h_k^{(\mu)}}\Big[F_i\left(x_1^{(\mu)}, \ldots, x_{k-1}^{(\mu)}, x_k^{(\mu)} + h_k^{(\mu)}, x_{k+1}^{(\mu)}, \ldots, x_n^{(\mu)}\right)$$

$$-F_i\left(x_1^{(\mu)}, \ldots, x_k^{(\mu)}, \ldots, x_n^{(\mu)}\right)\Big] \quad (i = 1, 2, \ldots, m;\ k = 1, 2, \ldots, n;\ \mu = 0, 1, 2, \ldots). \tag{19.68}$$

The so-called *discretization step sizes* $h_k^{(\mu)}$ may depend on the iteration steps and the values of the variables.

If the approximations (19.68) are used, then only function values F_i are to be calculated while performing the Gauss-Newton method, i.e., the method is *derivative free*.

19.3 Numerical Integration

19.3.1 General Quadrature Formulas

The numerical evaluation of the definite integral

$$I(f) = \int_a^b f(x)\,dx \tag{19.69}$$

must be done only approximately if the integrand $f(x)$ cannot be integrated by elementary calculus, or it is too complicated, or when the function is known only at certain points x_ν, at the so-called *interpolation nodes* from the integration interval $[a, b]$. The so-called *quadrature formulas* are used for the approximate calculation of (19.69). They have the general form

$$Q(f) = \sum_{\nu=0}^{n} c_{0\nu} y_\nu + \sum_{\nu=0}^{n} c_{1\nu} y_\nu^{(1)} + \cdots + \sum_{\nu=0}^{n} c_{p\nu} y_\nu^{(p)} \tag{19.70}$$

with $y_\nu^{(\mu)} = f^{(\mu)}(x_\nu)$ $(\mu = 1, 2, \ldots, p;\ \nu = 1, 2, \ldots, n)$, $y_\nu = f(x_\nu)$, and constant values of $c_{\mu\nu}$. Obviously,

$$I(f) = Q(f) + R, \tag{19.71}$$

where R is the error of the quadrature formula. In the application of quadrature formulas it is supposed that the required values of the integrand $f(x)$ and its derivatives at the interpolation nodes are known as numerical values. Formulas using only the values of the function are called *mean value formulas*; formulas using also the derivatives are called *Hermite quadrature formulas*.

19.3.2 Interpolation Quadratures

The following formulas represent so-called *interpolation quadratures*. Here, the integrand $f(x)$ is interpolated at certain interpolation nodes (possibly the least number of them) by a polynomial $p(x)$ of corresponding degree, and the integral of $f(x)$ is replaced by that of $p(x)$. The formula for the integral over the entire interval is given by summation. Here the formulas are given for the most practical cases. The interpolation nodes are *equidistant*:

$$x_\nu = x_0 + \nu h \quad (\nu = 0, 1, 2, \dots, n), \quad x_0 = a, \;\; x_n = b, \;\; h = \frac{b-a}{n}. \tag{19.72}$$

An upper bound for the magnitude of the error $|R|$ is given for every quadrature formula. Here, M_μ means an upper bound of $|f^{(\mu)}(x)|$ on the entire domain.

19.3.2.1 Rectangular Formula

In the interval $[x_0, x_0+h]$, $f(x)$ is replaced by the constant function $y = y_0 = f(x_0)$, which interpolates $f(x)$ at the interpolation node x_0, which is the left endpoint of the integration interval. In this way the *simple rectangular formula* is obtained:

$$\int\limits_{x_0}^{x_0+h} f(x)\,dx \approx h \cdot y_0, \quad |R| \le \frac{h^2}{2} M_1. \tag{19.73a}$$

The *left-sided rectangular formula* by summation is

$$\int\limits_a^b f(x)\,dx \approx h(y_0 + y_1 + y_2 + \cdots + y_{n-1}), \quad |R| \le \frac{(b-a)h}{2} M_1. \tag{19.73b}$$

M_1 denotes an upper bound of $|f'(x)|$ on the entire domain of integration.
One gets analogously the *right-sided rectangular sum*, if one replaces y_0 by y_1 in (19.73a). The formula is

$$\int\limits_a^b f(x)\,dx \approx h(y_1 + y_2 + \cdots + y_n), \quad |R| \le \frac{(b-a)h}{2} M_1. \tag{19.74}$$

19.3.2.2 Trapezoidal Formula

$f(x)$ is replaced by a polynomial of first degree in the interval $[x_0, x_0+h]$, which interpolates $f(x)$ at the interpolation nodes x_0 and $x_1 = x_0 + h$. The approximation is

$$\int\limits_{x_0}^{x_0+h} f(x)\,dx \approx \frac{h}{2}(y_0 + y_1), \quad |R| \le \frac{h^3}{12} M_2. \tag{19.75}$$

The so-called *trapezoidal formula* can be obtained by summation:

$$\int\limits_a^b f(x)\,dx \approx h\left(\frac{y_0}{2} + y_1 + y_2 + \cdots + y_{n-1} + \frac{y_n}{2}\right), \quad |R| \le \frac{(b-a)h^2}{12} M_2. \tag{19.76}$$

M_2 denotes an upper bound of $|f''(x)|$ on the entire integration domain. The error of the trapezoidal formula is proportional to h^2, i.e., the trapezoidal sum has an error of order 2. It follows that it converges to the definite integral for $h \to 0$ (hence, $n \to \infty$), if rounding errors are not considered.

19.3.2.3 Simpson's Formula

$f(x)$ is replaced by a polynomial of second degree in the interval $[x_0, x_0 + 2h]$, which interpolates $f(x)$ at the interpolation nodes x_0, $x_1 = x_0 + h$ and $x_2 = x_0 + 2h$:

$$\int\limits_{x_0}^{x_0+2h} f(x)\,dx \approx \frac{h}{3}(y_0 + 4y_1 + y_2), \quad |R| \le \frac{h^5}{90}M_4. \tag{19.77}$$

n must be an even number for a complete Simpson formula. The approximation is

$$\int\limits_a^b f(x)\,dx \approx \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n), \tag{19.78}$$

$$|R| \le \frac{(b-a)h^4}{180}M_4.$$

M_4 is an upper bound for $|f^{(4)}(x)|$ on the entire integration domain. The Simpson formula has an error of order 4 and it is exact for polynomials up to third degree.

19.3.2.4 Hermite's Trapezoidal Formula

$f(x)$ is replaced by a polynomial of third degree in the interval $[x_0, x_0 + h]$, which interpolates $f(x)$ and $f'(x)$ at the interplation nodes x_0 and $x_1 = x_0 + h$:

$$\int\limits_{x_0}^{x_0+h} f(x)\,dx \approx \frac{h}{2}(y_0 + y_1) + \frac{h^2}{12}(y_0' - y_1'), \quad |R| \le \frac{h^5}{720}M_4. \tag{19.79}$$

The *Hermite trapezoidal formula* is obtained by summation:

$$\int\limits_a^b f(x)\,dx \approx h\left(\frac{y_0}{2} + y_1 + y_2 + \cdots + y_{n-1} + \frac{y_n}{2}\right) + \frac{h^2}{12}(y_0' - y_n'), \quad |R| \le \frac{(b-a)h^4}{720}M_4. \tag{19.80}$$

M_4 denotes an upper bound for $|f^{(4)}(x)|$ on the entire integration domain. The Hermite trapezoidal formula has an error of order 4 and it is exact for polynomials up to third degree.

19.3.3 Quadrature Formulas of Gauss

Quadrature formulas of Gauss have the general form

$$\int\limits_a^b f(x)\,dx \approx \sum_{\nu=0}^{n} c_\nu y_\nu \text{ with } y_\nu = f(x_\nu) \tag{19.81}$$

where not only the coefficients c_ν are considered as parameters but also the interpolation nodes x_ν. These parameters are determined in order to make the formula (19.81) exact for polynomials of the highest possible degree.
The quadrature formulas of Gauss result in very accurate approximations, but the interpolation nodes must be chosen in a very special way.

19.3.3.1 Gauss Quadrature Formulas

If the integration interval in (19.81) is chosen as $[a, b] = [-1, 1]$, and the interpolation nodes are chosen as the roots of the Legendre polynomials (see 9.1.2.6, **3.**, p. 566, 21.12, p. 1108), then the coefficients c_ν can be determined so that the formula (19.81) gives the exact value for polynomials up to degree $2n + 1$. The roots of the Legendre polynomials are symmetric with respect to the origin. For the cases $n = 1, 2$ and 3 they are:

$$\begin{array}{lll} n=1: & x_0 = -x_1, & c_0 = 1, \\ & x_1 = \dfrac{1}{\sqrt{3}} = 0.577\,350\,269\ldots, & c_1 = 1. \\ n=2: & x_0 = -x_2, & c_0 = \dfrac{5}{9}, \\ & x_1 = 0, & c_1 = \dfrac{8}{9}, \\ & x_2 = \sqrt{\dfrac{3}{5}} = 0.774\,596\,669\ldots, & c_2 = c_0. \\ n=3: & x_0 = -x_3, & c_0 = 0.347\,854\,854\ldots, \\ & x_1 = -x_2, & c_1 = 0.652\,145\,154\ldots, \\ & x_2 = 0.339\,981\,043\ldots, & c_2 = c_1, \\ & x_3 = 0.861\,136\,311\ldots, & c_3 = c_0. \end{array} \tag{19.82}$$

Remark: A general integration interval $[a,b]$ can be transformed into $[-1,1]$ by the transformation $t = \frac{b-a}{2}x + \frac{a+b}{2}$ $(t \in [a,b], x \in [-1,1])$. Then

$$\int_a^b f(t)\,dt \approx \frac{b-a}{2}\sum_{\nu=0}^{n} c_\nu f\left(\frac{b-a}{2}x_\nu + \frac{a+b}{2}\right) \tag{19.83}$$

with the values x_ν and c_ν given above for the interval $[-1,1]$.

19.3.3.2 Lobatto's Quadrature Formulas

In some cases it is reasonable also to choose the endpoints of the subintervals as interpolation nodes. Then, there are $2n$ more free parameters in (19.81). These values can be determined so that polynomials up to degree $2n-1$ can be integrated exactly. In the cases $n=2$ and $n=3$:

$n = 2$:

$$\begin{array}{ll} x_0 = -1, & c_0 = \dfrac{1}{3}, \\ x_1 = 0, & c_1 = \dfrac{4}{3}, \\ x_2 = 1, & c_2 = c_0. \end{array} \tag{19.84a}$$

$n = 3$:

$$\begin{array}{ll} x_0 = -1, & c_0 = \dfrac{1}{6}, \\ x_1 = -x_2, & c_1 = \dfrac{5}{6}, \\ x_2 = \dfrac{1}{\sqrt{5}} = 0.447\,213\,595\ldots, & c_2 = c_1, \\ x_3 = 1, & c_3 = c_0. \end{array} \tag{19.84b}$$

The case $n=2$ represents the Simpson formula.

19.3.4 Method of Romberg

To increase the accuracy of numerical integration the method of Romberg can be recommended, where one starts with a sequence of trapezoid sums, which is obtained by repeated halving of the integration step size.

19.3.4.1 Algorithm of the Romberg Method

The method consists of the following steps:

1. Trapezoid sums determination

The trapezoid sum $T(h_i)$ according to (19.76) in 19.3.2.2, p. 964 is determined as an approximation of the integral $\int_a^b f(x)\,dx$ with the step sizes

$$h_i = \frac{b-a}{2^i} \quad (i = 0,1,2,\ldots,m). \tag{19.85}$$

Here, the recursive relation

$$\begin{aligned} T(h_i) = T\left(\frac{h_{i-1}}{2}\right) &= \frac{h_{i-1}}{2}\left[\frac{1}{2}f(a) + f\left(a + \frac{h_{i-1}}{2}\right) + f(a + h_{i-1}) + f\left(a + \frac{3}{2}h_{i-1}\right)\right. \\ &\left. + f(a + 2h_{i-1}) + \cdots + f\left(a + \frac{2n-1}{2}h_{i-1}\right) + \frac{1}{2}f(b)\right] \\ &= \frac{1}{2}T(h_{i-1}) + \frac{h_{i-1}}{2}\sum_{j=0}^{n-1} f\left(a + \frac{h_{i-1}}{2} + jh_{i-1}\right) \quad (i = 1, 2, \ldots, m;\ n = 2^{i-1}) \end{aligned} \tag{19.86}$$

is considered. Recursion formula (19.86) tells that for the calculation of $T(h_i)$ from $T(h_{i-1})$ the function values must be calculated only at the new interpolation nodes.

2. Triangular Scheme

$T_{0i} = T(h_i)$ $(i = 0, 1, 2, \ldots)$ is substituted and the values

$$T_{ki} = T_{k-1,i} + \frac{T_{k-1,i} - T_{k-1,i-1}}{4^k - 1} \quad (k = 1, 2, \ldots, m;\ i = k, k+1, \ldots) \tag{19.87}$$

are calculated recursively. The arrangement of the values calculated according to (19.87) is most practical in a triangular scheme, whose elements are calculated in a column-wise manner:

$$\begin{array}{llll} T(h_0) = T_{00} & & & \\ T(h_1) = T_{01} & T_{11} & & \\ T(h_2) = T_{02} & T_{12} & T_{22} & \\ T(h_3) = T_{03} & T_{13} & T_{23} & T_{33} \\ \cdots\cdots\cdots & \cdots & \cdots & \cdots \end{array} \tag{19.88}$$

The scheme will be continued downwards (with a fixed number of columns) until the lower values at the right are almost the same. The values T_{1i} $(i = 1, 2, \ldots)$ of the second column correspond to those calculated by the Simpson formula.

19.3.4.2 Extrapolation Principle

The Romberg method represents an application of the so-called *extrapolation principle*. This will be demonstrated by deriving the formula (19.87) for the case $k = 1$. The required integral is denoted by I, the corresponding trapezoid sum (19.76) by $T(h)$. If the integrand of I is $(2m + 2)$ times continuously differentiable in the integration interval, then it can be shown that an *asymptotical expansion* with respect to h is valid for the error R of the quadrature formula, and it has the form

$$R(h) = I - T(h) = a_1h^2 + a_2h^4 + \cdots + a_mh^{2m} + O(h^{2m+2}) \tag{19.89a}$$

or

$$T(h) = I - a_1h^2 - a_2h^4 - \cdots - a_mh^{2m} + O(h^{2m+2}). \tag{19.89b}$$

The coefficients a_1, a_2,. . . ,a_m are constants and independent of h.

$T(h)$ and $T\left(\frac{h}{2}\right)$ are formed according to (19.89b) and the linear combination

$$T_1(h) = \alpha_1T(h) + \alpha_2T\left(\frac{h}{2}\right) = (\alpha_1 + \alpha_2)I - a_1\left(\alpha_1 + \frac{\alpha_2}{4}\right)h^2 - a_2\left(\alpha_1 + \frac{\alpha_2}{16}\right)h^4 - \cdots \tag{19.90}$$

is considered. If $\alpha_1 + \alpha_2 = 1$ and $\alpha_1 + \frac{\alpha_2}{4} = 0$ are substituted, then $T_1(h)$ has an error of order 4, while $T(h)$ and $T(h/2)$ both have errors of order only 2. The formula is

$$T_1(h) = -\frac{1}{3}T(h) + \frac{4}{3}T\left(\frac{h}{2}\right) = T\left(\frac{h}{2}\right) + \frac{T\left(\frac{h}{2}\right) - T(h)}{3}. \tag{19.91}$$

This is the formula (19.87) for $k = 1$. Repeated application of the above procedure results in the approximation T_{ki} according to (19.87) and

$$T_{ki} = I + O(h_i^{2k+2}). \tag{19.92}$$

■ The definite integral $I = \int_0^1 \frac{\sin x}{x}\,dx$ (integral sine, see 8.2.5, **1.**, p. 513) cannot be obtained in an elementary way. Calculate the approximate values of this integral (calculating for 8 digits).

1. Romberg method:

$k = 0$	$k = 1$	$k = 2$	$k = 3$
0.92073549			
0.93979328	0.94614588		
0.94451352	0.94608693	0.94608300	
0.94569086	0.94608331	0.94608307	$\underline{\underline{0.94608307.}}$

The Romberg method results in the approximation value 0.94608307. The value calculated for 10 digits is 0.9460830704. The order $O\left((1/8)^8\right) \approx 6 \cdot 10^{-8}$ of the error according to (19.92) is verified.

2. Trapezoidal and Simpson Formulas: From the scheme of the Romberg method it can be got directly that for $h_3 = 1/8$ the trapezoid formula has the approximation value 0.94569086 and the Simpson formula gives the value 0.94608331.
The correction of the trapezoidal formula by Hermite according to (19.79) results in the value $I \approx 0.94569086 + \frac{0.30116868}{64 \cdot 12} = 0.94608301$.

3. Gauss Formula: By the formula (19.83) we get for

$$n = 1:\quad I \approx \frac{1}{2}\left[c_0 f\left(\frac{1}{2}x_0 + \frac{1}{2}\right) + c_1 f\left(\frac{1}{2}x_1 + \frac{1}{2}\right)\right] = 0.94604113;$$

$$n = 2:\quad I \approx \frac{1}{2}\left[c_0 f\left(\frac{1}{2}x_0 + \frac{1}{2}\right) + c_1 f\left(\frac{1}{2}x_1 + \frac{1}{2}\right) + c_2 f\left(\frac{1}{2}x_2 + \frac{1}{2}\right)\right] = 0.94608313;$$

$$n = 3:\quad I \approx \frac{1}{2}\left[c_0 f\left(\frac{1}{2}x_0 + \frac{1}{2}\right) + \cdots + c_3 f\left(\frac{1}{2}x_3 + \frac{1}{2}\right)\right] = 0.94608307.$$

It can be observed that the Gauss formula results in an 8-digit exact approximation value for $n = 3$, i.e., with only four function values. With the trapezoidal rule this accuracy would need a very large number (> 1000) of function values.

Remarks:

1. *Fourier analysis* has an important role in integrating periodic functions (see 7.4.1.1, **1.**, p. 474). The details of numerical realizations can be found under the title of *harmonic analysis* (see 19.6.4, p. 992). The actual computations are based on the so-called *Fast Fourier Transformation* FFT (see 19.6.4.2, p. 993).

2. In many applications it is useful to take the special properties of the integrands under consideration. Further integration routines can be developed for such special cases. A large variety of convergence properties, error analysis, and optimal integration formulas is discussed in the literature (see, e.g., [19.4]).

3. Numerical methods to find the values of *multiple integrals* are discussed in the literature (see, e.g., [19.26]).

19.4 Approximate Integration of Ordinary Differential Equations

In many cases, the solution of an ordinary differential equation cannot be given in closed form as an expression of known elementary functions. The solution, which still exists under rather general circumstances (see 9.1.1.1, p. 540), must be determined by numerical methods. These result only in particular solutions, but it is possible to reach high accuracy. Since differential equations of higher order than one can be either initial value problems or boundary value problems, numerical methods were developed for both types of problems.

19.4.1 Initial Value Problems

The principle of the methods presented in the following discussion to solve initial value problems

$$y' = f(x,y), \quad y(x_0) = y_0 \tag{19.93}$$

is to give approximate values y_i for the unknown function $y(x)$ at a chosen set of interpolation points x_i. Usually, *equidistant interpolation nodes* are considered with a previously given *step size* h:

$$x_i = x_0 + ih \quad (i = 0, 1, 2, \ldots). \tag{19.94}$$

19.4.1.1 Euler Polygonal Method

An integral representation of the initial value problem (19.93) is given by integration

$$y(x) = y_0 + \int_{x_0}^{x} f(x, y(x))\,dx. \tag{19.95}$$

This is the starting point for the approximation

$$y(x_1) = y_0 + \int_{x_0}^{x_0+h} f(x, y(x))\,dx \approx y_0 + hf(x_0, y_0) = y_1, \tag{19.96}$$

which is generalized as the *Euler broken line method* or *Euler polygonal method*:

$$y_{i+1} = y_i + hf(x_i, y_i) \quad (i = 0, 1, 2, \ldots\,;\ y(x_0) = y_0). \tag{19.97}$$

For a geometric interpretation see **Fig. 19.5**. Comparison of (19.96) with the Taylor expansion

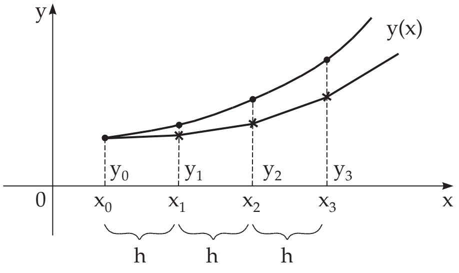


Figure 19.5

$$y(x_1) = y(x_0 + h) = y_0 + f(x_0, y_0)h + \frac{y''(\xi)}{2}h^2 \tag{19.98}$$

with $x_0 < \xi < x_0 + h$ shows that the approximation y_1 has an error of order h^2. The accuracy can be improved by reducing the step size h. Practical calculations show that halving the step size h results in halving the error of the approximations y_i.

A quick overview of the approximate shape of the solution curve can be got by using the Euler method.

19.4.1.2 Runge-Kutta Methods

1. Calculation Scheme

The equation $y'(x) = f(x, y)$ determines at every point (x_0, y_0) a direction, the direction of the tangent line of the solution curve passing through the point (x_0, y_0). The Euler method follows this direction until the next interpolation node. The Runge-Kutta methods consider more points "between" (x_0, y_0)

and the possible next point (x_0+h, y_1) of the curve, and depending on the appropriate choice of these additional points more accurate values are obtained for y_1. There exist Runge-Kutta methods of different orders depending on the number and the arrangements of these "auxiliary" points. Here a fourth-order method (see 19.4.1.5, **1.**, p. 972) is shown. (The Euler method is a first-order Runge-Kutta method.)

The calculation scheme of fourth order for the step from x_0 to $x_1 = x_0 + h$ to get an approximate value for y_1 of (19.93) is given in (19.99). The further steps follow the same scheme.

The error of this Runge-Kutta method has order h^5 (at every step) according to (19.99), so with an appropriate choice of the step size high accuracy can be obtained.

x	y	$k = h \cdot f(x,y)$
x_0	y_0	k_1
$x_0 + h/2$	$y_0 + k_1/2$	k_2
$x_0 + h/2$	$y_0 + k_2/2$	k_3
$x_0 + h$	$y_0 + k_3$	k_4
$x_1 = x_0 + h$	$y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$	

(19.99)

■ $y' = \frac{1}{4}(x^2 + y^2)$ with $y(0) = 0$. $y(0.5)$ is determined in one step, i.e. $h = 0.5$ (see the table on the right). The exact value for 8 digits is 0.01041860.

x	y	$k = \frac{1}{8}(x^2 + y^2)$
0	0	0
0.25	0	0.00781250
0.25	0.00390625	0.00781441
0.5	0.00781441	0.03125763
0.5	$\underline{\underline{0.01041858}}$	

2. Remarks

1. For the special differential equation $y' = f(x)$, this Runge-Kutta method becomes the Simpson formula (see 19.3.2.3, p. 965).

2. For a large number of integration steps, a change of step size is possible or sometimes necessary. The change of step size can be decided by checking the accuracy so that the step is repeated with a double step size $2h$. If, e.g., the approximate value is $y_2(h)$ for $y(x_0+2h)$ (calculated by the single step size) and $y_2(2h)$ (calculated by the doubled step size), then the estimation for the error $R_2(h) = y(x_0+2h) - y_2(h)$ is

$$R_2(h) \approx \frac{1}{15}[y_2(h) - y_2(2h)]. \qquad (19.100)$$

Information about the implementation of the step size changes can be found in the literature (see [19.24]).

3. Runge-Kutta methods can easily be used also for higher-order differential equations, see [19.24]. Higher-order differential equations can be rewritten in a first-order differential equation system (see p. 550). Then, the approximation methods are performed as parallel calculations according to (19.99), as the differential equations are connected to each other.

19.4.1.3 Multi-Step Methods

The Euler method (19.97) and the Runge-Kutta method (19.99) are so-called *single-step methods*, since we start only from y_i in the calculation of y_{i+1}. In general, *linear multi-step methods* have the form

$$\begin{aligned} &y_{i+k} + \alpha_{k-1}y_{i+k-1} + \alpha_{k-2}y_{i+k-2} + \cdots + \alpha_1 y_{i+1} + \alpha_0 y_i \\ &= h(\beta_k f_{i+k} + \beta_{k-1}f_{i+k-1} + \cdots + \beta_1 f_{i+1} + \beta_0 f_i) \end{aligned} \qquad (19.101)$$

with appropriately chosen constants α_j and β_j ($j = 0, 1, \ldots, k$; $\alpha_k = 1$). The formula (19.101) is called a *k-step method* if $|\alpha_0| + |\beta_0| \neq 0$. It is called *explicit*, if $\beta_k = 0$, since in this case the values $f_{i+j} = f(x_{i+j}, y_{i+j})$ on the right-hand side of (19.101) only contain the already known approximation values y_i, $y_{i+1}, \ldots, y_{i+k-1}$. If $\beta_k \neq 0$ holds, the method is called *implicit*, since then the required new value y_{i+k} occurs on both sides of (19.101).

The k initial values $y_0, y_1, \ldots, y_{k-1}$ must be known in the application of a k-step method. These initial values can be got, e.g., by one-step methods.

A special multi-step method to solve the initial value problem (19.93) can be derived if the derivative

$y'(x_i)$ in (19.93) is replaced by a *difference formula* (see 9.1.1.5, **1.**, p. 549) or if the integral in (19.95) is approximated by a *quadrature formula* (see 19.3.1, p. 963).

Examples of special multi-step methods are:

1. Midpoint Rule The derivative $y'(x_{i+1})$ in (19.93) is replaced by the slope of the secant line between the interpolation nodes x_i and x_{i+2}, i.e.:

$$y_{i+2} - y_i = 2hf_{i+1}. \tag{19.102}$$

2. Rule of Milne The integral in (19.95) is approximated by the Simpson formula:

$$y_{i+2} - y_i = \frac{h}{3}(f_i + 4f_{i+1} + f_{i+2}). \tag{19.103}$$

3. Rule of Adams and Bashforth The integrand in (19.95) is replaced by the interpolation polynomial of Lagrange (see 19.6.1.2, p. 983) based on the k interpolation nodes x_i, $x_{i+1}, \ldots, x_{i+k-1}$. Integrating between x_{i+k-1} and x_{i+k} results in

$$y_{i+k} - y_{i+k-1} = \sum_{j=0}^{k-1}\left[\int\limits_{x_{i+k-1}}^{x_{i+k}} L_j(x)\,dx\right] f(x_{i+j}, y_{i+j}) = h\sum_{j=0}^{k-1}\beta_j f(x_{i+j}, y_{i+j}). \tag{19.104}$$

The method (19.104) is explicit for y_{i+k}. For the calculation of the coefficients β_j see [19.2].

19.4.1.4 Predictor-Corrector Method

In practice, implicit multi-step methods have a great advantage compared to explicit ones in that they allow much larger step sizes with the same accuracy. But, an implicit multi-step method usually requires the solution of a non-linear equation to get the approximation value y_{i+k}. This follows from (19.101) and has the form

$$y_{i+k} = h\sum_{j=0}^{k}\beta_j f_{i+j} - \sum_{j=0}^{k-1}\alpha_j y_{i+j} = F(y_{i+k}). \tag{19.105}$$

The solution of (19.105) is an iterative one. The procedure is the following: An initial value $y_{i+k}^{(0)}$ is determined by an explicit formula, the so-called *predictor*. Then it will be corrected by an iteration rule

$$y_{i+k}^{(\mu+1)} = F(y_{i+k}^{(\mu)}) \quad (\mu = 0, 1, 2, \ldots), \tag{19.106}$$

which is called the *corrector* coming from the implicit method. Special predictor–corrector formulas are:

1. $$y_{i+1}^{(0)} = y_i + \frac{h}{12}(5f_{i-2} - 16f_{i-1} + 23f_i), \tag{19.107a}$$

$$y_{i+1}^{(\mu+1)} = y_i + \frac{h}{12}(-f_{i-1} + 8f_i + 5f_{i+1}^{(\mu)}) \quad (\mu = 0, 1, \ldots); \tag{19.107b}$$

2. $$y_{i+1}^{(0)} = y_{i-2} + 9y_{i-1} - 9y_i + 6h(f_{i-1} + f_i), \tag{19.108a}$$

$$y_{i+1}^{(\mu+1)} = y_{i-1} + \frac{h}{3}(f_{i-1} + 4f_i + f_{i+1}^{(\mu)}) \quad (\mu = 0, 1, \ldots). \tag{19.108b}$$

The Simpson formula as the corrector in (19.108b) is numerically unstable and it can be replaced, e.g., by

$$y_{i+1}^{(\mu+1)} = 0.9y_{i-1} + 0.1y_i + \frac{h}{24}(0.1f_{i-2} + 6.7f_{i-1} + 30.7f_i + 8.1f_{i+1}^{(\mu)}). \tag{19.109}$$

19.4.1.5 Convergence, Consistency, Stability

1. Global Discretization Error and Convergence

Single-step methods can be written generally in the form:

$$y_{i+1} = y_i + hF(x_i, y_i, h) \quad (i = 0, 1, 2, \ldots;\ y_0 \text{ given}). \tag{19.110}$$

Here $F(x, y, h)$ is called the *increment function* or progressive direction of the single-step method. The approximating solution obtained by (19.110) depends on the step size h and it should be denoted by $y(x, h)$. Its difference from the exact solution $y(x)$ of the initial value problem (19.93) is called the *global discretization error* $g(x, h)$ (see (19.111)), and we say: The single-step method (19.110) is *convergent with order* p if p is the largest natural number such that

$$g(x, h) = y(x, h) - y(x) = O(h^p) \tag{19.111}$$

holds. Formula (19.111) says that the approximation $y(x, h)$ determined with the step size $h = \dfrac{x - x_0}{n}$ converges to the exact solution $y(x)$ for every x from the domain of the initial value problem if $h \to 0$.

■ The Euler method (19.97) has order of convergence $p = 1$. For the Runge-Kutta method (19.99) $p = 4$ holds.

2. Local Discretization Error and Consistency

The order of convergence according to (19.111) shows how well the approximating solution $y(x, h)$ approximates the exact solution $y(x)$. Beside this, it is an interesting question of how well the increment function $F(x, y, h)$ approximates the derivative $y' = f(x, y)$. For this purpose the so-called *local discretization error* $l(x, h)$ (see (19.112)) is introduced. The single-step method (19.110) is *consistent with order* p, if p is the largest natural number with

$$l(x, h) = \frac{y(x + h) - y(x)}{h} - F(x, y, h) = O(h^p). \tag{19.112}$$

It follows directly from (19.112) that for a consistent single-step method

$$\lim_{h \to 0} F(x, y, h) = f(x, y). \tag{19.113}$$

■ The Euler method has order of consistency $p = 1$, the Runge-Kutta method in (19.99) has order of consistency $p = 4$.

3. Stability with Respect to Perturbation of the Initial Values

In the practical performance of a single-step method, a rounding error $O(1/h)$ adds to the global discretization error $O(h^p)$. Consequently, we have to select a not too small, finite step size $h > 0$. It is also an important question of how the numerical solution y_i behaves under perturbations of the initial values or in the case $x_i \to \infty$.

In the theory of ordinary differential equations, an initial value problem (19.93) is called *stable with respect to perturbations of its initial values* if:

$$|\tilde{y}(x) - y(x)| \le |\tilde{y}_0 - y_0|. \tag{19.114}$$

Here $\tilde{y}(x)$ is the solution of (19.93) with the perturbed initial value $\tilde{y}(x_0) = \tilde{y}_0$ instead of y_0. Estimation (19.114) tells that the absolute value of the difference of the solutions is not larger than the perturbation of the initial values.

In general, it is hard to check (19.114). Therefore the *linear test problem*

$$y' = \lambda y \ \text{ with } \ y(x_0) = y_0 \quad (\lambda \ \text{constant}, \ \lambda \le 0) \tag{19.115}$$

is considered which is stable, and a single-step method is applied to this special initial value problem. A consistent method is called *absolutely stable* with step size $h > 0$ with respect to perturbed initial values if the approximating solution y_i of the above linear test problem (19.115) obtained by using the method satisfies the condition

$$|y_i| \le |y_0|. \tag{19.116}$$

■ Applying the Euler polygon method for equation (19.115) results in the solution $y_{i+1} = (1 + \lambda h)y_i$ $(i = 0, 1, \ldots)$. Obviously, (19.116) holds if $|1 + \lambda h| \le 1$, and so the step size must satisfy $-2 \le \lambda h \le 0$.

4. Stiff Differential Equations

Many application problems, including those in chemical kinetics, can be modeled by differential equations whose solutions consist of terms converging to zero exponentially but in a high different kind of exponential decreasing. These equations are called *stiff differential equations*. For example:

$$y(x) = C_1 e^{\lambda_1 x} + C_2 e^{\lambda_2 x} \quad (C_1, C_2, \lambda_1, \lambda_2 \text{ const}) \tag{19.117}$$

with $\lambda_1 < 0$, $\lambda_2 < 0$ and $|\lambda_1| \ll |\lambda_2|$, e.g., $\lambda_1 = -1$, $\lambda_2 = -1000$. The term with λ_2 does not have a significant affect on the solution function, but it does in selecting the step size h for a numerical method. In such cases the choice of the most appropriate numerical method has special importance (see [19.23]).

19.4.2 Boundary Value Problems

The most important methods for solving boundary value problems of ordinary differential equations will be demonstrated on the following simple linear boundary value problem for a differential equation of the second order:

$$y''(x) + p(x)y'(x) + q(x)y(x) = f(x) \quad (a \le x \le b) \text{ with } y(a) = \alpha,\ y(b) = \beta. \tag{19.118}$$

The functions $p(x)$, $q(x)$ and $f(x)$ and also the constants α and β are given.
The given method can also be adapted for boundary value problems of higher-order differential equations.

19.4.2.1 Difference Method

The interval $[a, b]$ is subdivided by equidistant interpolation points $x_\nu = x_0 + \nu h$ $(\nu = 0, 1, 2, \ldots, n;$ $x_0 = a,\ x_n = b)$ and the values of the derivatives are substituted into the differential equation at the interior interpolation points

$$y''(x_\nu) + p(x_\nu)y'(x_\nu) + q(x_\nu)y(x_\nu) = f(x_\nu) \quad (\nu = 1, 2, \ldots, n-1) \tag{19.119}$$

by so-called *finite divided differences*, e.g.:

$$y'(x_\nu) \approx y'_\nu = \frac{y_{\nu+1} - y_{\nu-1}}{2h}, \tag{19.120a}$$

$$y''(x_\nu) \approx y''_\nu = \frac{y_{\nu+1} - 2y_\nu + y_{\nu-1}}{h^2}. \tag{19.120b}$$

In this way $n-1$ linear equations are obtained for the $n-1$ approximation values $y_\nu \approx y(x_\nu)$ in the interior of the integration interval $[a, b]$, considering the conditions $y_0 = \alpha$ and $y_n = \beta$. If the boundary conditions also contain derivatives, they must also be replaced by finite expressions.

Eigenvalue problems of differential equations (see 9.1.3.2, p. 569) are handled analogously. The application of the *difference method*, described by (19.119) and (19.120a,b), leads to a matrix eigenvalue problem (see 4.6, p. 314).

■ The solution of the homogeneous differential equation $y'' + \lambda^2 y = 0$ with boundary conditions $y(0) = y(1) = 0$ leads to a matrix eigenvalue problem. The difference method transforms the differential equation into the difference equation $y_{\nu+1} - 2y_\nu + y_{\nu-1} + h^2\lambda^2 y_\nu = 0$. If three interior points are chosen, hence $h = 1/4$, then considering $y_0 = y(0) = 0$, $y_4 = y(1) = 0$ the discrete system is

$$\begin{aligned}
\left(-2 + \frac{\lambda^2}{16}\right) y_1 + {} & y_2 & & = 0,\\
y_1 + {} & \left(-2 + \frac{\lambda^2}{16}\right) y_2 + {} & y_3 & = 0,\\
& y_2 + {} & \left(-2 + \frac{\lambda^2}{16}\right) y_3 & = 0.
\end{aligned}$$

This homogeneous system of equations has a non-trivial solution only when the coefficient determinant is zero. This condition results in the eigenvalues ${\lambda_1}^2 = 9.37$, ${\lambda_2}^2 = 32$ and ${\lambda_3}^2 = 54.63$. Among them only the smallest one is close to its corresponding true value 9.87.

Remark: The accuracy of the difference method can be improved by

1. decreasing the step size h,

2. application of a derivative approximation of higher order (approximations as (19.120a,b) have an error of order $O(h^2)$),

3. application of multi-step methods (see 19.4.1.3, p. 970).

If the problem is a non-linear boundary value problem, then the difference method leads to a system of non-linear equations of the unknown approximation values y_ν (see 19.2.2, p. 961).

19.4.2.2 Approximation by Using Given Functions

The approximate solution of the boundary value problem (19.118) is a linear combination of suitably chosen functions $g_i(x)$, which are linearly independent and each one satisfies the boundary value conditions:

$$y(x) \approx g(x) = \sum_{i=1}^{n} a_i g_i(x). \tag{19.121}$$

Substituting $g(x)$ into the differential equation (19.118) results in an error, the so-called *defect*

$$\varepsilon(x; a_1, a_2, \ldots, a_n) = g''(x) + p(x)g'(x) + q(x)g(x) - f(x). \tag{19.122}$$

To determine the coefficients a_i the following principles (see also p. 978) can be used:

1. Collocation Method The defect has to be zero at n given points x_ν, the so-called *collocation points*. The conditions

$$\varepsilon(x_\nu; a_1, a_2, \ldots, a_n) = 0 \quad (\nu = 1, 2, \ldots, n), \quad a < x_1 < x_2 < \ldots < x_n < b \tag{19.123}$$

result in a linear system of equations for the unknown coefficients.

2. Least Squares Method The integral

$$F(a_1, a_2, \ldots, a_n) = \int_a^b \varepsilon^2(x; a_1, a_2, \ldots, a_n)\, dx, \tag{19.124}$$

depending on the coefficients, should be minimal. The necessary conditions

$$\frac{\partial F}{\partial a_i} = 0 \quad (i = 1, 2, \ldots, n) \tag{19.125}$$

give a linear system of equations for the coefficients a_i.

3. Galerkin Method The requirement is that the so-called *error orthogonality* is satisfied, i.e.,

$$\int_a^b \varepsilon(x; a_1, a_2, \ldots, a_n) g_i(x)\, dx = 0 \quad (i = 1, 2, \ldots, n), \tag{19.126}$$

and in this way a linear system of equations is obtained for the unknown coefficients.

4. Ritz Method The solution $y(x)$ often has the property that it minimizes the *variational integral*,

$$I[y] = \int_a^b H(x, y, y')\, dx \tag{19.127}$$

(see (10.4), p. 610). If the function $H(x, y, y')$ is known, then $y(x)$ is replaced by the approximation $g(x)$ as in (19.121) and $I[y] = I(a_1, a_2, \ldots, a_n)$ is minimized. The necessary conditions

$$\frac{\partial I}{\partial a_i} = 0 \quad (i = 1, 2, \ldots, n) \tag{19.128}$$

result in n equation for the coefficients a_i.

■ Under certain conditions on the functions p, q, f and y, the boundary value problem

$$-[p(x)y'(x)]' + q(x)y(x) = f(x) \text{ with } y(a) = \alpha,\, y(b) = \beta \tag{19.129}$$

and the variational problem

$$I[y] = \int_a^b [p(x)y'^2(x) + q(x)y^2(x) - 2f(x)y(x)]\,dx = \text{min! with } y(a) = \alpha,\, y(b) = \beta \tag{19.130}$$

are equivalent, so $H(x, y, y')$ can be got immediately from (19.130) for the boundary value problem of the form (19.129).

Instead of the approximation (19.121), one often considers

$$g(x) = g_0(x) + \sum_{i=1}^{n} a_i g_i(x), \tag{19.131}$$

where $g_0(x)$ satisfies the boundary values and the functions $g_i(x)$ satisfy the conditions

$$g_i(a) = g_i(b) = 0 \quad (i = 1, 2, \ldots, n). \tag{19.132}$$

For the problem (19.118), an appropriate choice is, e.g.,

$$g_0(x) = \alpha + \frac{\beta - \alpha}{b - a}(x - a). \tag{19.133}$$

Remark: In a linear boundary value problem, the forms (19.121) and (19.131) result in a linear system of equations for the coefficients. In the case of non-linear boundary value problems non-linear systems of equations are obtained, which can be solved by the methods given in Section 19.2.2, p. 961.

19.4.2.3 Shooting Method

With the shooting method, the solution of a boundary value problem is reduced to the solution of an initial value problem. The basic idea of the method is described below as the *single-target method*.

1. Single-Target Method

The initial value problem

$$y'' + p(x)y' + q(x)y = f(x) \text{ with } y(a) = \alpha,\, y'(a) = s \tag{19.134}$$

is associated to the boundary value problem (19.118). Here s is a parameter, from which the solution y of the initial-value problem (19.134) depends on, i.e., $y = y(x, s)$ holds. The function $y(x, s)$ satisfies the first boundary condition $y(a, s) = \alpha$ according to (19.134). The parameter s should be determined so that $y(x, s)$ satisfies the second boundary condition $y(b, s) = \beta$. Therefore, one has to solve the equation

$$F(s) = y(b, s) - \beta, \tag{19.135}$$

and the regula falsi (or secant) method is an appropriate method to do this. It needs only the values of the function $F(s)$, but the computation of every function value requires the solution of an initial value problem (19.134) until $x = b$ for the special parameter value s with one of the methods given in 19.4.1.

2. Multiple-Target Method

In a so-called *multiple-target method*, the integration interval $[a, b]$ is divided into subintervals, and we use the single-target method on every subinterval. Then, the required solution is composed from the solutions of the subintervals, where the continuous transition at the endpoints of the subintervals must be ensured.

This requirement results in further conditions. For the numerical implementation of the multiple-target method, which is used mostly for non-linear boundary value problems, see. [19.24].

19.5 Approximate Integration of Partial Differential Equations

In this section only the principles of numerical solutions of partial differential equations are discussed using the example of linear second-order partial differential equations with two independent variables with the corresponding boundary or/and initial conditions.

19.5.1 Difference Method

A regular grid is considered on the integration domain by the chosen points (x_μ, y_ν). Usually, this grid is chosen to be rectangular and equally spaced:

$$x_\mu = x_0 + \mu h, \quad y_\nu = y_0 + \nu l \quad (\mu, \nu = 1, 2, \ldots). \tag{19.136}$$

It results in squares for $l = h$. If the required solution is denoted by $u(x, y)$, then the partial derivatives occurring in the differential equation and in the boundary or initial conditions are replaced by *finite divided differences* in the following way, where $u_{\mu\nu}$ denotes an approximate value for the function value $u(x_\mu, y_\nu)$:

Partial Derivative	**Finite Divided Difference**	**Order of Error**
$\frac{\partial u}{\partial x}(x_\mu, y_\nu)$	$\frac{1}{h}(u_{\mu+1,\nu} - u_{\mu,\nu})$ or $\frac{1}{2h}(u_{\mu+1,\nu} - u_{\mu-1,\nu})$	$O(h)$ or $O(h^2)$
$\frac{\partial u}{\partial y}(x_\mu, y_\nu)$	$\frac{1}{l}(u_{\mu,\nu+1} - u_{\mu,\nu})$ or $\frac{1}{2l}(u_{\mu,\nu+1} - u_{\mu,\nu-1})$	$O(l)$ or $O(l^2)$
$\frac{\partial^2 u}{\partial x \partial y}(x_\mu, y_\nu)$	$\frac{1}{4hl}(u_{\mu+1,\nu+1} - u_{\mu+1,\nu-1} - u_{\mu-1,\nu+1} + u_{\mu-1,\nu-1})$	$O(hl)$
$\frac{\partial^2 u}{\partial x^2}(x_\mu, y_\nu)$	$\frac{1}{h^2}(u_{\mu+1,\nu} - 2u_{\mu,\nu} + u_{\mu-1,\nu})$	$O(h^2)$
$\frac{\partial^2 u}{\partial y^2}(x_\mu, y_\nu)$	$\frac{1}{l^2}(u_{\mu,\nu+1} - 2u_{\mu,\nu} + u_{\mu,\nu-1})$	$O(l^2)$

(19.137)

The error order in (19.137) is given by using the Landau symbol O.
In some cases, it is more practical to apply the approximation

$$\frac{\partial^2 u}{\partial x^2}(x_\mu, y_\nu) \approx \sigma \frac{u_{\mu+1,\nu+1} - 2u_{\mu,\nu+1} + u_{\mu-1,\nu+1}}{h^2} + (1-\sigma)\frac{u_{\mu+1,\nu} - 2u_{\mu,\nu} + u_{\mu-1,\nu}}{h^2} \tag{19.138}$$

with a fixed parameter σ $(0 \le \sigma \le 1)$. Formula (19.138) represents a convex linear combination of two finite expressions obtained from the corresponding formula (19.137) for the values $y = y_\nu$ and $y = y_{\nu+1}$.

A partial differential equation can be rewritten as a *difference equation* at every interior point of the grid by the formulas (19.137), where the boundary and initial conditions are considered, as well. This system of equations for the approximation values $u_{\mu,\nu}$ has a large dimension for small step sizes h and l, so usually, it is solved by an iteration method (see 19.2.1.4, p. 960).

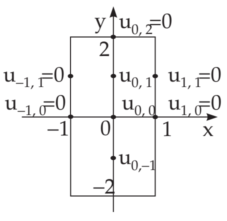


Figure 19.6

■ **A:** The function $u(x, y)$ should be the solution of the differential equation $\Delta u = u_{xx} + u_{yy} = -1$ for the points (x, y) with $|x| < 1$, $|y| < 2$, i.e., in the interior of a rectangle, and it should satisfy the boundary conditions $u = 0$ for $|x| = 1$ and $|y| = 2$. The difference equation corresponding to the differential equation for a square grid with step size h is: $4u_{\mu,\nu} = u_{\mu+1,\nu} + u_{\mu,\nu+1} + u_{\mu-1,\nu} + u_{\mu,\nu-1} + h^2$. The step size $h = 1$ (**Fig. 19.6**) results in a first rough approximation for the function values at the three interior points: $4u_{0,1} = 0 + 0 + 0 + u_{0,0} + 1$, $4u_{0,0} = 0 + u_{0,1} + 0 + u_{0,-1} + 1$, $4u_{0,-1} = 0 + u_{0,0} + 0 + 0 + 1$.

The solution is $u_{0,0} = \frac{3}{7} \approx 0.429$, $u_{0,1} = u_{0,-1} = \frac{5}{14} \approx 0.357$.

■ **B:** The system of equations arising in the application of the difference method for partial differential equations has a very special structure. It is demonstrated by the following example which is a more general boundary value problem. The integration domain is the square G: $0 \le x \le 1$, $0 \le y \le 1$. A function $u(x,y)$ should be determined for which $\Delta u = u_{xx} + u_{yy} = f(x,y)$ in the interior of G, $u(x,y) = g(x,y)$ on the boundary of G. The functions f and g are given. The difference equation associated to this differential equation is, for $h = l = 1/n$:

$$u_{\mu+1,\nu} + u_{\mu,\nu+1} + u_{\mu-1,\nu} + u_{\mu,\nu-1} - 4u_{\mu,\nu} = \frac{1}{n^2} f(x_\mu, y_\nu) \quad (\mu, \nu = 1, 2, \dots, n-1).$$

In the case of $n = 5$, the left-hand side of this system of difference equations for the approximation values $u_{\mu,\nu}$ in the 4×4 interior points has the form (19.139)

$$\left(\begin{array}{cccc}
\boxed{\begin{array}{cccc} -4 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 \\ 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & -4 \end{array}} & \boxed{\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}} & & \mathbf{O} \\
\boxed{\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}} & \boxed{\begin{array}{cccc} -4 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 \\ 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & -4 \end{array}} & \boxed{\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}} & \\
 & \boxed{\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}} & \boxed{\begin{array}{cccc} -4 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 \\ 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & -4 \end{array}} & \boxed{\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}} \\
\mathbf{O} & & \boxed{\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}} & \boxed{\begin{array}{cccc} -4 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 \\ 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & -4 \end{array}}
\end{array}\right)
\left(\begin{array}{c} u_{11} \\ u_{21} \\ u_{31} \\ u_{41} \\ \hline u_{12} \\ u_{22} \\ u_{32} \\ u_{42} \\ \hline u_{13} \\ u_{23} \\ u_{33} \\ u_{43} \\ \hline u_{14} \\ u_{24} \\ u_{34} \\ u_{44} \end{array}\right) \tag{19.139}$$

if the grid is considered row-wise from left to right, and considering that the values of the function are given on the boundary. The coefficient matrix is symmetric and is a *sparse matrix*. This form is called *block-tridiagonal*. It is obvious that the form of the matrix depends on how the grid-points are selected. For different classes of partial differential equations of second order, such as elliptic, parabolic and hyperbolic differential equations, more effective methods have been developed, and also the convergence and stability conditions have been investigated. There is a huge number of books about this topic (see, e.g., [19.22], [19.24]).

19.5.2 Approximation by Given Functions

The solution $u(x,y)$ is approximated by a function in the form

$$u(x,y) \approx v(x,y) = v_0(x,y) + \sum_{i=1}^{n} a_i v_i(x,y). \tag{19.140}$$

Here, two cases are distinguished:

1. $v_0(x,y)$ satisfies the given inhomogeneous differential equation, and the further functions $v_i(x,y)$ $(i = 1, 2, \dots, n)$ satisfy the corresponding homogeneous differential equation (then the linear combination has to be found which approximates the given boundary conditions as well as possible).

2. $v_0(x,y)$ satisfies the inhomogeneous boundary conditions and the other functions $v_i(x,y)$ $(i = 1, 2, \dots, n)$ satisfy the homogeneous boundary conditions (then the linear combination has to be found which approximates the solution of the differential equation on the considered domain as well as possible).

In both cases substituting the approximating function $v(x, y)$ from (19.140) in the first case into the boundary conditions, in the second case into the differential equation results in an error term, the so-called *defect*:

$$\varepsilon = \varepsilon(x, y; a_1, a_2, \ldots, a_n). \tag{19.141}$$

To determine the unknown coefficients a_i one of the following methods can be applied.

1. Collocation Method

The defect ε should be zero in n reasonably distributed points, at the *collocation points* (x_ν, y_ν) $(\nu = 1, 2, \ldots, n)$:

$$\varepsilon(x_\nu, y_\nu; a_1, a_2, \ldots, a_n) = 0 \quad (\nu = 1, 2, \ldots, n). \tag{19.142}$$

The collocation points in the first case are boundary points (it is called *boundary collocation*), in the second case they are interior points of the integration domain (it is called *domain collocation*).
From (19.142) are obtained n equations for the coefficients. Boundary collocation is usually preferred to domain collocation.

■ This method is applied to the example solved in 19.5.1 by the difference method, with the functions satisfying the differential equation:
$v(x, y; a_1, a_2, a_3) = -\frac{1}{4}(x^2 + y^2) + a_1 + a_2(x^2 - y^2) + a_3(x^4 - 6x^2y^2 + y^4)$.
The coefficients are determined to satisfy the boundary conditions at the points $(x_1, y_1) = (1, 0.5)$, $(x_2, y_2) = (1, 1.5)$ and $(x_3, y_3) = (0.5, 2)$ (boundary collocation). The linear system of equations

$$\begin{aligned}
-0.3125 + a_1 + 0.75a_2 - 0.4375a_3 &= 0,\\
-0.8125 + a_1 - 1.25a_2 - 7.4375a_3 &= 0,\\
-1.0625 + a_1 - 3.75a_2 + 10.0625a_3 &= 0
\end{aligned}$$

has the solution $a_1 = 0.4562$, $a_2 = -0.2000$, $a_3 = -0.0143$. The approximate values of the solution can be calculated at arbitrary points with the approximating function. To compare the values with those obtained by the difference method: $v(0, 1) = 0.3919$ and $v(0, 0) = 0.4562$.

2. Least Squares Method

Depending on whether the approximation function (19.140) satisfies the differential equation or the boundary conditions, it is required

1. either the line integral over the boundary C

$$I = \int_{(C)} \varepsilon^2(x(t), y(t)\,;\, a_1, \ldots, a_n)\, dt = \min, \tag{19.143a}$$

where the boundary curve C is given by a parametric representation $x = x(t)$, $y = y(t)$,

2. or the double integral over the domain G

$$I = \iint_{(G)} \varepsilon^2(x, y;\, a_1, \ldots, a_n)\, dx\, dy = \min. \tag{19.143b}$$

From the necessary conditions, $\dfrac{\partial I}{\partial a_i} = 0$ $(i = 1, 2, \ldots, n)$, n equations are obtained for computing the parameters $a_1, a_2, \ldots, a_n$.

19.5.3 Finite Element Method (FEM)

After the appearance of modern computers the finite element methods became the most important technique for solving partial differential equations. These powerful methods give results which are easy to interpret.
Depending on the types of various applications, the FEM is implemented in very different ways, so here only the basic idea is given. It is similar to those used in the Ritz method (see 19.4.2.2, p. 974) for numerical solution of boundary value problems for ordinary differential equations and is related to spline approximations (see 19.7, p. 996).
The finite element method has the following steps:

1. Defining a Variational Problem A variational problem is formulated to the given boundary value problem. The process is demonstrated on the following boundary value problem:

$$\Delta u = u_{xx} + u_{yy} = f \text{ in the interior of } G,\ u = 0 \text{ on the boundary of } G. \tag{19.144}$$

The differential equation in (19.144) is multiplied by an appropriate smooth function $v(x, y)$ vanishing on the boundary of G, and it is integrated over the entire G to get

$$\iint\limits_{(G)} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) v\, dx\, dy = \iint\limits_{(G)} f v\, dx\, dy. \tag{19.145}$$

Applying the Gauss integral formula (see 13.3.3.1, **2.**, p. 725), where $P(x, y) = -vu_y$ and $Q(x, y) = vu_x$ are substituted in (13.121), the *variational equation* from (19.145)

$$a(u, v) = b(v) \tag{19.146a}$$

is obtained with

$$a(u, v) = -\iint\limits_{(G)} \left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y}\right) dx\, dy, \quad b(v) = \iint\limits_{(G)} f v\, dx\, dy. \tag{19.146b}$$

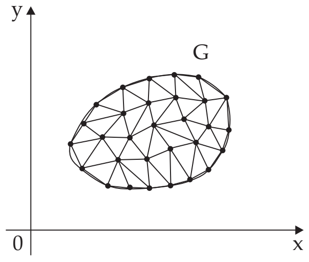


Figure 19.7

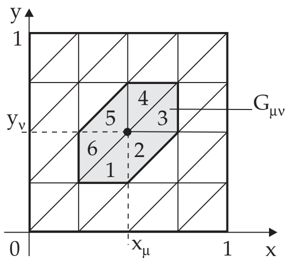


Figure 19.8

2. Triangularization The domain of integration G is decomposed into simple subdomains. Usually, a *triangularization* is used, where G is covered by triangles so that the neighboring triangles have a complete side or only a single vertex in common. Every domain bounded by curves can be approximated quite well by a union of triangles **(Fig. 19.7)**.

Remark: To avoid numerical difficulties, the triangularization should not contain obtuse-angled triangles.

■ A triangularization of the unit square could be performed as shown in **Fig. 19.8**. Here one starts from the grid points with coordinates $x_\mu = \mu h$, $y_\nu = \nu h$ $(\mu, \nu = 0, 1, 2, \ldots, N;\ h = 1/N)$. There are $(N-1)^2$ interior points. Considering the choice of the solution functions, it is always useful to consider the surface elements $G_{\mu\nu}$ composed of the six triangles having the common point (x_μ, y_ν). (In other cases, the number of triangles may differ from six. These surface elements are obviously not mutually exclusive.)

3. Solution A supposed approximating solution is defined for the required function $u(x, y)$ in every triangle. A triangle with the corresponding supposed solution is called a *finite element*. Polynomials in x and y are the most suitable choices. In many cases, the linear approximation

$$\tilde{u}(x, y) = a_1 + a_2 x + a_3 y \tag{19.147}$$

is sufficient. The supposed approximating function must be continuous under the transition from one triangle to neighboring ones, so a continuous final solution arises.

The coefficients a_1, a_2 and a_3 in (19.147) are uniquely defined by the values of the functions u_1, u_2 and u_3 at the three vertices of the triangle. The continuous transition to the neighboring triangles is ensured by this at the same time. The supposed solution (19.147) contains the approximating values u_i of the required function as unknown parameters. For the supposed solution, which is applied as an approximation in the entire domain G for the required solution $u(x, y)$,

$$\tilde{u}(x, y) = \sum_{\mu=1}^{N-1} \sum_{\nu=1}^{N-1} \alpha_{\mu\nu} u_{\mu\nu}(x, y). \tag{19.148}$$

is chosen. The appropriate coefficients $\alpha_{\mu\nu}$ are determined. The following must be valid for the functions $u_{\mu\nu}(x, y)$: They represent a linear function over every triangle of $G_{\mu\nu}$ according to (19.147) with the following conditions:

1. $$u_{\mu\nu}(x_k, y_l) = \begin{cases} 1 & \text{for } k = \mu, l = \nu, \\ 0 & \text{at any other grid point of } G_{\mu\nu}. \end{cases} \tag{19.149a}$$

2. $$u_{\mu\nu}(x, y) \equiv 0 \quad \text{for } (x, y) \notin G_{\mu\nu}. \tag{19.149b}$$

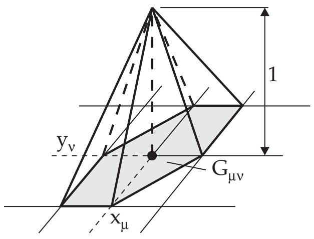


Figure 19.9

The representation of $u_{\mu\nu}(x, y)$ over $G_{\mu\nu}$ is shown in **Fig. 19.9**.
The calculation of $u_{\mu\nu}$ over $G_{\mu\nu}$, i.e., over all triangles 1 to 6 in **Fig. 19.8** is shown here only for triangle 1:

$$u_{\mu\nu}(x, y) = a_1 + a_2 x + a_3 \text{ with} \tag{19.150}$$

$$u_{\mu\nu}(x, y) = \begin{cases} 1 \text{ for } x = x_\mu, y = y_\nu, \\ 0 \text{ for } x = x_{\mu-1}, y = y_{\nu-1}, \\ 0 \text{ for } x = x_\mu, y = y_{\nu-1}. \end{cases} \tag{19.151}$$

From (19.151) $a_1 = 1 - \nu, a_2 = 0,\ a_3 = 1/h$, follow and so for triangle 1

$$u_{\mu\nu}(x, y) = 1 + \left(\frac{y}{h} - \nu\right). \tag{19.152}$$

Analogously, there is:

$$u_{\mu\nu}(x, y) = \begin{cases} 1 - \left(\frac{x}{h} - \mu\right) + \left(\frac{y}{h} - \nu\right) & \text{for triangle 2,} \\ 1 - \left(\frac{x}{h} - \mu\right) & \text{for triangle 3,} \\ 1 - \left(\frac{y}{h} - \nu\right) & \text{for triangle 4,} \\ 1 + \left(\frac{x}{h} - \mu\right) + \left(\frac{y}{h} - \nu\right) & \text{for triangle 5,} \\ 1 + \left(\frac{x}{h} - \mu\right) & \text{for triangle 6.} \end{cases} \tag{19.153}$$

4. Calculation of the Solution Coefficients The solution coefficients $\alpha_{\mu\nu}$ are determined by the requirements that the solution (19.148) satisfies the variational problem (19.146a) for every solution function $u_{\mu\nu}$, i.e., $\tilde{u}(x, y)$ is substituted for $u(x, y)$ and $u_{\mu\nu}(x, y)$ for $v(x, y)$ in (19.146a). In this way, a linear system of equations

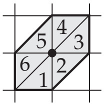
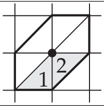
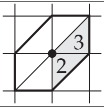
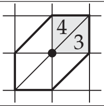
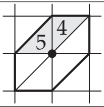
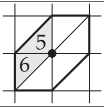
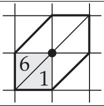
$$\sum_{\mu=1}^{N-1} \sum_{\nu=1}^{N-1} \alpha_{\mu\nu} a(u_{\mu\nu}, u_{kl}) = b(u_{kl}) \quad (k, l = 1, 2, \dots, N-1) \tag{19.154}$$

is obtained for the unknown coefficients, where

$$a(u_{\mu\nu}, u_{kl}) = \iint_{G_{kl}} \left(\frac{\partial u_{\mu\nu}}{\partial x} \frac{\partial u_{kl}}{\partial x} + \frac{\partial u_{\mu\nu}}{\partial y} \frac{\partial u_{kl}}{\partial y} \right) dx\, dy, \quad b(u_{kl}) = \iint_{G_{kl}} f u_{kl}\, dx\, dy. \tag{19.155}$$

In the calculation of $a(u_{\mu\nu}, u_{kl})$ the fact must be kept in mind that integration is needed only in the cases of domains $G_{\mu\nu}$ and G_{kl} with non-empty intersection. These domains are denoted by shadowing in **Table 19.1**.

Table 19.1 Auxiliary table for FEM

Surface region	**Graphical representation**	**Triangle of G_{kl} $G_{\mu\nu}$**	$\frac{\partial u_{kl}}{\partial x}$	$\frac{\partial u_{\mu\nu}}{\partial x}$	$\sum \frac{\partial u_{kl}}{\partial x}\frac{\partial u_{\mu\nu}}{\partial x}$
1. $\mu = k$, $\nu = l$		1 1 2 2 3 3 4 4 5 5 6 6	0 $-1/h$ $-1/h$ 0 $1/h$ $1/h$	0 $-1/h$ $-1/h$ 0 $1/h$ $1/h$	$\frac{4}{h^2}$
2. $\mu = k$, $\nu = l-1$		1 5 2 4	0 $-1/h$	$1/h$ 0	0
3. $\mu = k+1$, $\nu = l$		2 6 3 5	$-1/h$ $-1/h$	$1/h$ $1/h$	$-\frac{2}{h^2}$
4. $\mu = k+1$, $\nu = l+1$		3 1 4 6	$-1/h$ 0	0 $1/h$	0
5. $\mu = k$, $\nu = l+1$		4 2 5 1	0 $-1/h$	$1/h$ 0	0
6. $\mu = k-1$, $\nu = l$		5 3 6 2	$1/h$ $1/h$	$-1/h$ $-1/h$	$-\frac{2}{h^2}$
7. $\mu = k-1$, $\nu = l-1$		6 4 1 3	$1/h$ 0	0 $-1/h$	0

The integration is always performed over a triangle with an area $h^2/2$, so for the partial derivatives with respect to x:

$$\frac{1}{h^2}(4\alpha_{kl} - 2\alpha_{k+1,l} - 2\alpha_{k-1,l})\frac{h^2}{2} \tag{19.156a}$$

is obtained. Analogously, for the partial derivatives with respect to y the corresponding term is

$$\frac{1}{h^2}(4\alpha_{kl} - 2\alpha_{k,l+1} - 2\alpha_{k,l-1})\frac{h^2}{2}. \tag{19.156b}$$

The calculation of the right-hand side $b(u_{kl})$ of (19.154) gives:

$$b(u_{kl}) = \iint\limits_{G_{kl}} f(x,y)u_{kl}(x,y)\,dx\,dy \approx f_{kl}V_P, \tag{19.157a}$$

where V_P is the volume of the pyramid over G_{kl} with height 1, determined by $u_{kl}(x,y)$ **(Fig. 19.9)**. Since

$$V_P = \frac{1}{3}\cdot 6\cdot\frac{1}{2}h^2\,, \quad \text{the approximation is} \quad b(u_{kl}) \approx f_{kl}h^2. \tag{19.157b}$$

So, the variational equations (19.154) result in the linear system of equations

$$4\alpha_{kl} - \alpha_{k+1,l} - \alpha_{k-1,l} - \alpha_{k,l+1} - \alpha_{k,l-1} = h^2 f_{kl} \quad (k,l = 1,2,\ldots,N-1) \tag{19.158}$$

for the determination of the solution coefficients.

Remarks:

1. If the solution coefficients are determined by (19.158), then $\tilde{u}(x,y)$ from (19.148) represents an explicit approximating solution, whose values can be calculated for an arbitrary point (x,y) from G.

2. If the integration domain must be covered by an irregular triangular grid then it is useful to introduce *triangular coordinates* (also called *barycentric coordinates*). In this way, the position of a point can be easily determined with respect to the triangular grid, and the calculation of the multidimensional integral is made easier as in (19.155), because every triangle can be easily transformed into the unit triangle with vertices $(0,0)$, $(0,1)$, $(1,0)$.

3. If accuracy must be improved or also the differentiability of the solution is required, then piecewise quadratic or cubic functions must be applied to obtain the supposed approximation (see, e.g., [19.22]).

4. In practical applications, usually systems of huge dimensions are obtained. This is the reason why so many special methods have been developed, e.g., for automatic triangularization and for practical enumeration of the elements (the structure of the system of equations depends on it). For detailed discussion of FEM see [19.13], [19.7], [19.22].

19.6 Approximation, Computation of Adjustment, Harmonic Analysis

19.6.1 Polynomial Interpolation

The basic problem of interpolation is to fit a curve through a sequence of points (x_ν, y_ν) $(\nu = 0,1,\ldots,n)$. This can happen graphically by any curve-fitting gadget, or numerically by a function $g(x)$, which takes given values y_ν at the points x_ν, at the so-called *interpolation points*. That is $g(x)$ satisfies the *interpolation conditions*

$$g(x_\nu) = y_\nu \quad (\nu = 0,1,2,\ldots,n). \tag{19.159}$$

In the first place, polynomials are used as interpolation functions, or for periodic functions so-called trigonometric polynomials. In this last case one talks about *trigonometric interpolation* (see 19.6.4.1, **2.**, p. 992). There are $n+1$ interpolation points, the order of the interpolation is n, and the highest degree of the interpolation polynomial is at most n. Since with increasing degree of the polynomials, strong oscillation may occur, which is usually not required, the interpolation interval can be decomposed into subintervals and a *spline interpolation* (see 19.7, p. 996) can be performed.

19.6.1.1 Newton's Interpolation Formula

To solve the interpolation problem (19.159) a polynomial of degree n is considered in the following form:

$$\begin{aligned} g(x) &= p_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots \\ &\quad + a_n(x-x_0)(x-x_1)\ldots(x-x_{n-1}). \end{aligned} \tag{19.160}$$

This is called the *Newton interpolation formula*, and it gives an easy calculation of the coefficients a_i $(i = 0,1,\ldots,n)$, since the interpolation conditions (19.159) result in a linear system of equations with a triangular matrix.

■ For $n = 2$ one gets the annexed system of equations from (19.159). The interpolation polynomial $p_n(x)$ is uniquely determined by the interpolation conditions (19.159).

$$\begin{aligned} p_2(x_0) &= a_0 &&= y_0\\ p_2(x_1) &= a_0 + a_1(x_1 - x_0) &&= y_1\\ p_2(x_2) &= a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) &&= y_2 \end{aligned}$$

The calculation of the function values can be simplified by the Horner schema (see 19.1.2.1, p. 952).

19.6.1.2 Lagrange's Interpolation Formula

A polynomial of n-th degree can be fitted through $n + 1$ points (x_ν, y_ν) $(\nu = 0, 1, \ldots, n)$, with the Lagrange formula:

$$g(x) = p_n(x) = \sum_{\mu=0}^{n} y_\mu L_\mu(x). \tag{19.161}$$

Here $L_\mu(x)$ $(\mu = 0, 1, \ldots, n)$ are the Lagrange interpolation polynomials. Equation (19.161) satisfies the interpolation conditions (19.159), since

$$L_\mu(x_\nu) = \delta_{\mu\nu} = \begin{cases} 1 & \text{for } \mu = \nu,\\ 0 & \text{for } \mu \neq \nu. \end{cases} \tag{19.162}$$

Here $\delta_{\mu\nu}$ is the Kronecker symbol. The Lagrange interpolation polynomials are defined by the formula

$$L_\mu = \frac{(x - x_0)(x - x_1)\cdots(x - x_{\mu-1})(x - x_{\mu+1})\cdots(x - x_n)}{(x_\mu - x_0)(x_\mu - x_1)\cdots(x_\mu - x_{\mu-1})(x_\mu - x_{\mu+1})\cdots(x_\mu - x_n)} = \prod_{\substack{\nu=0\\ \nu\neq\mu}}^{n} \frac{x - x_\nu}{x_\mu - x_\nu}. \tag{19.163}$$

■ A polynomial is fitted through the points given by the table

x	0	1	3
y	1	3	2

.

The Lagrange interpolation formula (19.161) is used:

$$\begin{aligned} L_0(x) &= \frac{(x-1)(x-3)}{(0-1)(0-3)} = \frac{1}{3}(x-1)(x-3),\\ L_1(x) &= \frac{(x-0)(x-3)}{(1-0)(1-3)} = -\frac{1}{2}x(x-3),\\ L_2(x) &= \frac{(x-0)(x-1)}{(3-0)(3-1)} = \frac{1}{6}x(x-1); \end{aligned}$$

$$p_2(x) = 1 \cdot L_0(x) + 3 \cdot L_1(x) + 2 \cdot L_2(x) = -\frac{5}{6}x^2 + \frac{17}{6}x + 1.$$

The Lagrange interpolation formula depends explicitly and linearly on the given values y_μ of the function. This is its theoretical importance (see, e.g., the rule of Adams-Bashforth, 19.4.1.3, **3.**, p. 971). For practical calculation the Lagrange interpolation formula is rarely reasonable.

19.6.1.3 Aitken-Neville Interpolation

In several practical cases, the explicit form of the polynomial $p_n(x)$ is not needed, but only its value at a given location x of the interpolation domain. These function values can be obtained in a recursive way developed by Aitken and Neville. The useful notation

$$p_n(x) = p_{0,1,\ldots,n}(x), \tag{19.164}$$

is applied in which the interpolation points $x_0, x_1, \ldots, x_n$ and the degree n of the polynomial are denoted. Notice that

$$p_{0,1,\ldots,n}(x) = \frac{(x - x_0)p_{1,2,\ldots,n}(x) - (x - x_n)p_{0,1,2,\ldots,n-1}(x)}{x_n - x_0}, \tag{19.165}$$

i.e., the function value $p_{0,1,\ldots,n}(x)$ can be obtained by linear interpolation of the function values of $p_{1,2,\ldots,n}(x)$ and $p_{0,1,2,\ldots,n-1}(x)$, two interpolation polynomials of degree $\le n-1$. Application of (19.165) leads to a scheme which is given here for the case of $n = 4$:

$$\begin{array}{c|l} x_0 & y_0 = p_0 \\ x_1 & y_1 = p_1\ p_{01} \\ x_2 & y_2 = p_2\ p_{12}\ p_{012} \\ x_3 & y_3 = p_3\ p_{23}\ p_{123}\ p_{0123} \\ x_4 & y_4 = p_4\ p_{34}\ p_{234}\ p_{1234}\ \underline{p_{01234} = p_4(x).} \end{array} \tag{19.166}$$

The elements of (19.166) are calculated column-wise. A new value in the scheme is obtained from its west and north-west neighbors

$$p_{23} = \frac{(x-x_2)p_3 - (x-x_3)p_2}{x_3 - x_2} = p_3 + \frac{x - x_3}{x_3 - x_2}(p_3 - p_2), \tag{19.167a}$$

$$p_{123} = \frac{(x-x_1)p_{23} - (x-x_3)p_{12}}{x_3 - x_1} = p_{23} + \frac{x - x_3}{x_3 - x_1}(p_{23} - p_{12}), \tag{19.167b}$$

$$p_{1234} = \frac{(x-x_1)p_{234} - (x-x_4)p_{123}}{x_4 - x_1} = p_{234} + \frac{x - x_4}{x_4 - x_1}(p_{234} - p_{123}). \tag{19.167c}$$

For performing the *Aitken-Neville algorithm* on a computer only a vector $\underline{\mathbf{p}}$ with $n+1$ components (see [19.4]) is introduced, which takes the values of the columns in (19.166) after each other according to the rule that the value $p_{i-k,i-k+1,\ldots,i}$ $(i = k, k+1, \ldots, n)$ of the k-th column will be the i-th component p_i of $\underline{\mathbf{p}}$. The columns of (19.166) must be calculated from down to the top, so p contains all necessary values. The algorithm has the following two steps:

1. For $i = 0, 1, \ldots, n$ set $p_i = y_i$. (19.168a)

2. For $k = 1, 2, \ldots, n$ and for $i = n, n-1, \ldots, k$ compute $p_i = p_i + \dfrac{x - x_i}{x_i - x_{i-k}}(p_i - p_{i-1})$. (19.168b)

After finishing (19.168b) we have the required function value $p_n(x)$ at x in element p_n.

19.6.2 Approximation in Mean

The principle of approximation in mean is known as the *Gauss least squares method*. In calculations continuous and discrete cases are distinguished.

19.6.2.1 Continuous Problems, Normal Equations

The function $f(x)$ is approximated by a function $g(x)$ on the interval $[a, b]$ so that the expression

$$F = \int_a^b \omega(x)[f(x) - g(x)]^2\,dx, \tag{19.169}$$

depending on the parameters contained by $g(x)$, should be minimal. $\omega(x)$ denotes a given weight function, such that $\omega(x) > 0$ in the integration interval.
If the best approximation $g(x)$ is supposed to have the form

$$g(x) = \sum_{i=0}^{n} a_i g_i(x) \tag{19.170}$$

with suitable linearly independent functions $g_0(x)$, $g_1(x)$, $\ldots$, $g_n(x)$, then the necessary conditions

$$\frac{\partial F}{\partial a_i} = 0 \quad (i = 0, 1, \ldots, n) \tag{19.171}$$

for an extreme value of (19.169) result in the so-called *normal system of equations*

$$\sum_{i=0}^{n} a_i(g_i,\, g_k) = (f, g_k) \quad (k = 0, 1, \ldots, n) \tag{19.172}$$

to determine the unknown coefficients a_i. Here the brief notations

$$(g_i, g_k) = \int_a^b \omega(x) g_i(x) g_k(x)\, dx, \tag{19.173a}$$

$$(f, g_k) = \int_a^b \omega(x) f(x) g_k(x)\, dx \quad (i, k = 0, 1, \ldots, n) \tag{19.173b}$$

are used, which are considered as the *scalar products* of the two indicated functions.
The system of normal equations can be solved uniquely, since the functions $g_0(x)$, $g_1(x)$, $\ldots$, $g_n(x)$ are linearly independent. The coefficient matrix of the system (19.172) is symmetric, so the Cholesky method (see 19.2.1.2, p. 958) can be applied. The coefficients a_i can be determined directly, without solving the system of equations, if the system of functions $g_i(x)$ is *orthogonal*, that is, if

$$(g_i, g_k) = 0 \ \text{ for } \ i \neq k. \tag{19.174}$$

It is called an *orthonormal* system, if

$$(g_i, g_k) = \begin{cases} 0 & \text{for } i \neq k, \\ 1 & \text{for } i = k \end{cases} \quad (i, k = 0, 1, \ldots, n). \tag{19.175}$$

With (19.175), the normal equations (19.172) are reduced to

$$a_i = (f, g_i) \quad (i = 0, 1, \ldots, n). \tag{19.176}$$

Linearly independent function systems can be orthogonalized. From the power functions $g_i(x) = x^i$ $(i = 0, 1, \ldots, n)$, depending on the weight function and on the interval, the *orthogonal polynomials* in **Table 19.2** can be obtained.

Table 19.2 Orthogonal polynomials

$[a, b]$	$\omega(x)$	Name of the polynomials	see p.
$[-1, 1]$	1	Legendre polynomial $P_n(x)$	566
$[-1, 1]$	$\dfrac{1}{\sqrt{1-x^2}}$	Chebysev polynomial $T_n(x)$	989
$[0, \infty)$	e^{-x}	Laguerre polynomial $L_n(x)$	568
$(-\infty, \infty)$	$e^{-x^2/2}$	Hermite polynomial $H_n(x)$	568

(19.177)

These polynomial systems can be used on arbitrary intervals:
1. Finite approximation interval.
2. Approximation interval infinite at one end, e.g., in time-dependent problems.
3. Approximation interval infinite at both ends, e.g., in stream problems.
Every finite interval $[a, b]$ can be transformed by the substitution

$$x = \frac{b+a}{2} + \frac{b-a}{2} t \quad (x \in [a, b],\ t \in [-1, 1]) \tag{19.178}$$

into the interval $[-1, 1]$.

19.6.2.2 Discrete Problems, Normal Equations, Householder's Method

Let N pairs of values (x_ν, y_ν) be given, e.g., by measured values. A function $g(x)$ has to be determined, whose values $g(x_\nu)$ differ from the given values y_ν in such a way that the quadratic expression

$$F = \sum_{\nu=1}^{N} [y_\nu - g(x_\nu)]^2 \tag{19.179}$$

is minimal. The value of F depends on the parameters contained in the function $g(x)$. Formula (19.179) represents the classical *sum of residual squares*. The minimization of the sum of residual squares is called the *least squares method*. From the assumption (19.170) and the necessary conditions $\frac{\partial F}{\partial a_i} = 0$ $(i = 0, 1, \ldots, n)$ for a relative minimum of (19.179) for the coefficients the *normal system of equations* is obtained:

$$\sum_{i=0}^{n} a_i[g_i g_k] = [y g_k] \quad (k = 0, 1, \ldots, n). \tag{19.180}$$

Here the Gaussean sum symbols are used in the following notation:

$$[g_i g_k] = \sum_{\nu=1}^{N} g_i(x_\nu) g_k(x_\nu), \tag{19.181a}$$

$$[y g_k] = \sum_{\nu=1}^{N} y_\nu g_k(x_\nu) \quad (i, k = 0, 1, \ldots, n). \tag{19.181b}$$

Usually, $n \ll N$.

■ For the polynomial $g(x) = a_0 + a_1 x + \cdots + a_n x^n$, the normal equations are $a_0[x^k] + a_1[x^{k+1}] + \cdots + a_n[x^{k+n}] = [x^k y]$ $(k = 0, 1, \ldots, n)$ with $[x^k] = \sum_{\nu=1}^{N} {x_\nu}^k$, $[x^0] = N$, $[x^k y] = \sum_{\nu=1}^{N} {x_\nu}^k y_\nu$, $[y] = \sum_{\nu=1}^{N} y_\nu$. The coefficient matrix of the normal system of equations (19.180) is symmetric, so for the numerical solution the Cholesky method can be applied.

The normal equations (19.180) and the residue sum square (19.179) have the following compact form:

$$\mathbf{G}^{\mathrm{T}}\mathbf{G}\underline{\mathbf{a}} = \mathbf{G}^{\mathrm{T}}\underline{\mathbf{y}}, \quad F = (\underline{\mathbf{y}} - \mathbf{G}\underline{\mathbf{a}})^{\mathrm{T}}(\underline{\mathbf{y}} - \mathbf{G}\underline{\mathbf{a}}) \quad \text{with} \tag{19.182a}$$

$$\mathbf{G} = \begin{pmatrix} g_0(x_1) & g_1(x_1) & g_2(x_1) & \ldots & g_n(x_1) \\ g_0(x_2) & g_1(x_2) & g_2(x_2) & \ldots & g_n(x_2) \\ g_0(x_3) & g_1(x_3) & g_2(x_3) & \ldots & g_n(x_3) \\ \vdots & & & & \\ g_0(x_N) & g_1(x_N) & g_2(x_N) & \ldots & g_n(x_N) \end{pmatrix}, \quad \underline{\mathbf{y}} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{pmatrix}, \quad \underline{\mathbf{a}} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}. \tag{19.182b}$$

If, instead of the minimalization of the sum of residual squares, one wants to solve the interpolation problem for the N points (x_ν, y_ν), then the following system of equations should be solved:

$$\mathbf{G}\underline{\mathbf{a}} = \underline{\mathbf{y}}. \tag{19.183}$$

This system of equations is over-determined in the case of $n < N - 1$, and usually it does not have any solution. The equations (19.180) or (19.182a) are obtained by multiplying (19.183) by $\mathbf{G}^{\mathrm{T}}$.
From a numerical viewpoint, the Householder method (see 4.5.3.2, **2.**, p. 314) is recommended to solve equation (19.183), and this solution results in the minimal sum of residual squares (19.179).

19.6.2.3 Multidimensional Problems

1. Computation of Adjustments

Suppose that there is a function $f(x_1, x_2, \ldots, x_n)$ of n independent variables $x_1, x_2, \ldots, x_n$. Its explicit form is not known; only N substitution values f_ν are given, which are, in general, measured values. These data can be written in a table (see (19.184)).

$$\begin{array}{c|cccc} x_1 & x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(N)} \\ x_2 & x_2^{(1)} & x_2^{(2)} & \ldots & x_2^{(N)} \\ \vdots & \vdots & \vdots & & \vdots \\ x_n & x_n^{(1)} & x_n^{(2)} & \ldots & x_n^{(N)} \\ \hline f & f_1 & f_2 & \ldots & f_N \end{array} \tag{19.184}$$

The formulation of the adjustment problem is clearer by introducing the following vectors:

$\underline{\mathbf{x}} = (x_1, x_2, \dots, x_n)^{\mathrm{T}}$	:	Vector of n independent variables,
$\underline{\mathbf{x}}^{(\nu)} = (x_1^{(\nu)}, x_2^{(\nu)}, \dots, x_n^{(\nu)})^{\mathrm{T}}$	:	Vector of the ν-th interpolation node $(\nu = 1, \dots, N)$,
$\underline{\mathbf{f}} = (f_1, f_2, \dots f_N)^{\mathrm{T}}$	:	Vector of the N function values at the N interpolation nodes.

$f(x_1, x_2, \dots, x_n) = f(\underline{\mathbf{x}})$ is approximated by a function of the form

$$g(x_1, x_2, \dots, x_n) = \sum_{i=0}^{m} a_i g_i(x_1, x_2, \dots, x_n). \tag{19.185}$$

Here, the $m+1$ functions $g_i(x_1, x_2, \dots, x_n) = g_i(\underline{\mathbf{x}})$ are suitable, selected functions.

■ **A:** Linear approximation by n variables: $g(x_1, x_2, \dots, x_n) = a_0 + a_1x_1 + a_2x_2 + \cdots + a_nx_n$.

■ **B:** Complete quadratic approximation with three variables:
$g(x_1, x_2, x_3) = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4{x_1}^2 + a_5{x_2}^2 + a_6{x_3}^2 + a_7x_1x_2 + a_8x_1x_3 + a_9x_2x_3$.

The coefficients are chosen to minimize $\sum_{\nu=1}^{N} \left[f_\nu - g\left(x_1^{(\nu)}, x_2^{(\nu)}, \dots, x_n^{(\nu)}\right)\right]^2$.

2. Normal System of Equations

Analogously to (19.182b) the matrix $\mathbf{G}$ is formed in which the interpolation nodes x_ν are replaced by vectorial interpolation nodes $\underline{\mathbf{x}}^{(\nu)}$ $(\nu = 1, 2, \dots, N)$. To determine the coefficients, the normal system of equations

$$\mathbf{G}^{\mathrm{T}}\mathbf{G}\underline{\mathbf{a}} = \mathbf{G}^{\mathrm{T}}\underline{\mathbf{f}} \tag{19.186}$$

can be used or the over-determined system of equations

$$\mathbf{G}\underline{\mathbf{a}} = \underline{\mathbf{f}}. \tag{19.187}$$

■ For an example of multidimensional regression see 16.3.4.3, **3.**, p. 842.

19.6.2.4 Non-Linear Least Squares Problems

The main idea is discussed for a one-dimensional discrete case. The approximation function $g(x)$ depends non-linearly on certain parameters.

■ **A:** $g(x) = a_0e^{a_1x} + a_2e^{a_3x}$. This expression does not depend linearly on the parameters a_1 and a_3.

■ **B:** $g(x) = a_0e^{a_1x}\cos a_2x$. This function does not depend linearly on the parameters a_1 and a_2.

The fact that the approximation function $g(x)$ depends on a parameter vector $\underline{\mathbf{a}} = (a_0, a_1, \dots, a_n)^{\mathrm{T}}$ is indicated by the notation

$$g = g(x, \underline{\mathbf{a}}) = g(x;\, a_0, a_1, \dots, a_n). \tag{19.188}$$

Suppose, N pairs of values (x_ν, y_ν) $(\nu = 1, 2, \dots, N)$ are given. To minimize the sum of residual squares

$$\sum_{\nu=1}^{N} [y_\nu - g(x_\nu;\, a_0, a_1, \dots, a_n)]^2 = F(a_0, a_1, \dots, a_n) \tag{19.189}$$

the necessary conditions $\dfrac{\partial F}{\partial a_i} = 0$ $(i = 0, 1, \dots, n)$ lead to a non-linear normal equation system which must be solved by an iterative method, e.g., by the Newton method (see 19.2.2.2, p. 962).

Another way to solve the problem, which is usually used in practical problems, is the application of the Gauss-Newton method (see 19.2.2.3, p. 962) given for the solution of the non-linear least squares problem (19.24). The following steps are needed to apply it for this non-linear approximation problem (19.189):

1. Linearization of the approximating function $g(x, \underline{\mathbf{a}})$ with the help of the Taylor formula with respect to a_i. To do this, the approximation values $a_i^{(0)}$ $(i = 0, 1, \dots, n)$ are needed:

$$g(x, \underline{\mathbf{a}}) \approx \tilde{g}(x, \underline{\mathbf{a}}) = g(x, \underline{\mathbf{a}}^{(0)}) + \sum_{i=0}^{n} \frac{\partial g}{\partial a_i}(x, \underline{\mathbf{a}}^{(0)})(a_i - a_i^{(0)}). \tag{19.190}$$

2. Solution of the linear minimum problem

$$\sum_{\nu=1}^{N}[y_\nu - \tilde{g}(x_\nu, \underline{\mathbf{a}})]^2 = \min! \tag{19.191}$$

with the help of the normal equation system

$$\tilde{\mathbf{G}}^{\mathrm{T}}\tilde{\mathbf{G}}\underline{\mathbf{\Delta a}} = \tilde{\mathbf{G}}^{\mathrm{T}}\underline{\mathbf{\Delta y}} \tag{19.192}$$

or by the Householder method. In (19.192) the components of the vectors $\underline{\mathbf{\Delta a}}$ and $\underline{\mathbf{\Delta y}}$ are given as

$$\Delta a_i = a_i - a_i^{(0)} \qquad (i = 0, 1, 2, \dots, n) \text{ and} \tag{19.193a}$$

$$\Delta y_\nu = y_\nu - g(x_\nu, \underline{\mathbf{a}}^{(0)}) \quad (\nu = 1, 2, \dots, N). \tag{19.193b}$$

The matrix $\tilde{G}$ can be determined analogously to G in (19.182b), where $g_i(x_\nu)$ are replaced by $\dfrac{\partial g}{\partial a_i}(x_\nu, \underline{\mathbf{a}}_1^{(0)})$ $(i = 0, 1, \dots, n;\ \nu = 1, 2, \dots, N)$.

3. Calculation of a new approximation

$$a_i^{(1)} = a_i^{(0)} + \Delta a_i \text{ or } a_i^{(1)} = a_i^{(0)} + \gamma\Delta a_i \quad (i = 0, 1, 2, \dots, n), \tag{19.194}$$

where $\gamma > 0$ is a step length parameter.

By repeating steps **2** and **3** with $a_i^{(1)}$ instead of $a_i^{(0)}$, etc. a sequence of approximation values is obtained for the required parameters, whose convergence strongly depends on the accuracy of the initial approximations. The value of the sum of residual squares can be reduced with the introduction of the multiplier γ .

19.6.3 Chebyshev Approximation

19.6.3.1 Problem Definition and the Alternating Point Theorem

1. Principle of Chebyshev Approximation

Chebyshev approximation or *uniform approximation* in the continuous case is the following: The function $f(x)$ has to be approximated in an interval $a \le x \le b$ by the approximation function $g(x) = g(x;\, a_0, a_1, \dots, a_n)$ so that the error defined by

$$\max_{a\le x\le b} |f(x) - g(x;\, a_0, a_1, \dots, a_n)| = \Phi(a_0, a_1, \dots, a_n) \tag{19.195}$$

should be as small as possible for the appropriate choice of the unknown parameters a_i $(i = 0, 1, \dots, n)$. If there exists such an approximating function for $f(x)$, then the maximum of the absolute error value will be taken at least at $n+2$ points x_ν of the interval, at the so-called *alternating points*, with changing signs (**Fig. 19.10**). This is actually the meaning of the *alternating point theorem* for the characterization of the solution of a Chebyshev approximation problem.

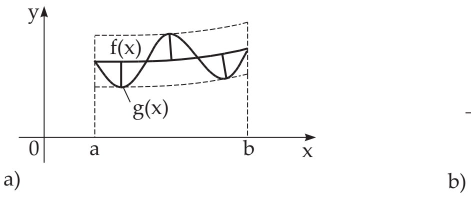


Figure 19.10

■ If the function $f(x) = x^n$ is approximated on the interval $[-1, 1]$ by a polynomial of degree $\le n - 1$ in the Chebyshev sense, then the *Chebyshev polynomial* $T_n(x)$ is obtained as an error function whose maximum is normed to one. The alternating points, being at the endpoints and at exactly $n-1$ points in the interior of the interval, correspond to the extreme points of $T_n(x)$ (**Fig. 19.11a–f**).

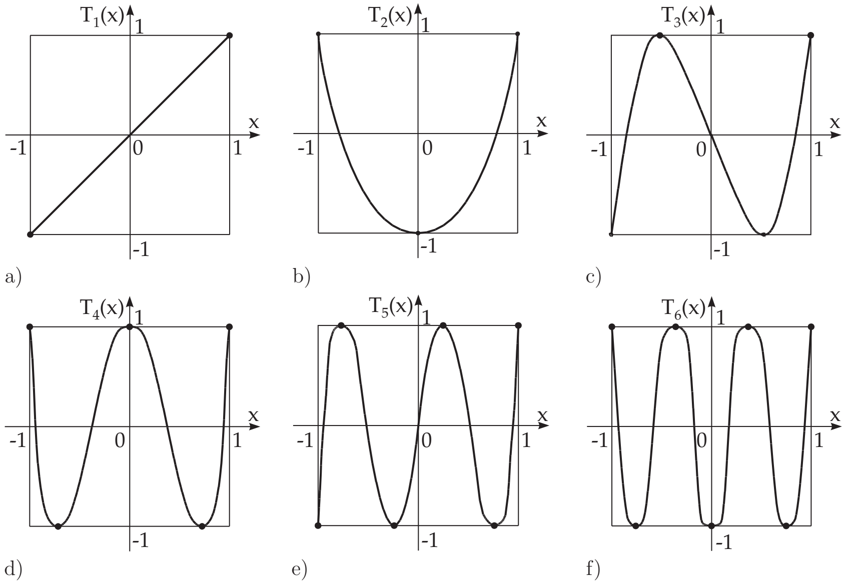


Figure 19.11

19.6.3.2 Properties of the Chebyshev Polynomials

1. Representation

$$T_n(x) = \cos(n \arccos x), \tag{19.196a}$$

$$T_n(x) = \frac{1}{2}\left[\left(x+\sqrt{x^2-1}\right)^n + \left(x-\sqrt{x^2-1}\right)^n\right], \tag{19.196b}$$

$$T_n(x) = \begin{cases} \cos nt,\ x=\cos t & \text{for } |x|<1, \\ \cosh nt,\ x=\cosh t & \text{for } |x|>1 \end{cases} \quad (n=1,2,\ldots). \tag{19.196c}$$

2. Roots of $T_n(x)$

$$x_\mu = \cos\frac{(2\mu-1)\pi}{2n} \quad (\mu=1,2,\ldots,n). \tag{19.197}$$

3. Position of the extreme values of $T_n(x)$ for $x \in [-1,1]$

$$x_\nu = \cos\frac{\nu\pi}{n} \quad (\nu=0,1,2,\ldots,n). \tag{19.198}$$

4. Recursion Formula

$$T_{n+1} = 2xT_n(x) - T_{n-1}(x) \quad (n=1,2,\ldots;\ T_0(x)=1,\ T_1(x)=x). \tag{19.199}$$

This recursion results in e.g.

$$T_2(x) = 2x^2-1, \quad T_3(x) = 4x^3-3x, \tag{19.200a}$$

$$T_4(x) = 8x^4 - 8x^2 + 1, \quad T_5(x) = 16x^5 - 20x^3 + 5x, \tag{19.200b}$$
$$T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1, \tag{19.200c}$$
$$T_7(x) = 64x^7 - 112x^5 + 56x^3 - 7x, \tag{19.200d}$$
$$T_8(x) = 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1, \tag{19.200e}$$
$$T_9(x) = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x, \tag{19.200f}$$
$$T_{10}(x) = 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1. \tag{19.200g}$$

19.6.3.3 Remes Algorithm

1. Consequences of the Alternating Point Theorem

The numerical solution of the continuous Chebyshev approximation problem originates from the alternating point theorem. The approximating function is chosen as

$$g(x) = \sum_{i=0}^{n} a_i g_i(x) \tag{19.201}$$

with $n+1$ linearly independent known functions, and the coefficients of the solution of the Chebyshev problem are denoted by ${a_i}^*$ $(i = 0, 1, \dots, n)$ and the minimal deviation according to (19.195) by $\varrho = \Phi({a_0}^*, {a_1}^*, \dots, {a_n}^*)$. In the case when the functions f and g_i $(i = 0, 1, \dots, n)$ are differentiable, from the alternating point theorem one has

$$\sum_{i=0}^{n} {a_i}^* g_i(x_\nu) + (-1)^\nu \varrho = f(x_\nu), \quad \sum_{i=0}^{n} {a_i}^* g_i'(x_\nu) = f'(x_\nu) \quad (\nu = 1, 2, \dots, n+2). \tag{19.202}$$

The nodes x_ν are the alternating points with

$$a \le x_1 < x_2 < \dots < x_{n+2} \le b. \tag{19.203}$$

The equations (19.202) give $2n + 4$ conditions for the $2n + 4$ unknown quantities of the Chebyshev approximation problem: $n + 1$ coefficients, $n + 2$ alternating points and the minimal deviation ϱ. If the endpoints of the interval belong to the alternating points, then the conditions for the derivatives are not necessarily valid there.

2. Determination of the Minimal Solution according to Remes

According to Remes, one proceeds with the numerical determination of the minimal solution as follows:

1. An approximation of the alternating points ${x_\nu}^{(0)}$ $(\nu = 1, 2, \dots, n+2)$ are determined according to (19.203), e.g., equidistant or as the positions of the extrema of $T_{n+1}(x)$ (see 19.6.3.2, p. 988).

2. The linear system of equations

$$\sum_{i=0}^{n} a_i g_i({x_\nu}^{(0)}) + (-1)^\nu \varrho = f({x_\nu}^{(0)}) \quad (\nu = 1, 2, \dots, n+2)$$

is solved and the solutions are the approximations ${a_i}^{(0)}$ $(i = 0, 1, \dots, n)$ and ϱ_0.

3. A new approximation of the alternating points ${x_\nu}^{(1)}$ $(\nu = 1, 2, \dots, n+2)$ is determined, e.g., as positions of the extrema of the error function $f(x) - \sum\limits_{i=0}^{n} {a_i}^{(0)} g_i(x)$. Now, it is sufficient to apply only approximations of these points.

By repeating steps **2** and **3** with ${x_\nu}^{(1)}$ and ${a_i}^{(1)}$ instead of ${x_\nu}^{(0)}$ and ${a_i}^{(0)}$, etc. a sequence of approximations is obtained for the coefficients and the alternating points, whose convergence is guaranteed under certain conditions, which can be given (see [19.25]). The calculations are stopped if, e.g., from a certain iteration index μ

$$|\varrho_\mu| = \max_{a \le x \le b} \left| f(x) - \sum_{i=0}^{n} {a_i}^{(\mu)} g_i(x) \right| \tag{19.204}$$

holds with a sufficient accuracy.

19.6.3.4 Discrete Chebyshev Approximation and Optimization

From the continuous Chebyshev approximation problem

$$\max_{a\le x\le b}\left|f(x)-\sum_{i=0}^{n}a_i g_i(x)\right| = \min! \tag{19.205}$$

the corresponding discrete problem can be got, if requiring N nodes x_ν $(\nu = 1,2,\ldots,N;\, N\ge n+2)$ are chosen with the property $a \le x_1 < x_2 \cdots < x_N \le b$ and requiring

$$\max_{\nu=1,2,\ldots,N}\left|f(x_\nu)-\sum_{i=0}^{n}a_i g_i(x_\nu)\right| = \min!\,. \tag{19.206}$$

The substitution

$$\gamma = \max_{\nu=1,2,\ldots,N}\left|f(x_\nu)-\sum_{i=0}^{n}a_i g_i(x_\nu)\right|, \tag{19.207}$$

has obviously the consequence

$$\left|f(x_\nu)-\sum_{i=0}^{n}a_i g_i(x_\nu)\right| \le \gamma \quad (\nu = 1,2,\ldots,N). \tag{19.208}$$

Eliminating the absolute values from (19.208) a linear system of inequalities is obtained for the coefficients a_i and γ, so the problem (19.206) becomes a linear programming problem (see 18.1.1.1, p. 909):

$$\gamma = \min! \quad \text{subject to} \quad \begin{cases} \gamma + \sum\limits_{i=0}^{n} a_i g_i(x_\nu) \ge f(x_\nu), \\ \gamma - \sum\limits_{i=0}^{n} a_i g_i(x_\nu) \ge -f(x_\nu) \end{cases} \quad (\nu = 1,2,\ldots,N). \tag{19.209}$$

Equation (19.209) has a minimal solution with $\gamma > 0$. For a sufficiently large number N of nodes and with some further conditions the solution of the discrete problem can be considered as the solution of the continuous problem.

If instead of the linear approximation function $g(x) = \sum\limits_{i=0}^{n} a_i g_i(x)$ a non-linear approximation function $g(x) = g(x; a_0, a_1, \ldots, a_n)$ is used, which does not depend linearly on the parameters $a_0, a_1, \ldots, a_n$, then analogously a *non-linear optimization problem* is obtained. It is usually non-convex even in the cases of simple function forms. This essentially reduces the number of numerical solution methods for non-linear optimization problems (see 18.2.2.1, p. 926).

19.6.4 Harmonic Analysis

A periodic function $f(x)$ with period 2π, which is given formally or empirically, should be approximated by a *trigonometric polynomial* or a *Fourier sum* of the form

$$g(x) = \frac{a_0}{2} + \sum_{k=1}^{n}(a_k \cos kx + b_k \sin kx), \tag{19.210}$$

where the coefficients a_0, a_k and b_k are unknown real numbers. The determination of the coefficients is the topic of harmonic analysis.

19.6.4.1 Formulas for Trigonometric Interpolation

1. Formulas for the Fourier Coefficients

Since the function system 1, $\cos kx$, $\sin kx$ $(k = 1, 2, \ldots, n)$ is orthogonal in the interval $[0, 2\pi]$ with respect to the weight function $\omega \equiv 1$, the formulas for the coefficients are obtained as

$$a_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos kx\,dx, \quad b_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin kx\,dx \quad (k = 0, 1, 2, \ldots, n) \tag{19.211}$$

by applying the continuous least squares method according to (19.172). The coefficients a_k and b_k calculated by formulas (19.211) are called *Fourier coefficients* of the periodic function $f(x)$ (see 7.4, p. 474).

If the integrals in (19.211) are complicated or the function $f(x)$ is known only at discrete points, then the Fourier coefficients can be determined only approximately by numerical integration.

Using the trapezoidal formula (see 19.3.2.2, p. 964) with $N + 1$ equidistant nodes

$$x_\nu = \nu h \quad (\nu = 0, 1, \ldots, N), \quad h = \frac{2\pi}{N} \tag{19.212}$$

the approximation formula

$$a_k \approx \tilde{a}_k = \frac{2}{N}\sum_{\nu=1}^{N} f(x_\nu)\cos kx_\nu, \quad b_k \approx \tilde{b}_k = \frac{2}{N}\sum_{\nu=1}^{N} f(x_\nu)\sin kx_\nu \quad (k = 0, 1, 2, \ldots, n) \tag{19.213}$$

is obtained. The trapezoidal formula becomes the very simple rectangular formula in the case of periodic functions. It has higher accuracy here as a consequence of the following fact: If $f(x)$ is periodic and $(2m + 2)$ times differentiable, then the trapezoidal formula has an error of order $O(h^{2m+2})$.

2. Trigonometric Interpolation

Some special trigonometric polynomials formed with the approximation coefficients $\tilde{a}_k$ and $\tilde{b}_k$ have important properties. Two of them are mentioned here:

1. Interpolation Suppose $N = 2n$ holds. The special trigonometric polynomial

$$\tilde{g}_1(x) = \frac{1}{2}\tilde{a}_0 + \sum_{k=1}^{n-1}(\tilde{a}_k \cos kx + \tilde{b}_k \sin kx) + \frac{1}{2}\tilde{a}_n \cos nx \tag{19.214}$$

with coefficients (19.213) satisfies the interpolation conditions

$$\tilde{g}_1(x_\nu) = f(x_\nu) \qquad (\nu = 1, 2, \ldots, N) \tag{19.215}$$

at the interpolation nodes x_ν (19.212). Because of the perodicity of $f(x)$ $f(x_0) = f(x_N)$ holds.

2. Approximation in Mean Suppose $N = 2n$. The special trigonometric polynomial

$$\tilde{g}_2(x) = \frac{1}{2}\tilde{a}_0 + \sum_{k=1}^{m}(\tilde{a}_k \cos kx + \tilde{b}_k \sin kx) \tag{19.216}$$

for $m < n$ and with the coefficients (19.213) approximates the function $f(x)$ in discrete quadratic mean with respect to the N nodes x_ν (19.212), that is, the residual sum of squares

$$F = \sum_{\nu=1}^{N} [f(x_\nu) - \tilde{g}_2(x_\nu)]^2 \qquad (19.217)$$

is minimal. The formulas (19.213) are the originating point for the different ways of effective calculation of Fourier coefficients.

19.6.4.2 Fast Fourier Transformation (FFT)

1. Computation costs of computing Fourier coefficients

The sums in the formulas (19.213) also occur in connection with discrete Fourier transformation, e.g., in electrotechnics, in impulse and picture processing. Here N can be very large, so the occurring sums must be calculated in a rational way, since the calculation of the N approximating values (19.213) of the Fourier coefficients requires about N^2 additions and multiplications.
For the special case of $N = 2^p$, the number of multiplications can be largely reduced from $N^2(= 2^{2p})$ to $pN(= p2^p)$ with the help of the so-called *fast Fourier transformation* **FFT**. The magnitude of this reduction is demonstrated on the example on the right-hand side.

p	N^2	pN
10	$\sim 10^6$	$\sim 10^4$
20	$\sim 10^{12}$	$\sim 10^7$

(19.218)

By this method, the computation costs and computation time are reduced so effectively that in some important application fields even a smaller computer is sufficient.
The FFT uses the properties of the N-th unit roots, i.e., the solutions of equation $z^N = 1$ to a successive sum up in (19.213).

2. Complex Representation of the Fourier Sum

The principle of FFT can be described fairly easily if the Fourier sum (19.210) is rewritten with the formulas

$$\cos kx = \frac{1}{2}\left(e^{\mathrm{i}kx} + e^{-\mathrm{i}kx}\right), \quad \sin kx = \frac{\mathrm{i}}{2}\left(e^{-\mathrm{i}kx} - e^{\mathrm{i}kx}\right) \qquad (19.219)$$

into the complex form

$$g(x) = \frac{1}{2}a_0 + \sum_{k=1}^{n}(a_k \cos kx + b_k \sin kx) = \frac{1}{2}a_0 + \sum_{k=1}^{n}\left(\frac{a_k - \mathrm{i}b_k}{2}e^{\mathrm{i}kx} + \frac{a_k + \mathrm{i}b_k}{2}e^{-\mathrm{i}kx}\right). \qquad (19.220)$$

By substitution

$$c_k = \frac{a_k - \mathrm{i}b_k}{2}, \quad (19.221\text{a}) \quad \text{because of (19.211)} \quad c_k = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-\mathrm{i}kx}\,dx, \qquad (19.221\text{b})$$

so (19.220) becomes the *complex representation of the Fourier sum*:

$$g(x) = \sum_{k=-n}^{n} c_k e^{\mathrm{i}kx} \quad \text{with} \quad c_{-k} = \bar{c}_k. \qquad (19.222)$$

If the complex coefficients c_k are known, then the required real Fourier coefficients can be got in the following simple way:

$$a_0 = 2c_0, \quad a_k = 2\mathrm{Re}(c_k), \quad b_k = -2\mathrm{Im}(c_k) \quad (k = 1, 2, \ldots, n). \qquad (19.223)$$

3. Numerical Calculation of the Complex Fourier Coefficients

For the numerical determination of c_k the trapezoidal formula can be applied for (19.221b) analogously to (19.212) and (19.213), and the discrete complex Fourier coefficients $\tilde{c}_k$ are obtained:

$$\tilde{c}_k = \frac{1}{N}\sum_{\nu=0}^{N-1} f(x_\nu)e^{-\mathrm{i}kx_\nu} = \sum_{\nu=0}^{N-1} f_\nu \omega_N^{k\nu} \quad (k = 0, 1, 2, \ldots, n) \quad \text{with} \qquad (19.224\text{a})$$

$$f_\nu = \frac{1}{N} f(x_\nu), \quad x_\nu = \frac{2\pi\nu}{N} \quad (\nu = 0, 1, 2, \ldots, N-1), \quad \omega_N = e^{-\frac{2\pi \mathrm{i}}{N}}. \tag{19.224b}$$

Relation (19.224a) with the quantities (19.224b) is called the *discrete complex Fourier transformation* of length N of the values f_ν ($\nu = 0, 1, 2, \ldots, N-1$).
The powers $\omega_N^\nu = z$ ($\nu = 0, 1, 2, \ldots, N-1$) satisfy equation $z^N = 1$. So, they are called the N*-th unit roots*. Since $e^{-2\pi \mathrm{i}} = 1$,

$$\omega_N^N = 1, \ \omega_N^{N+1} = \omega_N^1, \ \omega_N^{N+2} = \omega_N^2, \ldots . \tag{19.225}$$

The effective calculation of the sum (19.224a) uses the fact that a discrete complex Fourier transformation of length $N = 2n$ can be reduced to two transformations with length $\frac{N}{2} = n$ in the following way:

a) For every coefficient $\tilde{c}_k$ with an even index, i.e., $k = 2l$,

$$\tilde{c}_{2l} = \sum_{\nu=0}^{2n-1} f_\nu \omega_N^{2l\nu} = \sum_{\nu=0}^{n-1} \left[f_\nu \omega_N^{2l\nu} + f_{n+\nu} \omega_N^{2l(n+\nu)} \right] = \sum_{\nu=0}^{n-1} \left[f_\nu + f_{n+\nu} \right] \omega_N^{2l\nu} \tag{19.226}$$

holds. Here the equality $\omega_N^{2l(n+\nu)} = \omega_N^{2ln} \omega_N^{2l\nu} = \omega_N^{2l\nu}$ is used.
Substituting

$$y_\nu = f_\nu + f_{n+\nu} \quad (\nu = 0, 1, 2, \ldots, n-1) \tag{19.227}$$

and considering that $\omega_N^2 = \omega_n$, the sum

$$\tilde{c}_{2l} = \sum_{\nu=0}^{n-1} y_\nu \omega_n^{l\nu} \quad (\nu = 0, 1, 2, \ldots, n-1) \tag{19.228}$$

is the discrete complex Fourier transformation of the values y_ν ($\nu = 0, 1, 2, \ldots, n-1$) with length $n = \frac{N}{2}$.

b) For every coefficient $\tilde{c}_k$ with an odd index, i.e., with $k = 2l + 1$

$$\tilde{c}_{2l+1} = \sum_{\nu=0}^{2n-1} f_\nu \omega_N^{(2l+1)\nu} = \sum_{\nu=0}^{n-1} \left[(f_\nu - f_{n+\nu}) \omega_N^\nu \right] \omega_N^{2l\nu} \tag{19.229}$$

is obtained analogously. Substituting

$$y_{n+\nu} = (f_\nu - f_{n+\nu}) \omega_N^\nu \quad (\nu = 0, 1, 2, \ldots, n-1) \tag{19.230}$$

and considering that $\omega_N^2 = \omega_n$, the sum

$$\tilde{c}_{2l+1} = \sum_{\nu=0}^{n-1} y_{n+\nu} \omega_n^{l\nu} \quad (\nu = 0, 1, 2, \ldots, n-1) \tag{19.231}$$

is the discrete complex Fourier transformation of the values $y_{n+\nu}$ ($\nu = 0, 1, 2, \ldots, n-1$) with length $n = \frac{N}{2}$.
The reduction according to **a)** and **b)**, i.e., the reduction of a discrete complex Fourier transformation to two discrete complex Fourier transformations of half the length, can be continued if N is a power of 2, i.e., if $N = 2^p$ (p is a natural number). The application of the reduction after p times is called the FFT.
Since every reduction step requires $\frac{N}{2}$ complex multiplications because of (19.230), the computation cost of the FFT method is

$$\frac{N}{2} p = \frac{N}{2} \log_2 N. \tag{19.232}$$

4. Scheme for FFT

For the special case $N = 8 = 2^3$, the three corresponding reduction steps of the FFT according to (19.227) and (19.230) are demonstrated in the following **Scheme 1**:

Scheme 1:

	Step 1	Step 2	Step 3	
f_0	$y_0 = f_0 + f_4$	$y_0 := y_0 + y_2$	$y_0 := y_0 + y_1$	$= \tilde{c}_0$
f_1	$y_1 = f_1 + f_5$	$y_1 := y_1 + y_3$	$y_1 := (y_0 - y_1)\omega_2^0$	$= \tilde{c}_4$
f_2	$y_2 = f_2 + f_6$	$y_2 := (y_0 - y_2)\omega_4^0$	$y_2 := y_2 + y_3$	$= \tilde{c}_2$
f_3	$y_3 = f_3 + f_7$	$y_3 := (y_1 - y_3)\omega_4^1$	$y_3 := (y_2 - y_3)\omega_2^0$	$= \tilde{c}_6$
f_4	$y_4 = (f_0 - f_4)\omega_8^0$	$y_4 := y_4 + y_6$	$y_4 := y_4 + y_5$	$= \tilde{c}_1$
f_5	$y_5 = (f_1 - f_5)\omega_8^1$	$y_5 := y_5 + y_7$	$y_5 := (y_4 - y_5)\omega_2^0$	$= \tilde{c}_5$
f_6	$y_6 = (f_2 - f_6)\omega_8^2$	$y_6 := (y_4 - y_6)\omega_4^0$	$y_6 := y_6 + y_7$	$= \tilde{c}_3$
f_7	$y_7 = (f_3 - f_7)\omega_8^3$	$y_7 := (y_5 - y_7)\omega_4^1$	$y_7 := (y_6 - y_7)\omega_2^0$	$= \tilde{c}_7$
	$N = 8,\ n := 4,\ \omega_8 = \mathrm{e}^{-\frac{2\pi\mathrm{i}}{8}}$	$N := 4,\ n := 2,\ \omega_4 = \omega_8^2$	$N := 2,\ n := 1,\ \omega_2 = \omega_4^2$	

It can be observed how terms with even and odd indices appear. In **Scheme 2** (19.233) the structure of the method is illustrated.

Scheme 2:

$$\tilde{c}_k \Rightarrow \begin{cases} \tilde{c}_{2k} \Rightarrow \begin{cases} \tilde{c}_{4k} \Rightarrow \begin{cases} \tilde{c}_{8k} \\ \tilde{c}_{8k+4} \end{cases} \\ \tilde{c}_{4k+2} \Rightarrow \begin{cases} \tilde{c}_{8k+2} \\ \tilde{c}_{8k+6} \end{cases} \end{cases} \\ \tilde{c}_{2k+1} \Rightarrow \begin{cases} \tilde{c}_{4k+1} \Rightarrow \begin{cases} \tilde{c}_{8k+1} \\ \tilde{c}_{8k+5} \end{cases} \\ \tilde{c}_{4k+3} \Rightarrow \begin{cases} \tilde{c}_{8k+3} \\ \tilde{c}_{8k+7} \end{cases} \end{cases} \end{cases} \tag{19.233}$$

$(k = 0, 1, \ldots, 7)$ $(k = 0, 1, 2, 3)$ $(k = 0, 1)$ $(k = 0)$.

If the coefficients $\tilde{c}_k$ are substituted into **Scheme 1** and one considers the binary forms of the indices before step 1 and after step 3, then it is easy to recognize that the order of the required coefficients can be obtained by simply *reversing the order of the bits* of the binary form of their indices. This is shown in **Scheme 3**.

Scheme 3:	Index	Step 1	Step 2	Step 3	Index
$\tilde{c}_0$	000	$\tilde{c}_0$	$\tilde{c}_0$	$\tilde{c}_0$	000
$\tilde{c}_1$	00L	$\tilde{c}_2$	$\tilde{c}_4$	$\tilde{c}_4$	L00
$\tilde{c}_2$	0L0	$\tilde{c}_4$	$\tilde{c}_2$	$\tilde{c}_2$	0L0
$\tilde{c}_3$	0LL	$\tilde{c}_6$	$\tilde{c}_6$	$\tilde{c}_6$	LL0
$\tilde{c}_4$	L00	$\tilde{c}_1$	$\tilde{c}_1$	$\tilde{c}_1$	00L
$\tilde{c}_5$	L0L	$\tilde{c}_3$	$\tilde{c}_5$	$\tilde{c}_5$	L0L
$\tilde{c}_6$	LL0	$\tilde{c}_5$	$\tilde{c}_3$	$\tilde{c}_3$	0LL
$\tilde{c}_7$	LLL	$\tilde{c}_7$	$\tilde{c}_7$	$\tilde{c}_7$	LLL

■ In the case of the function $f(x) = \begin{cases} 2\pi^2 & \text{for } x = 0, \\ x^2 & \text{for } 0 < x < 2\pi, \end{cases}$ with period 2π, the FFT is used for the discrete Fourier transformation. $N = 8$ is chosen. With $x_\nu = \frac{2\pi}{8}$, $f_\nu = \frac{1}{8} f(x_\nu)$ $(\nu = 0, 1, 2, \ldots, 7)$, $\omega_8 = e^{-\frac{2\pi\mathrm{i}}{8}} = 0.707107(1 - \mathrm{i})$, $\omega_8^2 = -\mathrm{i}$, $\omega_8^3 = -0.707107(1 + \mathrm{i})$ **Scheme 4** is got:

Scheme 4:	Step 1	Step 2	Step 3
$f_0 = 2.467401$	$y_0 = 3.701102$	$y_0 = 6.785353$	$y_0 = 13.262281 = \tilde{c}_0$
$f_1 = 0.077106$	$y_1 = 2.004763$	$y_1 = 6.476928$	$y_1 = 0.308425 = \tilde{c}_4$
$f_2 = 0.308425$	$y_2 = 3.084251$	$y_2 = 0.616851$	$y_2 = 0.616851 + 2.467402\,\mathrm{i} = \tilde{c}_2$
$f_3 = 0.693957$	$y_3 = 4.472165$	$y_3 = 2.467402\,\mathrm{i}$	$y_3 = 0.616851 - 2.467402\,\mathrm{i} = \tilde{c}_6$
$f_4 = 1.233701$	$y_4 = 1.233700$	$y_4 = 1.233700 + 2.467401\,\mathrm{i}$	$y_4 = 2.106058 + 5.956833\,\mathrm{i} = \tilde{c}_1$
$f_5 = 1.927657$	$y_5 = -1.308537(1 - \mathrm{i})$	$y_5 = 0.872358 + 3.489432\,\mathrm{i}$	$y_5 = 0.361342 - 1.022031\,\mathrm{i} = \tilde{c}_5$
$f_6 = 2.775826$	$y_6 = 2.467401\,\mathrm{i}$	$y_6 = 1.233700 - 2.467401\,\mathrm{i}$	$y_6 = 0.361342 + 1.022031\,\mathrm{i} = \tilde{c}_3$
$f_7 = 3.778208$	$y_7 = 2.180895(1 + \mathrm{i})$	$y_7 = -0.872358 + 3.489432\,\mathrm{i}$	$y_7 = 2.106058 - 5.956833\,\mathrm{i} = \tilde{c}_7$

From the third (last) reduction step the required real Fourier coefficients are obtained according to (19.223). (See the right-hand side.)
In this example, the general property

$a_0 = 26.524\,562$	
$a_1 = 4.212\,116$	$b_1 = -11.913\,666$
$a_2 = 1.233\,702$	$b_2 = -\ 4.934\,804$
$a_3 = 0.722\,684$	$b_3 = -\ 2.044\,062$
$a_4 = 0.616\,850$	$b_4 = 0$

$$\tilde{c}_{N-k} = \bar{\tilde{c}}_k \tag{19.234}$$

of the discrete complex Fourier coefficients can be observed. For $k = 1, 2, 3$, it can be observed that $\tilde{c}_7 = \bar{\tilde{c}}_1$, $\tilde{c}_6 = \bar{\tilde{c}}_2$, $\tilde{c}_5 = \bar{\tilde{c}}_3$.

19.7 Representation of Curves and Surfaces with Splines

19.7.1 Cubic Splines

Since interpolation and approximation polynomials of higher degree usually have unwanted oscillations, it is useful to divide the approximation interval into subintervals by the so-called *nodes* and to consider a relatively simple approximation function on every subinterval. In practice, cubic polynomials are mostly used. A smooth transition is required at the nodes of this piecewise approximation.

19.7.1.1 Interpolation Splines

1. Definition of the Cubic Interpolation Splines, Properties

Suppose there are given N interpolation points (x_i, f_i) $(i = 1, 2, \ldots, N;\ x_1 < x_2 < \ldots x_N)$. The *cubic interpolation spline* $S(x)$ is determined uniquely by the following properties:

1. $S(x)$ satisfies the interpolation conditions $S(x_i) = f_i$ $(i = 1, 2, \ldots, N)$.
2. $S(x)$ is a polynomial of degree ≤ 3 in any subinterval $[x_i, x_{i+1}]$ $(i = 1, 2, \ldots, N-1)$.
3. $S(x)$ is twice continuously differentiable in the entire approximation interval $[x_1, x_N]$.
4. $S(x)$ satisfies the special boundary conditions:
 a) $S''(x_1) = S''(x_N) = 0$ (we call them *natural splines*) or
 b) $S'(x_1) = {f_1}'$, $S'(x_N) = {f_N}'$ (${f_1}'$ and ${f_N}'$ are given values) or
 c) $S(x_1) = S(x_N)$, in the case of $f_1 = f_N$, $S'(x_1) = S'(x_N)$ and $S''(x_1) = S''(x_N)$ (they are called *periodic splines*).

It follows from these properties that for all twice continuously differentiable functions $g(x)$ satisfying the interpolation conditions $g(x_i) = f_i$ $(i = 1, 2, \ldots, N)$

$$\int_{x_1}^{x_N} [S''(x)]^2\, dx \le \int_{x_1}^{x_N} [g''(x)]^2\, dx \tag{19.235}$$

is valid (*Holladay's Theorem*). Based on (19.235) one can say that $S(x)$ has *minimal total curvature*, since for the curvature κ of a given curve, in a first approximation, $\kappa \approx S''$ (see 3.6.1.2, **4.**, p. 246). It can be shown that if a thin elastic ruler (its name is spline) is led through the points (x_i, f_i) $(i = 1, 2, \ldots, N)$, its bending line follows the cubic spline $S(x)$.

2. Determination of the Spline Coefficients

The cubic interpolation spline $S(x)$ for $x \in [x_i, x_{i+1}]$ has the form:

$$S(x) = S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \quad (i = 1, 2, \ldots, N-1). \tag{19.236}$$

The length of the subinterval is denoted by $h_i = x_{i+1} - x_i$. The coefficients of the natural spline can be determined in the following way:

1. From the interpolation conditions we get

$$a_i = f_i \quad (i = 1, 2, \ldots, N-1). \tag{19.237}$$

It is reasonable to introduce the additional coefficient $a_N = f_N$, which does not occur in the polynomials.

2. The continuity of $S''(x)$ at the interior nodes requires that

$$d_{i-1} = \frac{c_i - c_{i-1}}{3h_{i-1}} \quad (i = 2, 3, \ldots, N-1). \tag{19.238}$$

The natural conditions result in $c_1 = 0$, and (19.238) still holds for $i = N$, if $c_N = 0$ is introduced.

3. The continuity of $S(x)$ at the interior nodes results in the relation

$$b_{i-1} = \frac{a_i - a_{i-1}}{h_{i-1}} - \frac{2c_{i-1} + c_i}{3} h_{i-1} \quad (i = 2, 3, \ldots, N). \tag{19.239}$$

4. The continuity of $S'(x)$ at the interior nodes requires that

$$c_{i-1}h_{i-1} + 2(h_{i-1} + h_i)c_i + c_{i+1}h_i = 3\left(\frac{a_{i+1} - a_i}{h_i} - \frac{a_i - a_{i-1}}{h_{i-1}}\right) \quad (i = 2, 3, \ldots, N-1). \tag{19.240}$$

Because of (19.237), the right-hand side of the linear equation system (19.240) to determine the coefficients c_i $(i = 2, 3, \ldots, N-1;\ c_1 = c_N = 0)$ is known. The left hand-side has the following form:

$$\begin{pmatrix} 2(h_1+h_2) & h_2 & & & \\ h_2 & 2(h_2+h_3) & h_3 & \mathbf{O} & \\ & h_3 & 2(h_3+h_4) & h_4 & \\ & & \ddots & \ddots & \ddots \\ & \mathbf{O} & & & h_{N-2} \\ & & & h_{N-2} & 2(h_{N-2}+h_{N-1}) \end{pmatrix} \begin{pmatrix} c_2 \\ c_3 \\ c_4 \\ \vdots \\ \\ c_{N-1} \end{pmatrix}. \tag{19.241}$$

The coefficient matrix is *tridiagonal*, so the system of equations (19.240) can be solved numerically very easily by an LR decomposition (see 19.2.1.1, **2.**, p. 956). Then all other coefficients in (19.239) and (19.238) can be determined with these values c_i.

19.7.1.2 Smoothing Splines

The given function values f_i are usually measured values in practical applications so they have some error. In this case, the interpolation requirement is not reasonable. This is the reason why *cubic smoothing splines* are introduced. This spline is obtained if in the cubic interpolation splines the interpolation requirements are replaced by

$$\sum_{i=1}^{N}\left[\frac{f_i - S(x_i)}{\sigma_i}\right]^2 + \lambda \int_{x_1}^{x_N} [S''(x)]^2 \, dx = \min!. \tag{19.242}$$

The requirements of continuity of S, S' and S'' are kept, so the determination of the coefficients is a constrained optimization problem with conditions given in equation form. The solution can be obtained by using a Lagrange function (see 6.2.5.6, p. 456). For details see [19.26].

In (19.242) λ $(\lambda \geq 0)$ represents a *smoothing parameter* , which must be given previously. For $\lambda = 0$ the result is the cubic interpolation spline, as a special case. For "large" λ the result is a smooth approximation curve, but it returns the measured values inaccurately, and for $\lambda = \infty$ the result is the approximating regression line as another special case. A suitable choice of λ can be made, e.g., by computer-screen dialog. The parameter σ_i $(\sigma_i > 0)$ in (19.242) represents the *standard deviation* (see 16.4.1.3, **2.**, p. 851) of the measurement errors, of the values f_i $(i = 1, 2, \ldots, N)$.

Until now, the abscissae of the interpolation points and the measurement points were the same as the nodes of the spline function. For large N this method results in a spline containing a large number of cubic functions (19.236). A possible solution is to choose the number and the position of the nodes freely, because in many practical applications only a few spline segments are satisfactory. It is reasonable also from a numerical viewpoint to replace (19.236) by a spline of the form

$$S(x) = \sum_{i=1}^{r+2} a_i N_{i,4}(x). \tag{19.243}$$

Here r is the number of freely chosen nodes, and the functions $N_{i,4}(x)$ are the so-called *normalized B-splines* (*basis splines*) of order 4, i.e., polynomials of degree three, with respect to the i-th node. For details see [19.5].

19.7.2 Bicubic Splines

19.7.2.1 Use of Bicubic Splines

Bicubic splines are used for the following problem: A rectangle R of the x, y plane, given by $a \leq x \leq b$, $c \leq y \leq d$, is decomposed by the *grid points* (x_i, y_j) $(i = 0, 1, \ldots, n;\ j = 0, 1, \ldots, m)$ with

$$a = x_0 < x_1 < \cdots < x_n = b, \quad c = y_0 < y_1 < \cdots < y_m = d \tag{19.244}$$

into *subdomains* R_{ij}, where the subdomain R_{ij} contains the points (x, y) with $x_i \leq x \leq x_{i+1}$, $y_j \leq y \leq y_{j+1}$ $(i = 0, 1, \ldots, n-1;\ j = 0, 1, \ldots, m-1)$. The values of the function $f(x, y)$ are given at the grid points

$$f(x_i, y_j) = f_{ij} \quad (i = 0, 1, \ldots, n;\ j = 0, 1, \ldots, m). \tag{19.245}$$

A possible simple, smooth surface over R is required which approximates the points (19.245).

19.7.2.2 Bicubic Interpolation Splines

1. Properties

The bicubic interpolation spline $S(x, y)$ is defined uniquely by the following properties:

1. $S(x, y)$ satisfies the interpolation conditions

$$S(x_i, y_j) = f_{ij} \quad (i = 0, 1, \ldots, n;\ j = 0, 1, \ldots, m). \tag{19.246}$$

2. $S(x, y)$ is identical to a bicubic polynomial on every R_{ij} of the rectangle R, that is,

$$S(x, y) = S_{ij}(x, y) = \sum_{k=0}^{3}\sum_{l=0}^{3} a_{ijkl}(x - x_i)^k (y - y_j)^l \tag{19.247}$$

on R_{ij}. So, $S_{ij}(x, y)$ is determined by 16 coefficients, and for the determination of $S(x, y)$ $16 \cdot m \cdot n$ coefficients are needed.

3. The derivatives

$$\frac{\partial S}{\partial x},\quad \frac{\partial S}{\partial y},\quad \frac{\partial^2 S}{\partial x \partial y} \tag{19.248}$$

are continuous on R. So, a certain smoothness is ensured for the entire surface.

4. $S(x, y)$ satisfies the special boundary conditions:

$$\begin{aligned} \frac{\partial S}{\partial x}(x_i, y_j) &= p_{ij} \quad \text{for} \quad i = 0, n;\ j = 0, 1, \ldots, m, \\ \frac{\partial S}{\partial y}(x_i, y_j) &= q_{ij} \quad \text{for} \quad i = 0, 1, \ldots, n;\ j = 0, m, \\ \frac{\partial^2 S}{\partial x \partial y}(x_i, y_j) &= r_{ij} \quad \text{for} \quad i = 0, n;\ j = 0, m. \end{aligned} \tag{19.249}$$

Here p_{ij}, q_{ij} and r_{ij} are previously given values.
The results of one-dimensional cubic spline interpolation can be used for the determination of the coefficients a_{ijkl}.

1. There is a very large number $(2n + m + 3)$ of linear systems of equations but only with tridiagonal coefficient matrices.

2. The linear systems of equations differ from each other only on their right-hand sides.

In general, it can be said that bicubic interpolation splines are useful with respect to computation cost and accuracy, and so they are appropriate procedures for practical applications. For practical methods of computing the coefficients see the literature.

2. Tensor Product Approach

The bicubic spline approach (19.247) is an example of the so-called *tensor product* approach having the form

$$S(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{m} a_{ij} g_i(x) h_j(y) \tag{19.250}$$

and which is especially suitable for approximations over a rectangular grid. The functions $g_i(x)$ $(i = 0, 1, \ldots, n)$ and $h_j(y)$ $(j = 0, 1, \ldots, m)$ form two linearly independent function systems. The tensor product approach has the big advantage, from numerical viewpoint, that, e.g., the solution of a two-dimensional interpolation problem (19.246) can be reduced to a one-dimensional one. Furthermore, the two-dimensional interpolation problem (19.246) is uniquely solvable with the approach (19.250) if

1. the one-dimensional interpolation problem with functions $g_i(x)$ with respect to the interpolation nodes $x_0, x_1, \ldots, x_n$ and

2. the one-dimensional interpolation problem with functions $h_j(y)$ with respect to the interpolation nodes $y_0, y_1, \ldots, y_m$
are uniquely solvable.

An important tensor product approach is that with the cubic B-splines:

$$S(x, y) = \sum_{i=1}^{r+2} \sum_{j=1}^{p+2} a_{ij} N_{i,4}(x) N_{j,4}(y). \tag{19.251}$$

Here, the functions $N_{i,4}(x)$ and $N_{j,4}(y)$ are normalized B-splines of order four. Here r denotes the number of nodes with respect to x, p denotes the number of nodes with respect to y. The nodes can be chosen freely but their positions must satisfy certain conditions for the solvability of the interpolation problem.

The B-spline approach results in a system of equations with a band structured coefficient matrix, which is a numerically useful structure.

For solutions of different interpolation problems using bicubic B-splines see the literature.

19.7.2.3 Bicubic Smoothing Splines

The one-dimensional cubic approximation spline is mainly characterized by the optimality condition (19.242). For the two-dimensional case there could be determined a whole sequence of corresponding optimality conditions, however only a few special cases make the existence of a unique solution possible. For appropriate optimality conditions and algorithms for solution of the approximation problem with bicubic B-splines see the literature.

19.7.3 Bernstein–Bézier Representation of Curves and Surfaces

1. Bernstein Basis Polynomials

The Bernstein–Bézier representation (briefly B–B representation) of curves and surfaces applies the *Bernstein polynomials*

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad (i = 0, 1, \ldots, n) \tag{19.252}$$

and uses the following fundamental properties:

1. $0 \le B_{i,n}(t) \le 1 \text{ for } 0 \le t \le 1,$ (19.253)

2. $\sum\limits_{i=0}^{n} B_{i,n}(t) = 1.$ (19.254)

Formula (19.254) follows directly from the binomial theorem (see 1.1.6.4, p. 12).

■ **A:** $B_{01}(t) = 1 - t\,,\ B_{1,1}(t) = t$ **(Fig. 19.12)**.

■ **B:** $B_{03}(t) = (1-t)^3\,,\ B_{1,3}(t) = 3t(1-t)^2\,,\ B_{2,3}(t) = 3t^2(1-t)\ B_{3,3}(t) = t^3$ **(Fig. 19.13)**.

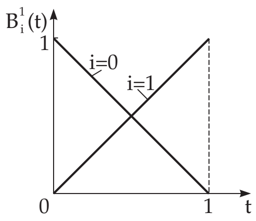


Figure 19.12

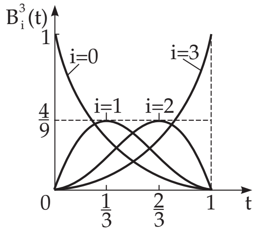


Figure 19.13

2. Vector Representation

Now, a space curve, whose parametric representation is $x = x(t)$, $y = y(t)$, $z = z(t)$, will be denoted in vector form by

$$\vec{\mathbf{r}} = \vec{\mathbf{r}}(t) = x(t)\,\vec{\mathbf{e}}_x + y(t)\,\vec{\mathbf{e}}_y + z(t)\,\vec{\mathbf{e}}_z. \tag{19.255}$$

Here t is the parameter of the curve. The corresponding representation of a surface is

$$\vec{\mathbf{r}} = \vec{\mathbf{r}}(u,v) = x(u,v)\,\vec{\mathbf{e}}_x + y(u,v)\,\vec{\mathbf{e}}_y + z(u,v)\,\vec{\mathbf{e}}_z. \tag{19.256}$$

Here, u and v are the surface parameters.

19.7.3.1 Principle of the B–B Curve Representation

Suppose there are given $n+1$ vertices P_i $(i = 0, 1, \ldots, n)$ of a three-dimensional polygon with the position vectors $\vec{\mathbf{P}}_i$. Introducing the vector-valued function

$$\vec{\mathbf{r}}(t) = \sum_{i=0}^{n} B_{i,n}(t)\vec{\mathbf{P}}_i \tag{19.257}$$

a space curve is assigned to these points, which is called the B–B curve. Because of (19.254) formula (19.257) can be considered as a "variable convex combination" of the given points. The three-dimensional curve (19.257) has the following important properties:

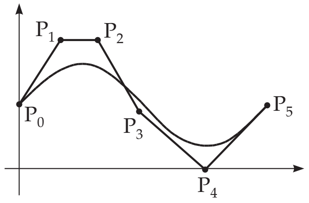


Figure 19.14

1. The points P_0 and P_n are interpolated.

2. Vectors $\overrightarrow{P_0P_1}$ and $\overrightarrow{P_{n-1}P_n}$ are tangents to $\vec{\mathbf{r}}(t)$ at points P_0 and P_n.

The relation between a polygon and a B–B curve is shown in **Fig. 19.14**.

The B–B representation is considered as a design of the curve, since it is easy to influence the shape of the curve by changing the polygon vertices.

Often normalized B-splines are used instead of Bernstein polynomials.

The corresponding space curves are called the B-spline curves. Their shape corresponds basically to the B–B curves with the following advantages:

1. The polygon is better approximated.

2. The B-spline curve changes only locally if the polygon vertices are changed.

3. In addition to the local changes of the shape of the curve the differentiability can also be influenced. So, it is possible to produce break points and line segments for example.

19.7.3.2 B–B Surface Representation

Suppose there are given the points P_{ij} $(i = 0, 1, \ldots, n;\ j = 0, 1, \ldots, m)$ with the position vectors $\vec{\mathbf{P}}_{ij}$, which can be considered as the nodes of a grid along the parameter curves of a surface. Analogously to the B–B curves (19.257), a surface is assigned to the grid points by

$$\vec{\mathbf{r}}(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,n}(u) B_{j,m}(v) \vec{\mathbf{P}}_{ij}. \tag{19.258}$$

Representation (19.258) is useful for surface design, since by changing the grid points the surface can be changed. Anyway, the influence of every grid point is global, so one should change from the Bernstein polynomials to the B-splines in (19.258).

19.8 Using the Computer

19.8.1 Internal Symbol Representation

Computers are machines that work with symbols. The interpretation and processing of these symbols is determined and controlled by the software. The external symbols, letters, cyphers and special symbols are internally represented in binary code by a form of bit sequence. A *bit* (binary digit) is the smallest representable information unit with values 0 and 1. Eight bits form the next unit, the *byte*. In a byte one can distinguish between 2^8 bit combinations, so 256 symbols can be assigned to them. Such an assignment is called a *code*. There are different codes; one of the most widespread is **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange).

19.8.1.1 Number Systems

1. Law of Representation

Numbers are represented in computers in a sequence of consecutive bytes. The basis for the internal representation is the binary system, which belongs to the polyadic systems, similarly to the decimal system.

The law of representation for a polyadic number system is

$$a = \sum_{i=-m}^{n} z_i B^i \quad (m > 0,\ n \geq 0;\ m, n \text{ integer}) \tag{19.259}$$

with B as basis and z_i $(0 \leq z_i < B)$ as a digit of the number system. The positions $i \geq 0$ form the integers, those with $i < 0$ the fractional part of the number.

■ The *decimal number representation*, i.e., $B = 10$, of the decimal number 139.8125 has the form $139.8125 = 1 \cdot 10^2 + 3 \cdot 10^1 + 9 \cdot 10^0 + 8 \cdot 10^{-1} + 1 \cdot 10^{-2} + 2 \cdot 10^{-3} + 5 \cdot 10^{-4}$.

The *number systems* occurring most often in computers are shown in **Table 19.3**.

Table 19.3 Number systems

Number system	**Basis**	**Corresponding digits**
Binary system	2	$0, 1$
Octal system	8	$0, 1, 2, 3, 4, 5, 6, 7$
Hexadecimal system	16	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, A,B,C,D,E,F (The letters A–F are for the values 10–15.)
Decimal system	10	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$

2. Conversion

The transition from one number system to another is called *conversion*. If different number systems are used in the same time, in order to avoid confusion the basis is denoted as an index.

■ The decimal number 139.8125 is in different systems: $139.8125_{10} = 10001011.1101_2 = 213.64_8 = 8\text{B.D}_{16}$.

1. Conversion of Binary Numbers into Octal or Hexadecimal Numbers The conversion of binary numbers into octal or hexadecimal numbers is simple. Groups of three or four bits are formed starting at the binary point to the left and to the right, and their values are determined. These values are the digits of the octal or hexadecimal systems.

2. Conversion of Decimal Numbers into Binary, Octal or Hexadecimal Numbers For the conversion of a decimal numbers into another system, the following rules are applied for the integer and for the fractional part separately:

a) Integer Part: If G is an integer in the decimal system, then for the number system with basis B the law of formation (19.259) is:

$$G = \sum_{i=0}^{n} z_i B^i \quad (n \geq 0). \tag{19.260}$$

If G is divided by B, then an integer part (the sum) is obtained and a residue:

$$\frac{G}{B} = \sum_{i=1}^{n} z_i B^{i-1} + \frac{z_0}{B}\,. \tag{19.261}$$

Here, z_0 can have the values $0, 1, \ldots, B-1$, and it is the lowest valued digit of the required number. If this method is repeated for the quotients, further digits can be got.

b) Fractional Part: If g is a proper fraction, then the method to convert it into the number system with basis B is

$$gB = z_{-1} + \sum_{i=2}^{m} z_{-i} B^{-i+1}, \tag{19.262}$$

i.e., the next digit is obtained as the integer part of the product gB. The values z_{-2}, z_{-3}, … can be obtained in the same way.

■ **A:** Conversion of the decimal number 139 into a binary number.

$139 : 2 = 69$	residue	1	$(1 = z_0)$
$69 : 2 = 34$	residue	1	$(1 = z_1)$
$34 : 2 = 17$	residue	0	$(0 = z_2)$
$17 : 2 = 8$	residue	1	:
$8 : 2 = 4$	residue	0	:
$4 : 2 = 2$	residue	0	:
$2 : 2 = 1$	residue	0	:
$1 : 2 = 0$	residue	1	$(1 = z_7)$

$139_{10} = 10001011_2$

■ **B:** Conversion of a decimal fraction 0.8125 into a binary fraction.

$0.8125 \cdot 2 = 1.625$	$(1 = z_{-1})$
$0.625 \cdot 2 = 1.25$	$(1 = z_{-2})$
$0.25 \cdot 2 = 0.5$	$(0 = z_{-3})$
$0.5 \cdot 2 = 1.0$	$(1 = z_{-4})$
$0.0 \cdot 2 = 0.0$	

$0.8125_{10} = 0.1101_2$

3. Conversion of Binary, Octal, and Hexadecimal Numbers into a Decimal Number The algorithm for the conversion of a value from the binary, octal, or hexadecimal system into the decimal system is the following, where the decimal point is after z_0:

$$a = \sum_{i=-m}^{n} z_i B^i \quad (m > 0,\ n \geq 0, \text{integer}). \tag{19.263}$$

The calculation is convenient with the Horner rule (see 19.1.2.1, p. 952).

■ $LLLOL = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 29.$

The corresponding Horner scheme is shown on the right.

$$\begin{array}{c|ccccc} & 1 & 1 & 1 & 0 & 1 \\ 2 & & 2 & 6 & 14 & 28 \\ \hline & 1 & 3 & 7 & 14 & \boxed{29} \end{array}$$

19.8.1.2 Internal Number Representation INR

Binary numbers are represented in computers in one or more bytes. Two types of form of representation are distinguished, the *fixed-point numbers* and the *floating-point numbers*. In the first case, the decimal point is at a fixed place, in the second case it is “floating” with the change of the exponent.

1. Fixed-Point Numbers
The range for fixed-point numbers with the given parameters is

$$0 \leq |\, a\, | \leq 2^t - 1. \tag{19.264}$$

Fixed-point numbers can be represented in the form of **Fig. 19.15**.

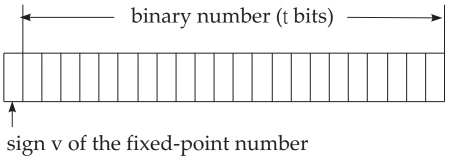


Figure 19.15

2. Floating-Point Numbers
Basically, two different forms are in use for the representation of floating-point numbers, where the internal implementation can vary in detail.

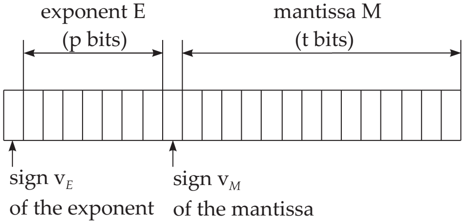


Figure 19.16

1. Normalized Semilogarithmic Form
In the first form, the signs of the exponent E and the mantissa M of the number a are stored separately

$$a = \pm M B^{\pm E}. \tag{19.265a}$$

Here the exponent E is chosen so that for the mantissa

$$1/B \leq M < 1 \tag{19.265b}$$

holds. It is called the *normalized semilogarithmic form* **(Fig. 19.16)**.

The range of the absolute value of the floating-point numbers with the given parameters is:

$$2^{-2^p} \leq |\, a \,| \leq \left(1 - 2^{-t}\right) \cdot 2^{(2^p - 1)}. \qquad (19.266)$$

2. IEEE Standard The second (nowadays used) form of floating-point numbers corresponds to the **IEEE** (**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers) *standard* accepted in 1985. It deals with the requirements of computer arithmetic, roundoff behavior, arithmetical operators, conversion of numbers, comparison operators and handling of exceptional cases such as over- and underflow.

The floating-point number representations are shown in **Fig. 19.17**.

The characteristic C comes from the exponent E by addition of a suitable constant K. This is chosen so that only positive numbers occur in the characteristic. The representable number is

$$a = (-1)^v \cdot 2^E \cdot 1.b_1 b_2 \ldots b_{t-1} \quad \text{with } E = C - K. \qquad (19.267)$$

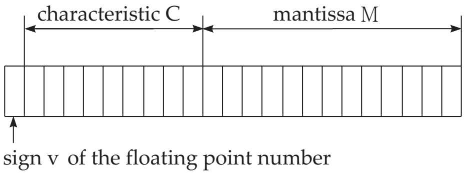


Figure 19.17

Here: $C_{min} = 1$, $C_{max} = 254$, since $C = 0$ and $C = 255$ are reserved.
The standard gives two basic forms of representation (single-precision and double-precision floating-point numbers), but other representations are also possible. **Table 19.4** contains the parameters for the basic forms.

Table 19.4 Parameters for the basic forms

Parameter	**Single precision**	**Double precision**
Word length in bits	32	64
Maximal exponent E_{max}	+127	+1023
Minimal exponent E_{min}	−126	−1022
Constant K	+127	+1023
Number of bits in exponent	8	11
Number of bits in mantissa	24	53

19.8.2 Numerical Problems in Calculations with Computers

19.8.2.1 Introduction, Error Types

The general properties of calculations with a computer are basically the same as those of calculations done by hand, however some of them need special attention, because the accuracy comes from the representation of the numbers, and from the missing judgement with respect to the errors of the computer. Furthermore, computers perform many more calculation steps than human can do manually.

So, there is the problem of how to influence and control the errors, e.g., by choosing the most appropriate numerical method among the mathematically equivalent methods.

In further discussions, the following notation is used, where x denotes the exact value of a quantity, which is mostly unknown, and $\tilde{x}$ is an approximation value of x:

Absolute error: $|\Delta x| = |x - \tilde{x}|$. (19.268) Relative error: $\left|\dfrac{\Delta x}{x}\right| = \left|\dfrac{x - \tilde{x}}{x}\right|$. (19.269)

The notations

$$\epsilon(x) = x - \tilde{x} \quad \text{and} \quad \epsilon_{rel}(x) = \frac{x - \tilde{x}}{x} \qquad (19.270)$$

are also often used.

19.8.2.2 Normalized Decimal Numbers and Round-Off

1. Normalized Decimal Numbers

Every real number $x \neq 0$ can be expressed as a decimal number in the form

$$x = \pm 0.b_1 b_2 \ldots \cdot 10^E \quad (b_1 \neq 0). \tag{19.271}$$

Here $0, b_1 b_2 \ldots$ is called the *mantissa* formed with the digits $b_i \in \{0, 1, 2, \ldots, 9\}$. The number E is an integer, the so-called exponent with respect to the base 10. Since $b_1 \neq 0$, (19.271) is called a *normalized decimal number*.

Since only finitely many digits can be handled by a real computer, one has to restrict himself to a fixed number t of mantissa digits and to a fixed range of the exponent E. So, from the number x given in (19.271) the number

$$\tilde{x} = \begin{cases} \pm 0.b_1 b_2 \cdots b_t \cdot 10^E & \text{for} \quad b_{t+1} \leq 5 \ \text{(round-down)}, \\ \pm(0.b_1 b_2 \cdots b_t + 10^{-t})10^E & \text{for} \quad b_{t+1} > 5 \ \text{(round-up)}, \end{cases} \tag{19.272}$$

is obtained by round-off (as it is usual in practical calculations). The absolute error caused by round-off

$$|\Delta x| = |x - \tilde{x}| \leq 0.5 \cdot 10^{-t} 10^E. \tag{19.273}$$

2. Basic Operations and Numerical Calculations

Every numerical process is a sequence of basic calculation operations. Problems arise especially with the finite number of positions in the floating-point representation. Here a short overview is given. It is supposed that x and y are normalized error-free floating-point numbers with the same sign and with a non-zero value:

$$x = m_1 B^{E_1}, \quad y = m_2 B^{E_2} \quad \text{with} \tag{19.274a}$$

$$m_i = \sum_{k=1}^{t} a_{-k}^{(i)} B^{-k}, \quad a_{-1}^{(i)} \neq 0, \quad \text{and} \tag{19.274b}$$

$$a_{-k}^{(i)} = 0 \text{ or } 1 \text{ or} \ldots \text{ or} \quad B-1 \ \text{ for} \quad k > 1 \quad (i = 1, 2). \tag{19.274c}$$

1. Addition If $E_1 > E_2$, then the common exponent becomes E_1, since normalization allows us to make only a left-shift. The mantissas are then added.

If $\quad B^{-1} \leq | m_1 + m_2 B^{-(E_1 - E_2)} | < 2 \qquad (19.275a) \qquad$ and $\quad | m_1 + m_2 B^{-(E_1 - E_2)} | \geq 1, \quad (19.275b)$

then shifting the decimal point by one position to the left results in an increase of the exponent by one.

■ $0.9604 \cdot 10^3 + 0.5873 \cdot 10^2 = 0.9604 \cdot 10^3 + 0.05873 \cdot 10^3 = 1.01913 \cdot 10^3 = 0.1019 \cdot 10^4$.

2. Subtraction The exponents are equalized as in the case of addition, the mantissas are then subtracted. If

$| m_1 - m_2 B^{-(E_1 - E_2)} | < 1 - B^{-t} \ (19.276a) \qquad$ and $\quad | m_1 - m_2 B^{-(E_1 - E_2)} | < B^{-1}, \quad (19.276b)$

shifting the decimal point to the right by a maximum of t positions results in the corresponding decrease of the exponent.

■ $0.1004 \cdot 10^3 - 0.9988 \cdot 10^2 = 0.1004 \cdot 10^3 - 0.09988 \cdot 10^3 = 0.00052 \cdot 10^3 = 0.5200 \cdot 10^0$. This example shows the critical case of subtractive cancellation. Because of the limited number of positions (here four), zeros are carried in from the right instead of the correct characters.

3. Multiplication The exponents are added and the mantissas are multiplied. If

$$m_1 m_2 < B^{-1}, \tag{19.277}$$

then the decimal point is shifted to the right by one position, and the exponent is decreased by one.

■ $(0.3176 \cdot 10^3) \cdot (0.2504 \cdot 10^5) = 0.07952704 \cdot 10^8 = 0.7953 \cdot 10^7$.

4. Division The exponents are subtracted and the mantissas are divided. If

$$\frac{m_1}{m_2} \geq B^{-1}, \tag{19.278}$$

then the decimal point is shifted to the left by one position, and the exponent is increased by one.

■ $(0.3176 \cdot 10^3)/(0.2504 \cdot 10^5) = 1.2683706\ldots 10^{-2} = 0.1268 \cdot 10^{-1}$.

5. Error of the Result The error of the result in the four basic operations with terms that are supposed to be error-free is a consequence of round-off. For the relative error with number of positions t and the base B, the limit is

$$\frac{B}{2}B^{-t}. \tag{19.279}$$

6. Subtractive cancelation As it was mentioned above, the critical operation is the subtraction of nearly equal floating-point numbers. If it is possible, one should avoid this by changing the order of operations, or by using certain identities.

■ $x = \sqrt{1985} - \sqrt{1984} = 0.4455 \cdot 10^2 - 0.4454 \cdot 10^2 = 0.1 \cdot 10^{-1}$ or $x = \sqrt{1985} - \sqrt{1984} = \dfrac{1985 - 1984}{\sqrt{1985} + \sqrt{1984}} = 0.1122 \cdot 10^{-1}$

19.8.2.3 Accuracy in Numerical Calculations

1. Types of Errors

Numerical methods have errors. There are several types of errors, from which the total error of the final result is accumulated **(Fig. 19.18)**.

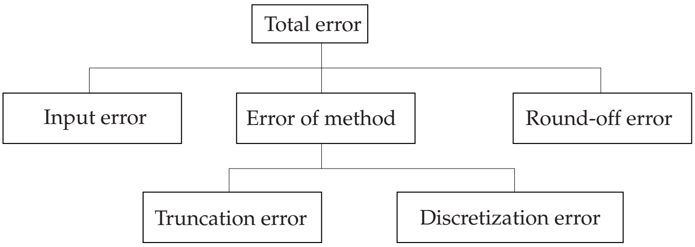


Figure 19.18

2. Input Error

1. Notion of Input Error *Input error* is the error of the result caused by inaccurate input data. Slight inaccuracies of input data are also called *perturbations*. The determination of the error of the input data is called the *direct problem of error calculus*. The *inverse problem* is the following: How large an error the input data may have such that the final input error does not exceed an acceptable tolerance value. The estimation of the input error in rather complex problems is very difficult and is usually hardly possible. In general, for a real-valued function $y = f(\underline{\mathbf{x}})$ with $\underline{\mathbf{x}} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ the absolute value of the input error is

$$\begin{aligned} |\Delta y| &= |f(x_1, x_2, \ldots, x_n) - f(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n)| \\ &= |\sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\xi_1, \xi_2, \ldots, \xi_n)(x_i - \tilde{x}_i)| \le \sum_{i=1}^{n} \left(\max_x |\frac{\partial f}{\partial x_i}(\underline{\mathbf{x}})| \right) |\Delta x_i|, \end{aligned} \tag{19.280}$$

if the Taylor formula (see 7.3.3.3, p. 471) is used for $y = f(x) = f(x_1, x_2, \ldots, x_n)$ with a linear residue. $\xi_1, \xi_2, \ldots, \xi_n$ denote the intermediate values, $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n$ denote the approximating values of $x_1, x_2, \ldots, x_n$. The approximating values are the perturbed input data. Here, also the Gauss error propagation law (see 16.4.2.1, p. 855) is considered.

2. Input Error of Simple Arithmetic Operations The input error is known for simple arithmetical operations. With the notation of (19.268)–(19.270) for the four basic operations:

$$\epsilon(x \pm y) = \epsilon(x) \pm \epsilon(y), \quad (19.281) \qquad \epsilon(xy) = y\epsilon(x) + x\epsilon(y) + \epsilon(x)\epsilon(y), \quad (19.282)$$

$$\epsilon\left(\frac{x}{y}\right) = \frac{1}{y}\epsilon(x) - \frac{x}{y^2}\epsilon(y) + \text{ terms of higher order in } \varepsilon, \quad (19.283)$$

$$\epsilon_{rel}(x \pm y) = \frac{x\epsilon_{rel}(x) \pm y\epsilon_{rel}(y)}{x \pm y}, \quad (19.284) \qquad \epsilon_{rel}(xy) = \epsilon_{rel}(x) + \epsilon_{rel}(y) + \epsilon_{rel}(x)\epsilon_{rel}(y), \quad (19.285)$$

$$\epsilon_{rel}\left(\frac{x}{y}\right) = \epsilon_{rel}(x) - \epsilon_{rel}(y) + \text{ terms of higher order in } \varepsilon. \quad (19.286)$$

The formulas show: Small relative errors of the input data result in small relative errors of the result on multiplication and division. For addition and subtraction, the relative error can be very large if $|x \pm y| \ll |x| + |y|$.

3. Error of the Method

1. Notion of the Error of the Method The *error of the method* comes from the fact that theoretically continuous phenomena are numerically approximated in many different ways as limits. Hence, there are *truncation errors* in limiting processes (as, e.g., in iteration methods) and *discretization errors* in the approximation of continuous phenomena by a finite discrete system (as, e.g., in numerical integration). Errors of methods exist independently of the input and round-off errors; consequently, they can be investigated only in connection with the applied solution methodology.

2. Applying Iteration Methods If an iteration method is used, then both cases may occur: A correct solution or also a false solution of the problem can be obtained. It is also possible that no solution is obtained by an iteration method although there exists one.
To make an iteration method clearer and safer, the following advices should be considered:

a) To avoid "infinite" iterations, count the number of steps and stop the process if this number exceeds a previously given value (i.e., stop without reaching the required accuracy).

b) The location of the intermediate result should be tracked on the screen by a numerical or a graphical representation of the intermediate results.

c) All known properties of the solution should be used such as gradient, monotonicity, etc.

d) The possibilities of scaling the variables and functions should be investigated.

e) Several tests should be performed by varying the step size, truncation conditions, initial values, etc.

4. Round-off Errors

Round-off errors occur because the intermediate results should be rounded. So, they have an essential importance in judging mathematical methods with respect to the required accuracy. They determine together with the errors of input and the error of the method, whether a given numerical method is strongly stable, weakly stable or unstable. Strong *stability*, weak stability, or *instability* occur if the total error, at an increasing number of steps, decreases, has the same order, or increases, respectively. At the instability one distinguishes between the sensitivity with respect to round-off errors and *discretization errors* (numerical instability) and with respect to the error in the initial data at a theoretically exact calculation (natural instability). A calculation process is appropriate if the numerical instability is not greater than the natural instability.

For the local error propagation of round-off errors, i.e., errors at the transition from a calculation step to the next one, the same estimation process can be used as the one applied at the input error.

5. Examples of Numerical Calculations

Some of the problems mentioned above are illustrated by numerical examples.

■ **A: Roots of a Quadratic Equation:**
$ax^2 + bx + c = 0$ with real coefficients a, b, c and $D = b^2 - 4ac \geq 0$ (real roots). Critical situations are the cases **a)** $| 4ac | \ll b^2$ and **b)** $4ac \approx b^2$. Recommended proceeding:

a) $x_1 = -\dfrac{b+\operatorname{sign}(b)\sqrt{D}}{2a}$, $x_2 = \dfrac{c}{ax_1}$ (Vieta root theorem, see 1.6.3.1, 3., p. 44).

b) The vanishing of D cannot be avoided by a direct method. Subtractive cancellation occurs but the error in $(b+\operatorname{sign}(b\sqrt{D})$ is not too large since $|b| \gg \sqrt{D}$ holds.

■ **B: Volume of a Thin Conical Shell for $h \ll r$**

$V = 4\pi\dfrac{(r+h)^3 - r^3}{3}$ because of $(r+h) \approx r$ there is a case of subtractive cancellation. However in the equation $V = 4\pi\dfrac{3r^2h + 3rh^2 + h^3}{3}$ there is no such problem.

■ **C: Determining the Sum $S = \sum\limits_{k=1}^{\infty}\dfrac{1}{k^2+1}$** $(S = 1.07667\ldots)$ with an accuracy of three significant digits. Performing the calculations with 8 digits, about 6000 terms should be added. After the identical transformation $\dfrac{1}{k^2+1} = \dfrac{1}{k^2} - \dfrac{1}{k^2(k^2+1)}$

$S = \sum\limits_{k=1}^{\infty}\dfrac{1}{k^2} - \sum\limits_{k=1}^{\infty}\dfrac{1}{k^2(k^2+1)}$ and $S = \dfrac{\pi^2}{6} - \sum\limits_{k=1}^{\infty}\dfrac{1}{k^2(k^2+1)}$ hold. By this transformation only eight terms are considered.

■ **D: Avoiding the $\dfrac{0}{0}$ Situation** in the function $z = (1-\sqrt{1+x^2+y^2})\dfrac{x^2-y^2}{x^2+y^2}$ for $x = y = 0$. Multiplying the numerator and the denominator by $(1+\sqrt{1+x^2+y^2})$ one avoids this situation.

■ **E: Example for an Unstable Recursive Process.** Algorithms with the general form $y_{n+1} = ay_n + by_{n-1}$ $(n = 1, 2, \ldots)$ are stable if the condition $\left|\dfrac{a}{2} \pm \sqrt{\dfrac{a^2}{4}+b}\right| < 1$ is satisfied. The special case $y_{n+1} = -3y_n + 4y_{n-1}$ $(n = 1, 2, \ldots)$ is unstable. If y_0 and y_1 have errors ε and $-\varepsilon$, then for $y_2, y_3, y_4, y_5, y_6, \ldots$ the errors are 7ε, -25ε, 103ε, -409ε, 1639ε, ... The process is instable for the parameters $a = -3$ and $b = 4$.

■ **F: Numerical Integration of a Differential Equation.** The numerical solution for the first-order ordinary differential equation

$$y' = f(x,y) \text{ with } f(x,y) = ay \tag{19.287}$$

and the initial value $y(x_0) = y_0$ will be represented.

a) Natural Instability. Together with the exact solution $y(x)$ for the exact initial values $y(x_0) = y_0$ let $u(x)$ be the solution for a perturbed initial value. Without loss of generality, it may be assumed that the perturbed solution has the form

$$u(x) = y(x) + \varepsilon\,\eta(x), \tag{19.288a}$$

where ε is a parameter with $0 < \varepsilon < 1$ and $\eta(x)$ is the so-called perturbation function. Considering that $u'(x) = f(x,u)$ one gets from the Taylor expansion (see 7.3.3.3, p. 471)

$$u'(x) = f(x, y(x) + \varepsilon\,\eta(x)) = f(x,y) + \varepsilon\,\eta(x)\,f_y(x,y) + \text{terms of higher order} \tag{19.288b}$$

which implies the so-called differential variation equation

$$\eta'(x) = f_y(x,y)\eta(x). \tag{19.288c}$$

The solution of the problem with $f(x,y) = ay$ is

$$\eta(x) = \eta_0\, e^{a(x-x_0)} \text{ with } \eta_0 = \eta(x_0). \tag{19.288d}$$

For $a > 0$ even a small initial perturbation η_0 results in an unboundedly increasing perturbation $\eta(x)$. So, there is a natural instability.

b) Investigation of the Error of the Method in the Trapezoidal Rule. With $a = -1$, the stable differential equation $y'(x) = -y(x)$ has the exact solution

$$y(x) = y_0 e^{-(x-x_0)}, \text{ where } y_0 = y(x_0). \tag{19.289a}$$

The trapezoidal rule is

$$\int\limits_{x_1}^{x_{i+1}} y(x)dx \approx \frac{y_i + y_{i+1}}{2} h \text{ with } h = x_{i+1} - x_i. \tag{19.289b}$$

By using this formula for the given differential equation

$$\tilde{y}_{i+1} = \tilde{y}_i + \int\limits_{x_i}^{x_{i+1}} (-y)dx = \tilde{y}_i - \frac{\tilde{y}_i + \tilde{y}_{i+1}}{2} h \quad \text{or} \quad \tilde{y}_{i+1} = \frac{2-h}{2+h}\tilde{y}_i \quad \text{or}$$

$$\tilde{y}_i = \left(\frac{2-h}{2+h}\right)^i \tilde{y}_0 \tag{19.289c}$$

is valid. With $x_i = x_0 + ih$, i.e., with $i = (x_i - x_0/h$ for $0 \le h < 2$

$$\tilde{y}_i = \left(\frac{2-h}{2+h}\right)^{(x_i - x_0)/h} \tilde{y}_0 = \tilde{y}_0 e^{c(h)(x_i - x_0)} \quad \text{with} \quad c(h) = \frac{\ln\left(\frac{2-h}{2+h}\right)}{h} = -1 - \frac{h^2}{12} - \frac{h^4}{80} - \cdots \tag{19.289d}$$

is obtained. If $\tilde{y}_0 = y_0$, then $\tilde{y}_i < y_i$, and so for $h \to 0$, $\tilde{y}_i$ also tends to the exact solution $y_0 e^{-(x_i - x_0)}$.

c) Input Error In **b)** it is supposed that the exact and the approximate initial values coincide. Now, the behavior is investigated when $y_0 \ne \tilde{y}_0$ with $|\ \tilde{y}_0 - y_0\ | < \varepsilon_0$.

$$\text{Since } (\tilde{y}_{i+1} - y_{i+1}) \le \frac{2-h}{2+h}(\tilde{y}_i - y_i) \text{ there is } (\tilde{y}_{i+1} - y_{i+1}) \le \left(\frac{2-h}{2+h}\right)^{i+1} (\tilde{y}_0 - y_0). \tag{19.290a}$$

So, ε_{i+1} is at most of the same order as ε_0, and the method is stable with respect to the initial values. It has to be mentioned that in solving the above differential equation with the Simpson method an artificial instability is introduced. In this case, for $h \to 0$, the general solution is obtained as

$$\tilde{y}_i = C_1 e^{-x_i} + C_2(-1)^i e^{x_i/3}. \tag{19.290b}$$

The problem is that the numerical solution method uses higher-order differences than those to which the order of the differential equation corresponds.

19.8.3 Libraries of Numerical Methods

Over time, *libraries* of functions and procedures have been developed independently of each other for numerical methods in different programming languages. An enormous amount of computer experimentation was considered in their development, so in solutions of practical numerical problems one should use the programs from one of these program libraries. Programs are available for current operating systems like WINDOWS, UNIX and LINUX and mostly for every computation problem type and they keep certain conventions, so it is more or less easy to use them.

The application of methods from program libraries does not relieve the user of the necessity of thinking about the expected results. This is a warning that the user should be informed about the advantages and also about the disadvantages and weaknesses of the mathematical method he/she is going to use.

19.8.3.1 NAG Library

The *NAG library* (**N**umerical **A**lgorithms **G**roup) is a rich collection of numerical methods in the form of functions and subroutines/procedures in the programming languages FORTRAN 77, FORTRAN 90 and C. Here is a contents overview:

1. Complex arithmetic
2. Roots of polynomials
3. Roots of transcendental equations
4. Series
5. Integration
6. Ordinary differential equations
7. Partial differential equations
8. Numeric differentiation
9. Integral equations
10. Interpolation
11. Approximation of curves and surfaces from data
12. Minimum/maximum of a function
13. Matrix operations, inversion
14. Eigenvalues and eigenvectors
15. Determinants
16. Simultaneous linear equations
17. Orthogonalization
18. Linear algebra
19. Simple calculations with statistical data
20. Correlation and regression analysis
21. Random number generators
22. Non-parametric statistics
23. Time series analysis
24. Operations research
25. Special functions
26. Mathematical and computer constants

Furthermore the NAG library contains extensive software concerning statistics and financial mathematics.

19.8.3.2 IMSL Library

The **IMSL library** (**I**nternational **M**athematical and **S**tatistical **L**ibrary) consists of three synchronized parts:
General mathematical methods,
Statistical problems,
Special functions.
The sublibraries contain functions and subroutines in FORTRAN 77, FORTRAN 90 and C. Here is a contents overview:

General Mathematical Methods

1. Linear systems
2. Eigenvalues
3. Interpolation and approximation
4. Integration and differentiation
5. Differential equations
6. Transformations
7. Non-linear equations
8. Optimization
9. Vector and matrix operations
10. Auxiliary functions

Statistical Problems

1. Elementary statistics
2. Regression
3. Correlation
4. Variance analysis
5. Categorization and discrete data analysis
6. Non-parametric statistics
7. Test of goodness of fit and test of randomness
8. Analysis of time series and forecasting
9. Covariance and factor analysis
10. Discriminance analysis
11. Cluster analysis
12. Random sampling
13. Life time distributions and reliability
14. Multidimensional scaling
15. Estimation of reliability function, hazard rate and risk function
16. Line-printer graphics
17. Probability distributions
18. Random number generators
19. Auxiliary algorithms
20. Auxiliary mathematical tools

Special Functions

1. Elementary functions
2. Trigonometric and hyperbolic functions
3. Exponential and related functions
4. Gamma function and relatives
5. Error functions and relatives
6. Bessel functions
7. Kelvin functions
8. Bessel functions with fractional orders
9. Weierstrass elliptic integrals and related functions
10. Different functions

19.8.3.3 Aachen Library

The **Aachen library** is based on the collection of formulas for numerical mathematics of G. Engeln–Müllges (Fachhochschule Aachen) and F. Reutter (Rheinisch–Westfälische Technische Hochschule Aachen). It exists in the programming languages BASIC, QUICKBASIC, FORTRAN 77, FORTRAN 90, C, MODULA 2 and TURBO PASCAL. Here is an overview:

1. Numerical methods to solve non-linear and special algebraic equations
2. Direct and iterative methods to solve systems of linear equations
3. Systems of non-linear equations
4. Eigenvalues and eigenvectors of matrices
5. Linear and non-linear approximation
6. Polynomial and rational interpolation, polynomial splines
7. Numerical differentiation
8. Numerical quadrature
9. Initial value problems of ordinary differential equations
10. Boundary value problems of ordinary differential equations

The programs of the Aachen library are especially suitable for the investigation of individual algorithms of numerical mathematics.

19.8.4 Application of Interactive Program Systems and Computeralgebra Systems

19.8.4.1 Matlab

The commercial program system Matlab Matlab (**Mat**rix **Lab**oratory) is an interactive environment for solving mathematically formulated problems and in the same time it is a high level script language for scientific technical computations. The set up priorities are the problems and algorithms of linear algebra. Matlab unifies the convenient well developed implementations of numerical procedures with advanced graphical representation of the results and data. The computations are processed mostly with double precision floating point numbers according to **IEEE**–*standards* (see **Table19.4**, p. 1004). As further alternatives which are compatible to Matlab are the systems Scilab and Octave with free downloads.

1. Functions Survey

There is a short survey of the procedures and functions available in Matlab:

General Mathematical Functions

1. Trigonometry
2. Exponential functions, Logarithms
3. Special functions
4. Complex arithmetic
5. Coordinate transformations
6. Round-off and fractions
7. Discrete mathematics
8. Mathematical constants

Numerical Linear Algebra

1. Manipulation of fields and matrices
2. Special matrices
3. Matrix analyzes (norms, condition)
4. Systems of linear equations
5. Eigenvalues and singular values
6. Matrix factorization
7. Matrix functions
8. Methods for sparse matrices

Numerical Methods

1. Calculation of statistical data
2. Correlation and regression
3. Discrete Fourier transformation
4. Polynomials and splines
5. One- and more-dimensional interpolation
6. Triangulations and decompositions
7. Determination of the convex closures
8. Numerical integration
9. Ordinary differential equations
10. Partial differential equations
11. Non-linear equations
12. Minimization of functions

In addition there are several program packages of Matlab, the so called toolbooxes, which can be applied in the cases of special mathematical classis of problems. As some examples can be mentioned here the curve fitting, filtering, business mathematics, time series analysis, signal and pattern processing, neural networks, optimization, partial differential equations, splines, statistics and wavelets.

In the following paragraphs the possibilities of Matlab are demonstrated by simple examples. The same problems are partially discussed here as in the paragraphs of numerical applications of Mathematica and Maple.

2. Numerical Linear Algebra

After starting with Matlab the command prompt >> appears in the command window to indicate the readiness to accept commands. If a command is not closed by a semicolon, then the result appears in the command window. The basic command to solve a system of linear equations $\mathbf{A}\,\underline{\mathbf{x}} = \underline{\mathbf{b}}$ (see 19.2.1, p. 955) is the backslash operator \.

■ Given matrix $\mathbf{A} = \begin{pmatrix} 1 & 0 & 3 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}$ and vector $\underline{\mathbf{b}} = (-2, 3, 2)^T$. For the input

```
>>    A = [1 0 3; 2 1 1; 1 2 3],    b = [-2; 3; 2],    x = A\b,    norm(A * x - b)
```

the output is

```
     1 0 3        -2          1.0000
A =  2 1 1   b =   3   x =    2.0000    ans = 8.8818e - 016
     1 2 3         2         -1.0000
```

As the Euclidean norm of the residual shows the obtained solution x (for which not all the digits are shown) satisfies the system of equations with an accuracy allowed by the mashine floating point representation.

If the matrix **A** is quadratic and nonsingular, then the linear system has a unique solution. By the backslash operator \ ordinary the Gaussian elimination is used with column pivoting, i.e., a triangle decomposition $\mathbf{PA} = \mathbf{LR}$ is obtained (see 19.2.1.1, p. 955).

■ The triangle decomposition of **A** can be realized also with the input

```
>>    [L, R, P] = lu(A)
```

giving the output

$$\mathtt{L} = \begin{pmatrix} 1.0000 & 0 & 0 \\ 0.5000 & 1.0000 & 0 \\ 0.5000 & -0.3333 & 1.0000 \end{pmatrix} \quad \mathtt{R} = \begin{pmatrix} 2.0000 & 1.0000 & 1.0000 \\ 0 & 1.5000 & 2.5000 \\ 0 & 0 & 3.3333 \end{pmatrix} \quad \mathtt{P} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

(Here the matrices are given in brackets in order to avoid confusion).

The backslash operator first tests the properties of the coefficient matrix **A**. If **A** is a permutation of a triangle matrix, then the corresponding echelon form is solved. For a symmetric **A** the Cholesky method is applied (see 19.2.1.2, p. 958).

If the condition number of the coefficient matrix **A** is too high, then numerical problems can occure during the solution. Because of this problem, during the procedure Matlab calculates an estimation of the reciprocal value of the condition number, and gives a warning if it is too small.

■ The Hilbert matrix $\mathbf{H} = (h_{ik})$ of order $n = 13$ can serve as an example, where $h_{ik} = 1/(i + k - 1)$.

```
>> x = hilb(13)\ones(13, 1);
   Warning:  Matrix is close to singular or badly scaled.
      Results may be inaccurate.  RCOND = 2.409320e-017.
```

In the case of overdetermined linear systems the corresponding linear fitting problem is handled by an orthogonalization procedure, i.e. by orthogonal transformations into a QR-decomposition $\mathbf{A} = \mathbf{QR}$

(see 19.2.1.3, p. 958).

■ $\gg$ `A = [1 0 3; 2 1 1; 1 2 3; 1 1 - 1];   b = [-2; 3; 2; 1];   x = A\b,   norm(A * x - b)`

$$\mathtt{x} = \begin{matrix} 0.4673 \\ 1.4393 \\ -0.4953 \end{matrix} \quad \mathtt{ans} = 2.0508$$

$\gg$ `[Q, R] = qr(A)`

$$\mathtt{Q} = \begin{pmatrix} -0.3780 & -0.4583 & 0.6466 & 0.4785 \\ -0.7559 & -0.2750 & -0.2321 & -0.5469 \\ -0.3780 & 0.8250 & 0.4145 & -0.0684 \\ -0.3780 & 0.1833 & -0.5969 & 0.6836 \end{pmatrix} \qquad \mathtt{R} = \begin{pmatrix} -2.6458 & -1.8898 & -2.6458 \\ 0 & 1.5584 & 0.6417 \\ 0 & 0 & 3.5480 \\ 0 & 0 & 0 \end{pmatrix}$$

The backslash operator also gives meaningful results in the cases of underdetermined and rank deficient linear systems of equations. The details of these cases with the ways how to handle large sparse matrices can be found in the corresponding documentation of Matlab and in the introductions [19.20], [19.29].

3. Numerical Solution of Equations

A polynomial

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

is in Matlab represented by the row vector $(a_n, a_{n-1}, \ldots, a_1, a_0)$ of the coefficients. Several functions are available to handle polynomials.

■ As an example the polynomial value at 1, the derivative (i.e. the coefficient vector of the derivative polynomial) and the roots are determined for the polynomial $p(x) = x^6 + 3x^2 - 5$.

```
>>   p = [1 0 0 0 3 0 - 5];
>>   polyval(p, 1)      ans = -1
>>   polyder(p)      ans = 6 0 0 0 6 0
>>   roots(p)      ans = 0.8673 + 1.1529i, 0.8673 - 1.1529i, 1.0743,
                              -0.8673 + 1.1529i, -0.8673 - 1.1529i, -1.0743
```

The roots are determined as the eigenvalues of the corresponding companion matrix.

The command `fzero` is used to find the approximate solutions of nonlinear scalar equations.

■ Calculation of three solutions of equation $e^{-x^3} - 4x^2 = 0$.

```
>>   fzero(@(x)exp(-x^3) - 4 * x^2, 1)       ans = 0.4741
>>   fzero(@(x)exp(-x^3) - 4 * x^2, 0)       ans = -0.5413
>>   fzero(@(x)exp(-x^3) - 4 * x^2, -1)       ans = -1.2085
```

The input of the equation for the command `fzero` is made as an unnamed function. As it is obvious, it depends on the given initial value in the second argument, which solution is approximated. A combination of the bisection method with regula falsi (see 19.1.1.3, p. 951) is used for the iteration process.

4. Interpolation

Function fitting based on a given data set can be done either by interpolation (see 19.6.1, p. 982 or 19.7.1.1, p. 996) or by best approximating functions (see 19.6.2.2, p. 985). In Matlab the command `plot` is the most simple way to represent the data set graphically. The menu in the selfopening graphical window contains tools for editing the figure (linetypes, symbols, titles and legends), for exporting and printing *Basic Fitting* under *Tools*.

The Basic Fitting is a subroutine of Tools by which a variety of interpolation methods and best approximating polynomials of different degrees are offered. It is realized by the functions `interp1` and `polyfit`.

■ By the input

```
>>  plot([1.70045, 1.2523, 0.638803, 0.423479, 0.249091, 0.160321, 0.0883432, 0.0570776,
          0.0302744, 0.0212794]);
```

the data values are located at the data positions $1, 2, \ldots, 10$ and are graphically represented. **Fig.19.19a** shows the data set, the corresponding cubic interpolation spline and the best approximating polynomial of degree four.

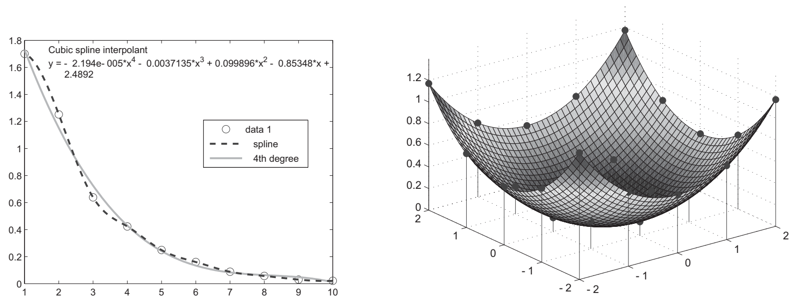


Figure 19.19 a) and b)

The Function `interp2` offers appropriate methods to interpolate the data given over a two dimensional rectangular grid (see 19.7.2.1, p. 998). To interpolate irregularly distributed data is served by calling `griddata`.

■ The command sequence

```
>>  [X, Y] = meshgrid(−2 : 1 : 2);    F = 4 − sqrt(16 − X.^2 − Y.^2);
>>  [Xe, Ye] = meshgrid(−2 : 0.1 : 2);    S = interp2(X, Y, F, Xe, Ye,′spline′);
>>  surf(Xe, Ye, S)
>>  hold on;    stem3(X, Y, F,′fill′)
```

realizes the bivariable cubic spline interpolation of the function $f(x, y) = \sqrt{16 - x^2 - y^2}$ given on a grid. The interpolation spline is evaluated on a finer rectangular grid. **Fig.19.19b** represents the interpolation function where the data points are also shown.

5. Numerical Integration

Numerical integration is available in Matlab by the procedures `quad` and `quadl`. Both procedures are based on the recursive application of interpolation quadratures with adaptive step-size selection. `quad` is based on the Simpson formula, and in `quadl` the Lobatto formulas of higher order are applied (see 19.3.2, p. 964). In the case of sufficiently smooth integrand and higher accuracy requirements `quadl` works more effectively than `quad`.

■ As the first example the approximation of the definite integral $I = \int_0^1 \frac{\sin x}{x} dx$ (Integral sine see 8.2.5,**1.** p. 513) is considered.

```
>>  format long;    [I, fwerte] = quad(@(x)(sin(x)./x), 0, 1)
```

```
Warning:  Divide by zero.    > In @(x)(sin(x)./x)   In quad at 62
I = 0.94608307007653   fwerte = 14
>>   format long;   [I, fwerte] = quadl(@(x)(sin(x)./x), 0, 1)
Warning:  Divide by zero.    > In @(x)(sin(x)./x)   In quadl at 64
I = 0.94608307036718   fwerte = 19
```

Both procedures obviously recognize the discontinuity of the integrand at the left endpoint of the interval, but the approximated value of the integral can be obtained without any difficulty. Based on the results of the same example in 19.3.4.2, p. 968 the number of function evaluations seems to be high but it is determined for the adaptive recursion.

```
>>   format long;   [I, fwerte] = quad(@(x)(sin(x)./x), 0, 1, 1e − 14)
I = 0.94608307036718   fwerte = 258
>>   format long;   [I, fwerte] = quadl(@(x)(sin(x)./x), 0, 1, 1e − 14)
I = 0.94608307036718   fwerte = 19
```

(The warning messages are not repeated here.) The determination of the accuracy of 10^{-14}, which is given as a further argument (the default is a 10^{-6} tolerance), makes the advantages of `quadl` obvious in this case.

■ The definite integral $I = \int_{-1000}^{1000} e^{-x^2} dx$ is to be determined.

```
>>   format long;   [I, fwerte] = quad(@(x)(exp(−x.^2)), −1000, 1000, 1e − 10)
I = 1.77245385094233   fwerte = 585
>>   format long;   [I, fwerte] = quadl(@(x)(exp(−x.^2)), −1000, 1000, 1e − 10)
I = 1.77245385090571   fwerte = 768
```

The flat shape of the integrand in a very wide part of the integration interval and the relatively steep peak at $x = 0$ make `quad` better in this case.

6. Numerical Solution of Differential Equations

Matlab offers several procedures to determine the numerical solutions of initial value problems of systems of first order ordinary differential equations. A standard procedure is `ode45`, in which Runge-Kutta methods of 4-th and 5-th order are applied with adaptive step-size selection (see 19.4.1.2, p. 969). To achieve higher accuracy the program `ode113` is more effective with implemented linear multi-step methods of predictor-corrector type (see 19.4.1.4, p. 971). Besides there are procedures which are especially effective for stiff systems of differential equations (see 19.4.1.5,**4.**, p. 973).

■ To solve the problem $y' = \frac{1}{4}(x^2 + y^2)$, $y(0) = 0$, by the Runge-Kutta method (see 19.4.1.2, p. 970) in the interval $[0, 1]$ the input is

```
>>   [x, y] = ode45(@(x, y)((x.^2 + y.^2)./4), [0 1], 0);   plot(x, y)
```

The shape of the resulted solution is represented in **Fig.19.20a**.

■ To solve the special Lorenz-system (see ■ in 17.2.4.3, p. 887)

$$x_1' = 10(x_2 - x_1), \quad x_2' = 28x_1 - x_2 - x_1x_3, \quad x_3' = x_1x_2 - \frac{8}{3}x_3$$

in the interval $0 \le t \le 50$ with initial conditions $x(0) = (0, 1, 0)^T$ the following commands are used:

```
>>   [t, x] = ode45(@(t, x)([10 ∗ (x(2) − x(1));
           28 ∗ x(1) − x(2) − x(1) ∗ x(3); x(1) ∗ x(2) − 8 ∗ x(3)/3]), [0 50], [0; 1; 0]);
>>   plot(x(:, 1), x(:, 3))
```

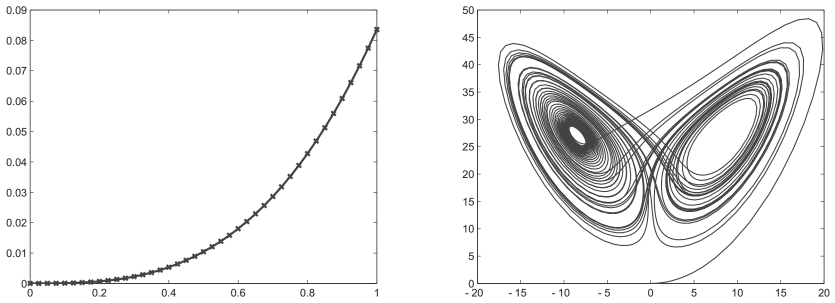


Figure 19.20 a) and b)

The last command creates a phase-diagram in the x_1, x_3 plane (see **Fig.19.20b**).

19.8.4.2 Mathematica

1. Tools for the Solution of Numerical Problems

The computer algebra system Mathematica offers a very effective tool that can be used to solve a large variety of numerical mathematical problems. The numerical procedures of Mathematica are totally different from symbolic calculations. Mathematica determines a table of values of the considered function according to certain previously given principles, similarly to the case of graphical representations, and it determines the solution using these values. Since the number of points must be finite, this could be a problem with " badly " behaving functions. Although Mathematica tries to choose more nodes in problematic regions, we have to suppose a certain continuity on the considered domain. This can be the cause of errors in the final result. It is advised to use as much information as possible about the problem under consideration, and if it is possible, then to perform calculations in symbolic form, even if this is possible only for subproblems.
In **Table 19.5**, we represent the operations for numerical computations:

Table 19.5 Numerical operations

`NIntegrate`	calculates definite integrals
`NSum`	calculates sums $\sum\limits_{i=1}^{n} f(i)$
`NProduct`	calculates products
`NSolve`	numerically solves algebraic equations
`NDSolve`	numerically solves differential equations

After starting Mathematica the " Prompt " *In[1]* := is shown; it indicates that the system is ready to except an input. Mathematica denotes the output of the corresponding result by *Out[1]*. In general: The text in the rows denoted by *In[n]* := is the input. The rows with the sign *Out[n]* are given back by Mathematica as answers. The arrow −> in the expressions means, e.g., replace x by the value a.

2. Curve Fitting and Interpolation

1. Curve Fitting Mathematica can perform the fitting of chosen functions to a set of data using the least squares method (see 6.2.5, p. 454ff.) and the approximation in mean to discrete problems (see 19.6.2.2, p. 986). The general instruction is:

$$\texttt{Fit}[\{y_1, y_2, \ldots\}, funkt, x]. \qquad (19.291)$$

Here the values y_i form the list of data, $funkt$ is the list of the chosen functions, by which the fitting should be performed, and x denotes the corresponding domain of the independent variables. If $funkt$ is chosen, e.g., as `Table`$[x\hat{}i, \{i, 0, n\}]$, then the fitting will be made by a polynomial of degree n.

■ Let the following list of data be given:

In[1] $:= l = \{1.70045, 1.2523, 0.638803, 0.423479, 0.249091, 0.160321, 0.0883432, 0.0570776, 0.0302744, 0.0212794\}$

With the input

In[2] $:= f1 =$ `Fit`$[l, \{1, x, x\hat{}2, x\hat{}3, x\hat{}4\}, x]$

it is supposed that the elements of l are assigned to the values $1, 2, \ldots, 10$ of x. The result is the following approximation polynomial of degree four:

Out[2] $= 2.48918 - 0.853487x + 0.0998996x^2 - 0.00371393x^3 - 0.0000219224x^4$

With the command

In[3] $:=$ `Plot`[`ListPlot`$[l, \{x, 10\}], f1, \{x, 1, 10\},$ `AxesOrigin`$\text{->} \{0, 0\}]$

a representation of the data and the approximation curve given in **Fig. 19.21a** can be obtained.

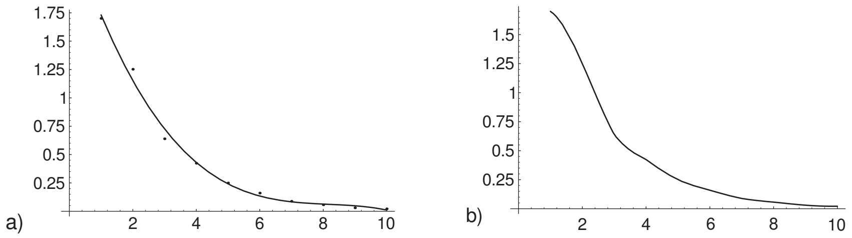


Figure 19.21

For the given data this is completely satisfactory. The terms are the first four terms of the series expansion of $e^{1-0.5x}$.

2. Interpolation Mathematica offers special algorithms for the determination of interpolation functions. They are represented as so-called `interpolating function` objects, which are formed similarly to pure functions. The directions for using them are in **Table 19.6**. Instead of the single function values y_i a list of function values and values of specified derivatives can be given at the given points.

■ With *In[3]* $:=$ `Plot`[`Interpolation`$[l][x], \{x, 1, 10\}]$ one gets **Fig. 19.21b**. Obviously Mathematica gives a precise correspondence to the data list.

Table 19.6 Commands for interpolation

`Interpolation`$[\{y_1, y_2, \ldots\}]$	gives an approximation function with the values y_i for the values $x_i = i$ as integers
`Interpolation`$[\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}]$	gives an approximation function for the point–sequence (x_i, y_i)

3. Numerical Solution of Polynomial Equations

As shown in 20.3.2.1, p. 1038 Mathematica can determine the roots of polynomials numerically. The command is `NSolve`$[p[x] == 0, x, n]$, where n prescribes the accuracy by which the calculations should be done. If n is omitted, then the calculations are made to machine accuracy. The complete solution is got, i.e., m roots, if the input polynomial is of degree m.

■ *In[1]* $:=$ `NSolve`$[x\hat{}6 + 3x\hat{}2 - 5 == 0]$

Out[1] $= \{x\text{->} -1.07432\}, \{x\text{->} -0.867262 - 1.15292\mathtt{I}\}, \{x\text{->} -0.867262 + 1.15292\mathtt{I}\},$

$\{x\text{-}> 0.867262 - 1.15292\texttt{I}\}, \{x\text{-}> 0.867262 + 1.15292\texttt{I}\}, \{x\text{-}> 1.07432\}\}.$

4. Numerical Integration

For numerical integration Mathematica offers the procedure `NIntegrate`. Differently from the symbolical method, here it works with a table of values of the integrand. Two improper integrals are considered (see 8.2.3, p. 506) as examples.

■ **A:** *In[1]* := `NIntegrate`[`Exp`[$-x$^2], {x, $-$`Infinity`, `Infinity`}]   *Out[1]* = 1.77245.

■ **B:** *In[2]* := `NIntegrate`[1/x^2, {x, -1, 1}]

Power::infy: Infinite expression $\frac{1}{0}$ encountered.
NIntegrate::inum: Integrand ComplexInfinity is not numerical at$\{x\} = \{0\}$.

Mathematica recognizes the discontinuity of the integrand at $x = 0$ in example **B** and gives a warning. Mathematica applies a table of values with a higher number of nodes in the problematic domain, and it recognizes the pole. However, the answer can be still wrong.

Mathematica applies certain previously specified options for numerical integration, and in some special cases they are not sufficient. The minimal and the maximal number of recursion steps, by which Mathematica works in a problematic domain, can be determined with parameters `MinRecursion` and `MaxRecursion`. The default options are always 0 and 6. If these values are increased, then although Mathematica works slower, it gives a better result.

■ *In[3]* := `NIntegrate`[`Exp`[$-x$^2], {x, -1000, 1000}] Mathematica cannot find the peak at $x = 0$, since the integration domain is too large, and the answers is:

NIntegrate::ploss:
Numerical integration stopping due to loss of precision. Achieved neither the requested PrecisionGoal nor AccuracyGoal; suspect one of the following: highly oscillatory integrand or the true value of the integral is 0.
Out[3] = $1.34946 \cdot 10^{-26}$

If the requirement is

In[4] := `NIntegrate`[`Exp`[$-$x^2], {x, -1000, 1000}, `MinRecursion`$-$> 3, `MaxRecursion`$-$> 10],

then the result is

Out[4] = 1.77245

Similarly, a result closer to the actual value of the integral can be got with the command:

`NIntegrate`[fun, $\{x, x_a, x_1, x_2, \ldots, x_e\}$]. (19.292)

One can give the points of singularities x_i between the lower and upper limit of the integral to force Mathematica to evaluate more accurately.

5. Numerical Solution of Differential Equations

In the numerical solution of ordinary differential equations and also in the solution of systems of differential equations Mathematica represents the result by an `InterpolatingFunction`. It allows us to get the numerical values of the solution at any point of the given interval and also to sketch the graphical representation of the solution function. The most often used commands are represented in **Table 19.7**.

Table 19.7 Commands for numerical solution of differential equations

`NDSolve`[$dgl, y, \{x, x_a, x_e\}$]	computes the numerical solution of the differential equation in the domain between x_a and x_e
`InterpolatingFunction`[$liste$][x]	gives the solution at the point x
`Plot`[`Evaluate`[$y[x]/.\ lös$]], $\{x, x_a, x_e\}$]	scetches the graphical representation

■ Solution of a differential equation describing the motion of a heavy object in a medium with friction. The equations of motion in two dimensions are

$$\ddot{x} = -\gamma\sqrt{\dot{x}^2+\dot{y}^2}\cdot\dot{x}, \quad \ddot{y} = -g - \gamma\sqrt{\dot{x}^2+\dot{y}^2}\cdot\dot{y}.$$

The friction is supposed to be proportional to the velocity. If $g = 10, \gamma = 0.1$ are substituted, then the following command can be given to solve the equations of motion with initial values $x(0) = y(0) = 0$ and $\dot{x}(0) = 100, \dot{y}(0) = 200$:

```
In[1] := dg = NDSolve[{x''[t] == -0.1Sqrt[x'[t]^2 + y'[t]^2] x'[t], y''[t] == -10
-0.1Sqrt[x'[t]^2 + y'[t]^2] y'[t], x[0] == y[0] == 0, x'[0] == 100, y'[0] == 200},
{x, y}, {t, 15}]
```

Mathematica gives the answer by the interpolating function:

```
Out[1] = {{x-> InterpolatingFunction[{0., 15.}, <>],
y-> InterpolatingFunction[{0., 15.}, <>]}}
```

The solution

```
In[2] := ParametricPlot[{x[t], y[t]}/.dg, {t, 0, 2}, PlotRange-> All]
```

is represented as a parametric curve **(Fig. 19.22a)**.

`NDSolve` accepts several options which affect the accuracy of the result.
The accuracy of the calculations can be given by the command `AccuracyGoal`. The command `PrecisionGoal` works similarly. During calculations, Mathematica works according to the so-called `WorkingPre cision`, which should be increased by five units in calculations requiring higher accuracy.

The numbers of steps by which Mathematica works in the considered domain is prescribed as 500. In general, Mathematica increases the number of nodes in the neighborhood of the problematic domain. In the neighborhood of singularities it can exhaust the step limit. In such cases, it is possible to increase the number of steps by `MaxSteps`. It is also possible the prescribe `Infinity` for `MaxSteps`.

■ The equations for the Foucault pendulum are:

$$\ddot{x}(t) + \omega^2 x(t) = 2\Omega\dot{y}(t), \quad \ddot{y}(t) + \omega^2 y(t) = -2\Omega\dot{x}(t).$$

With $\omega = 1$, $\Omega = 0.025$ and the initial conditions $x(0) = 0$, $y(0) = 10$, $\dot{x}(0) = \dot{y}(0) = 0$ the solution is:

```
In[3] := dg3 = NDSolve[{x''[t] == -x[t] + 0.05y'[t], y''[t] == -y[t] - 0.05x'[t],
x[0] == 0, y[0] == 10, x'[0] == y'[0] == 0}, {x, y}, {t, 0, 40}]
Out[3] = {{x-> InterpolatingFunction[{0., 40.}, <>],
y-> InterpolatingFunction[{0., 40.}, <>]}}
```

With

```
In[4] := ParametricPlot[{x[t], y[t]}/.dg3, {t, 0, 40}, AspectRatio-> 1]
```

one gets **Fig. 19.22b**.

19.8.4.3 Maple

The computer algebra system Maple can solve several problems of numerical mathematics with the use of built-in approximation methods. The number of nodes, which is required by the calculations, can be determined by specifying the value of the global variable `Digits` as an arbitrary n. But it should be kept in mind that selecting a higher n than the prescribed value results in a lower speed of calculation.

1. Numerical Calculation of Expressions and Functions

After starting Maple, the symbol " Prompt " > is shown, which denotes the readyness for input. Connected in- and outputs are often represented in one row, separated by the *arrow operator* $\longrightarrow$.

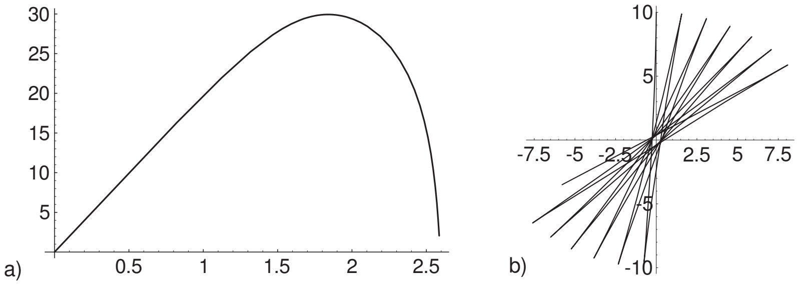


Figure 19.22

1. Operator `evalf` Numerical values of expressions containing built-in and user-defined functions which can be evaluated as a real number, can be calculated with the command

$$\texttt{evalf}(expr, n). \tag{19.293}$$

expr is the expression whose value should be determined; the argument n is optional, it is for evaluation to n digits accuracy. Default accuracy is set by the global variable `Digits`.

■ Prepare a table of values of the function $y = f(x) = \sqrt{x} + \ln x$.
First, the function is defined by the arrow operator:

$$> f := z \;\text{->}\; \texttt{sqrt}(z) + \ln(z); \longrightarrow f := z \;\text{->}\; \sqrt{x} + \ln x.$$

Then the required values of the function can be got with the command `evalf`$(f(x))$; , where a numerical value should be substituted for x.

A table of values of the function with steps size 0.2 between 1 and 4 can be obtained by

$$> \texttt{for } x \texttt{ from } 1 \texttt{ by } 0.2 \texttt{ to } 4 \texttt{ do print}(f[x] = \texttt{evalf}(f(x), 12)) \texttt{ od};$$

Here, it is required to work with twelve digits.
Maple gives the result in the form of a one-column table with elements in the form $f_{[3.2]} = 2.95200519181$.

2. Operator `evalhf`*(expr)***:** Beside `evalf` there is the operator `evalhf`. It can be used in a similar way to `evalf`. Its argument is also an expression which has a real value. It evaluates the symbolic expression numerically, using the hardware floating-point double-precision calculations available on the computer. A Maple floating-point value is returned. Using `evalhf` speeds up your calculations in most cases, but you lose the definiable accuracy of using `evalf` and `Digits` together. For instance in the problem in 19.8.2, p. 1004, it can produce a considerable error.

2. Numerical Solution of Equations

By using Maple equations or systems of equations can be solved numerically in many cases.
The command to do this is `fsolve`. It has the syntax

$$\texttt{fsolve}(eqn, var, option). \tag{19.294}$$

This command determines real solutions. If *eqn* is in polynomial form, the result is all the real roots. If *eqn* is not in polynomial form, it is likely that `fsolve` will return only one solution. The available options are given in **Table 19.8**.

Table 19.8 Options for the command `fsolve`

`complex`	determines a complex root (or all roots of a polynomial)
`maxsols` $= n$	determines at least the n roots (only for polynomial equations)
`fulldigits`	ensures that `fsolve` does not lower the number of digits used during computations
`intervall`	looks for roots in the given interval

■ **A:** Determination of all solutions of a polynomial equation $x^6 + 3x^2 - 5 = 0$. With

$> eq := x\hat{}6 + 3 * x\hat{}2 - 5 = 0:$

the result is

$> \texttt{fsolve}(eq, x); \longrightarrow -1.074323739,\ 1.074323739$

Maple determined only the two real roots. With the option `complex`, also the complex roots are obtained:

$> \texttt{fsolve}(eq, x, \texttt{complex});$

$-1.074323739, -0.8672620244 - 1.152922012\texttt{I}, -0.8672620244 + 1.152922012\texttt{I},$
$0.8672620244 - 1.152922012\texttt{I}, 0.8672620244 + 1.152922012\texttt{I}, 1.074323739$

■ **B:** Determination of both solutions of the transcendental equation $e^{-x^3} - 4x^2 = 0$.
After defining the equation

$> eq := \texttt{exp}(-x\hat{}3) - 4 * x\hat{}2 = 0$

the result is

$> \texttt{fsolve}(eq, x); \longrightarrow 0.4740623572$

as the positive solution. With

$> \texttt{fsolve}(eq, x, x = -2..0); \longrightarrow -0.5412548544$

Maple also determines the second (negative) root.

3. Numerical Integration

The determination of definite integrals is often possible only numerically. This is the case when the integrand is too complicated, or if the primitive function cannot be expressed by elementary functions. The command to determine a definite integral in Maple is `evalf`:

$$\texttt{evalf}(\texttt{int}(f(x), x = a..b), n). \tag{19.295}$$

Maple calculates the integral by using an approximation formula.

■ Calculation of the definite integral $\int_{-2}^{2} \mathrm{e}^{-x^3}\, dx$. Since the primitive function is not known, for the integral command the following answer is got

$$> \texttt{int}(\texttt{exp}(-x\hat{}3), x = -2..2); \longrightarrow \int_{-2}^{2} e^{-x^3}\, dx.$$

If

$> \texttt{evalf}(\texttt{int}(\texttt{exp}(-x\hat{}3), x = -2..2), 15);$

is typed,then the answer is 277.745841695583.
Maple used the built-in approximation method for numerical integration with 15 digits.

In certain cases this method fails, especially if the integration interval is too large. Then, another approximation procedure can be tried with the call to a library

```
readlib(`evalf/int`):
```

which applies an adaptive Newton method.

■ The input

$> \texttt{evalf}(\texttt{int}(\texttt{exp}(-x\hat{}2), x = -1000..1000));$

results in an error message. With

```
> readlib(`evalf/int`):
> `evalf/int`(exp(-x^2), x = -1000..1000, 10, _NCrule);
  1.772453851
```

the correct result is obtained. The third argument specifies the accuracy and the last one specifies the internal notation of the approximation method.

4. Numerical Solution of Differential Equations

Ordinary differential equations can be solved with the Maple operation `dsolve`. However, in most cases it is not possible to determine the solution in closed form. In these cases, it can be solved numerically, where the corresponding initial conditions have to be given.
In order to do this, the command `dsolve` is used in the form

$$\texttt{dsolve}(deqn, var, \texttt{numeric}) \tag{19.296}$$

with the option `numeric` as a third argument. Here, the argument *deqn* contains the actual differential equation and the initial conditions. The result of this operation is a procedure, and if it is denoted, e.g., by f, for using the command $f(t)$, the value of the solution function at the value t of the independent variable is returned.
Maple applies the Runge-Kutta method to get this result (see 19.4.1.2, p. 969). The default accuracy for the relative and for the absolute error is $10^{-\mathrm{Digits}+3}$. The user can modify these default error tolerances with the global symbols `_RELERR` and `_ABSERR`. If there are some problems during calculations, then Maple gives different error messages.

■ At solving the problem given in the Runge-Kutta methods in 19.4.1.2, p. 970, Maple gives:

$$> \ r := \texttt{dsolve}(\{\texttt{diff}(y(x), x) = (1/4) * (x\verb|^|2 + y(x)\verb|^|2), y(0) = 0\}, y(x), \texttt{numeric});$$

$$r := \texttt{proc `dsolve/numeric/result2`}\ (x, 1592392, [1])\ \texttt{end}$$

With

$$> \ \texttt{r}(0.5); \longrightarrow \{x(.5) = 0.5000000000, y(x)(.5) = 0.01041860472\}$$

we can determine the value of the solution, e.g., at $x = 0.5$.